

Srikanth Machiraju

Foreword by:

Ritesh Modi

Senior Technology Evangelist, Microsoft India

Learning Windows Server Containers

Build and deploy high-quality portable apps faster



Packt>

Learning Windows Server Containers

Build and deploy high-quality portable apps faster

Srikanth Machiraju



BIRMINGHAM - MUMBAI

Learning Windows Server Containers

Copyright © 2017 Packt Publishing

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 Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

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

First published: April 2017

Production reference: 1260417

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78588-793-2

www.packtpub.com

Credits

Author

Srikanth Machiraju

Copy Editors

Safis Editing
Dipti Mankame

Reviewer

Romeo Mlinar

Project Coordinator

Judie Jose

Commissioning Editor

Kartikey Pandey

Proofreader

Safis Editing

Acquisition Editor

Rahul Nair

Indexer

Pratik Shirodkar

Content Development Editor

Abhishek Jadhav

Graphics

Kirk D'Penha

Technical Editor

Gaurav Suri

Production Coordinator

Shantanu N. Zagade

Foreword

Containers have become an elegant way to virtualize system level services in the recent past. Coupled with microservices which provide well-bounded, self-contained, application services, containers bring a revolutionary shift to the way solutions are built and deployed on computer infrastructure. Docker has been a favorite way to build containers and container clusters on open source. The ability to pack hundreds or even thousands of these containers into a physical machine provides a great way to deploy services in an optimal and scalable fashion. The ability to use different operating systems in each of the containers (such as, Linux, Windows) along with any needed functionality packed together is a convenience that contributes to building modular systems. Microsoft Windows and Azure cloud platforms have matured over time and plays a critical role in building systems addressing the needs of digital transformation currently underway in enterprise IT systems across the world. Whether it is the enterprise application that handles millions (or billions) of mobile users or one that handles zillions of IoT sensors (connected cars, refrigerators, or whatever you have), it can now be built using Windows as a platform using the containers model and hosted on Azure cloud to provide the necessary scale.

Deployment and frequent releases is another area which has been revolutionized by container technology. DevOps, the practice that bring together development, deployment, and operations into a seamless continuum by making *infrastructure as code* a reality, is a part of the agile way of building solutions today. By using the concept of containers, enterprise IT teams can effortlessly and efficiently deploy solutions to immutable infrastructure.

This book deals with all the above aspects and is a timely addition to the windows and Azure developer's toolkit to understand windows containers and their usage in building different types of systems. The integration of DevOps into the subject brings a well thought out addition that helps in implementing cutting edge application development practices.

The author is a well-seasoned developer and architect, who has significant experience building modern enterprise solutions for large customers and his practical approach in dealing with this complex subject shines through in every page of this book. Bringing his consulting background and enormous technical prowess to the task, he has detailed out approach to application development with windows containers which will help developers, architects and operation teams in building modern cloud based enterprise scale applications.

Starting with an introduction to containers and microservices, the author takes a developer on a profound journey that covers from building a simple *Hello World* container to advanced practical usage, such as building complex enterprise applications using SQL Server, Redis Cache, storage volumes, VSTS for continuous build and deployment, resource management, and insights.

Ritesh Modi

Senior Technology Evangelist

Microsoft India

About the Author

Srikanth Machiraju is an ardent techie, DevOps practitioner, and developer consultant on Microsoft Azure and .NET technologies. He works for Microsoft Global Services, Modern Apps team, Hyderabad India. Over the past 9 years he has worked as a lead consultant for design and development of cloud-based application, migrating legacy applications to cloud, corporate trainer on Azure technologies, and speaker at various user group meetings on Microsoft Technologies. He loves to teach and evangelize best practices and patterns in application development. He continues to explore building modern, smart, and cloud-born applications with more reachability and integration using Azure, IoT devices, Artificial Intelligence, deep learning, parallelism, and enterprise level security. When away from work he loves to cook for his family, explore and review new food joints, watch movies, swim, and play Xbox.

He is also working on another artwork called Developing Bots using Microsoft Bot Framework which is due for release by end of 2017. The book focuses on building smart and intelligent bots using Microsoft Bot framework and Azure Cognitive Services.

I would like to thank my family, specially my wife Sonia Madan for being unconditionally supportive throughout the journey of this book. I would also like to thank my mentors Vineet Bhatia and Phani Tipparaju from BrainScale for their guidance throughout my career without learning from them this achievement would not have been possible. It is the people above and many more I may not have listed here inspired me learn, achieve, and feel obligated to pass the knowledge and learnings to the willing.

About the Reviewer

Romeo Mlinar has been working as Microsoft Senior System Engineer. Professionally connected with computer technology for more than a decade. Passionately devoted with Microsoft products and technology, for instance, system center, planning and design of Active Directory, as well as Windows Server services, devoting special attention to virtualization (Hyper-V), which is his recent preoccupation. He bears large number of Microsoft industrial certificates. Since 2012, he is Microsoft's Most Valuable Professional (MVP) for Cloud and Datacenter Management. He is a regular speaker at various IT conferences in the region and abroad. Also, he is an IT Pro and Edu IT Pro User Group lead in Zagreb, Croatia. He spends his free time with people from the IT world, acquiring new knowledge, eagerly sharing it with others, while at the same time enjoying his life with his family.

He blogs at: <http://blog.mlinar.biz/>.

He was also a reviewer on *Hyper-V 2016 Best Practices* written by Romain Serre and Benedict Berger.

I would like to thank my wife Ana and to my lovely son Vito, for their kind and full support, their understanding, and encouragement. Without you guys I could not have followed my passion. You are the source of my inspiration and happiness.

www.PacktPub.com

For support files and downloads related to your book, please visit www.PacktPub.com.

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

At www.PacktPub.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 Packt books and eBooks.



<https://www.packtpub.com/mapt>

Get the most in-demand software skills with Mapt. Mapt gives you full access to all Packt books and video courses, as well as industry-leading tools to help you plan your personal development and advance your career.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Customer Feedback

Thanks for purchasing this Packt book. At Packt, quality is at the heart of our editorial process. To help us improve, please leave us an honest review on this book's Amazon page at: <https://www.amazon.com/dp/1785887939>.

If you'd like to join our team of regular reviewers, you can e-mail us at: customerreviews@packtpub.com. We award our regular reviewers with free eBooks and videos in exchange for their valuable feedback. Help us be relentless in improving our products!

Table of Contents

Preface	1
Chapter 1: Exploring Virtualization	8
Microsoft's history of virtualization	10
Understanding virtualization levels	11
Hardware/platform/server virtualization	12
Storage virtualization	13
Network virtualization	14
OS virtualization	14
Limitations of virtualization	15
Machine turn up time	15
Low resource utilization	16
Operational costs	16
Application packaging and deployment	17
Introduction to containerization	17
A few key implementations of containers	18
Docker	18
Docker architecture	19
Development life cycle	20
Docker's success stories	21
The road ahead for Dockers	21
Introduction to Windows Server Containers	22
A little background	22
Windows Server Container versions	23
Hyper-V Containers	24
Why another container type?	24
Containers terminology	25
Container host	25
Container OS image	26
Container images	26
Container registry	26
Dockerfile	27
Benefits of containers	27
Windows Server Containers on Azure	28
Comparing containerization and VM virtualization	30
Cluster management	31
Docker Swarm	31

Kubernetes	32
DC/OS	33
Tooling support	33
Visual Studio Tools for Docker	34
Visual Studio Code	34
Visual Studio Online	34
Docker for Windows	34
Docker Toolbox for Windows	35
Who else is working with containers?	35
Turbo	35
Rocket	36
Summary	36
Chapter 2: Deploying First Container	37
Preparing the development environment	37
Containers on Windows 10	39
Windows Server Containers on-premise	41
Windows Server Containers on Azure	42
Container options on Windows Server 2016 TP5	43
Create Windows Server 2016 TP5 on Azure	43
Installing base OS images and verifying installation	46
Windows Server Containers development	49
Pulling images from Docker Hub	49
Preparing a Windows Containers image	53
Running web application in Docker	54
Creating a container	57
Decoding image preparation	58
FROM	58
MAINTAINER	59
LABEL	59
RUN	59
COPY	60
EXPOSE	60
CMD	60
Starting and stopping Docker Service	61
Summary	61
Chapter 3: Working with Container Images	62
Listing images	63
Searching images	63
docker pull	66
docker run	68

Detached versus foreground mode	68
Name	69
Isolation technology	69
Overriding Dockerfile image defaults	70
docker build	71
Build context	71
Build Docker image	71
dockerignore	73
Docker tags	74
docker commit	75
docker exec	79
docker push	80
Cleaning up containers or images	84
Summary	85
Chapter 4: Developing Container Applications	86
Setting up the development environment	87
Understanding .NET Core and ASP.NET Core	89
ASP.NET Core architecture	89
Hosting ASP.NET applications	90
Developing an ASP.NET Core application	91
The Music Store application	92
Deploying ASP.NET Core application as Windows Server Container	93
Dockerizing the application	94
Connecting the dots	101
Test and run on a developer machine	103
Hosting ASP.NET Core on IIS in Windows Server Container	106
Developing ASP.NET 4.5 applications as Windows Server Containers	107
Dockerizing ASP.NET 4.5 Web Application	110
Connecting the dots	113
Windows Server Container networking	116
Summary	117
Chapter 5: Deploying Container Applications	119
Deploy Azure VMs using ARM	120
Configure VM for remote connectivity	126
Configuring remote Docker host	129
docker.pid	129
dockerd configuration options	130
Debug	130
max-concurrent-downloads	131

max-concurrent-uploads	131
Host	131
Security	131
Deploying containers remotely	132
Configuring load balancer	135
Deploy Hyper-V Containers	140
Pre-requisites	140
Steps to run Music Store as a Hyper-V Container	141
Dangling images	142
Summary	146
Chapter 6: Storage Volumes	147
Storage volumes	148
Docker volumes	151
Sharing volumes	154
Music Store—store images using volumes	155
Deleting volumes	159
Relational databases and SQL Server container	160
Music Store—store data using SQL Server container	164
Summary	167
Chapter 7: Redis Cache Containers	168
Creating Redis Cache container	169
Creating Redis image and container	172
Operating Redis Cache containers	173
Redis Desktop Manager	174
Publishing Redis container	176
Persistent Redis containers	176
Master-slave configuration	180
Integrating Redis containers with Music Store	184
Summary	189
Chapter 8: Container Network	190
Introduction to Windows networking	191
Windows Containers--network management stack	193
Configuring container networks	195
Static port mappings	201
Disconnecting containers from network	202
Limitations of WinNAT network implementation	203
Networking modes	204
Transparent	204

L2 bridge or L2 tunnel	206
Multiple container networks	210
Container network routing	211
Single node	212
Multinode	212
Multi-subnet deployment of Music Store	213
Managing Docker networks using Windows PowerShell for Docker	224
Summary	225
Chapter 9: Continuous Integration and Delivery	227
Introduction to Visual Studio Team Services	228
Continuous integration	232
Signing up for a VSTS account	232
Uploading Music Store to VSTS	236
Configuring automated builds	241
Why do we need a custom build agent?	242
Custom build agent	243
Queuing build	257
Continuous delivery	261
Configuring service principal name	264
Configure staging environment	265
Configure the production environment	270
Testing CI/CD	271
Summary	272
Chapter 10: Manage Resource Allocation and REST API	273
Container resource allocation	274
CPU resource allocation	276
Memory allocation	278
Network allocation	278
Insights and telemetry	279
Application Insights	279
Operations Management Site	284
Optimizing Dockerfiles	286
Optimizing image size	287
Optimize build speed	289
Ordering instructions	291
Docker REST API	292
List containers	293
Create container	295
Start/stop container	296

Removing a container	296
Docker .NET SDK	297
List containers	298
Creating a container	298
Starting a container	299
Stopping a container	299
Removing a container	299
Downloading an image	300
Summary	301
Chapter 11: Composite Containers and Clustering	302
Orchestrating multi-container deployment using docker-compose	303
A docker-compose file reference	307
Build options	308
Naming containers	309
Dependencies	310
Named volumes	312
Docker CLI options	313
Start/stop services	313
Building images	314
Creating containers	314
Executing commands	315
Killing commands	315
Pause or unpause	315
Scale	315
Cluster management	316
Docker Swarm architecture	316
Setting up a swarm cluster	319
Generate SSH key	322
Create swarm cluster on Azure Container Service	323
Docker Swarm features	334
Summary	336
Chapter 12: Nano Server	337
A brief introduction to Nano Server	338
Benefits of running workloads on Nano Server	341
Live migrations	341
Zero footprint	341
Package management	342
Reboots	342

Provisioning time	342
Server management	343
PowerShell core	343
Provisioning Nano Server on Azure	344
Provisioning Nano Server on Windows 10	349
Package management	354
Deploy .NET Core applications on Nano Server	362
Configuration management using PowerShell DSC	364
Nano containers	368
Summary	370
Index	371

Preface

Containers is the next breakthrough in building modern and cloud based applications, comparing it with its predecessors like VM virtualization one would realize that containerization is the fastest, most resource-efficient, scalable, and secure way of building application hosting environments we know so far. *Learning Windows Server Containers* take you through a long and profound journey of building containerized ASP.NET applications on latest windows server platforms using Docker command line and Docker REST API. The book shows you how to build and ship containers from one environment to other with less hassle during the continuous integration and delivery process. You will learn to build containerized applications using scalable storage containers, cache containers with isolation levels like in VMs. The book helps you build an ecosystem of container hosts, manage composite container deployments, and resource governance.

What this book covers

Chapter 1, *Exploring Virtualization*, teaches you different virtualization levels, challenges with each type of virtualization, containers as a virtualization platform, and benefits of running containerized applications, tooling support, other container platforms available in market today.

Chapter 2, *Deploying First Container*, teaches you to set up development environment, understand the Docker terminology, installing images from Docker Hub, create custom windows container images using Docker CLI and authoring Dockerfile.

Chapter 3, *Working with Container Images*, will introduce you to common container management tasks such as listing the containers, start/stop, cleaning up unused containers or images using Docker CLI on Windows Server environment.

Chapter 4, *Developing Container Applications*, teaches you to create and deploy ASP.NET Core Web applications using Visual Studio 2015, .NET Core, and C# to Windows Server 2016 Core as Windows Container using PowerShell and Docker CLI.

Chapter 5, *Deploying Container Applications*, teaches you to create Windows Server Container environment on Azure using Azure Resource Manager templates and Azure PowerShell, configure remote management for container hosts, deploy container applications remotely as Windows Containers and Hyper-V containers, configuring software load balancer and so on.

Chapter 6, *Storage Volumes*, talks about building file based storage based containers using Docker volumes and relational database containers using Microsoft SQL Server.

Chapter 7, *Redis Cache Containers*, teaches you to create persistent Redis Cache containers using Redis and storage volumes.

Chapter 8, *Container Network*, introduces you to Windows Container networks, different networking modes, building custom container networks using different networking modes and deploying containers on custom networks.

Chapter 9, *Continuous Integration and Delivery*, teaches you to build continuous integration and deployment pipelines for container applications using Visual Studio Team Services (TFS Online) on Azure, Docker Hub, and Git. You will learn to create a custom build server for building, packaging and releasing containers to windows container hosts.

Chapter 10, *Manage Resource Allocation and REST API*, teaches you to manage container resource utilization, create and manage containers using Docker REST API via Postman and C#, image optimization strategies and monitoring options available for containers and container hosts.

Chapter 11, *Composite Containers and Clustering*, teaches you to orchestrate multiple container deployments using Docker Compose, set up scaling for multicontainer environments and authoring Docker Compose service definition. Also, you will learn the concepts of cluster management using Docker Swarm and Azure Container Service.

Chapter 12, *Nano Server*, serves as an introduction to Windows Nano Server, building custom Nano Server images using PowerShell, deploying containers on Nano Server, working with Nano containers, and configuring Nano Server using PowerShell DSC.

What you need for this book

This book assumes basic knowledge of PowerShell, C#, .NET, ASP.NET 5, cloud computing, and Azure. The book will help your setup development environment on desktop operating system like Windows 10 (with Anniversary update) and deploying container applications on VMs running on-premise and on Azure. To practice building containerized applications on an on-premise environment like Windows 10, the host machine should have Hyper-V feature enabled. The book shows building virtual environments using Windows 10 built-in feature called Hyper-V but you can try any other desktop virtualization software like VMware or virtual box. The book also uses Visual Studio 2015 for application development, if you have hands-on experience working with Visual Studio 2015 it will easy to execute the samples. Following is the basic hardware configuration for running Hyper-V and Visual Studio 2015:

- CPU: 1.6 GHz or faster processor, 4 cores
- 64-bit processor with Second Level Address Translation (SLAT) – (for Hyper-V only)
- CPU support for VM Monitor Mode Extension (VT-c on Intel CPU's) – (for Hyper-V only)
- Minimum of 8 GB RAM
- Disk space: 80 GB

Apart from the these, you will also need an Azure subscription for creating container environments on Azure. Microsoft offers a free subscription which can be used for 30 days in-case you do not have a paid subscription. The software requirements for the book are as follows:

- Visual Studio 2015 (Community Edition or above)
- Windows 10 with Anniversary update
- SQL Server Management Studio
- Redis Desktop Manager
- Postman

Internet connectivity is required to install any packages for application development, download images (ISO files) for windows server 2016 or source code from GitHub repository.

Who this book is for

The primary target audience for this book would be developers who would like to use Windows Server Containers to build portable apps that can run anywhere (laptop, server, and public or private cloud) without little or no changes to the code. Developers will be able to build and ship high-quality applications. As Windows Containers has a broad impact on developers and administrators alike, this book will also help IT professionals or DevOps Engineers prepare infrastructure which is easy to use and maintain. IT professionals will be able to optimize resource utilization by increasing the density of applications per machine. The concepts discussed in this book also help DevOps develop a container mindset, establish practices around publishing developed code as containers from development environment to production easily.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "The `remove()` method finds the first instance of the element (passed an argument) and removes it from the list."

Any command-line input or output is written as follows:

```
docker search microsoft
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "Click on **Test Connection** and ensure the connection is successful."



Warnings or important notes appear in a box like this.



Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book-what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail feedback@packtpub.com, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this book from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

You can download the latest code samples from the code repository belonging to this book from the author's code repository at <https://github.com/vishwanathsrikanth/learningwsc>.

The code bundle for the book is also hosted on Packt's GitHub repository at <https://github.com/PacktPublishing/Learning-Windows-Server-Containers>. We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1

Exploring Virtualization

In this highly competitive and rapidly changing world, for enterprises to be at the leading edge a highly reliable, cost effective, and infinitely scalable IT infrastructure is required. It is very important for enterprises to adapt to changing customer needs, fail fast, learn, and reinvent the wheel. Ever since hardware costs have come down the emphasis has shifted to making the most out of the capital investments made on physical infrastructure or reducing the amount of investments to build or rent new infrastructure. This fundamentally means running more applications/services out of the existing IT landscape.

Virtualized infrastructure solves the preceding problems and it caters to all the IT needs of a modern enterprise. Virtualization provides an abstraction over compute, storage, or networking resources and provides a unified platform for managing the infrastructure. Virtualization facilitates resource optimization, governance over costs, effective utilization of physical space, high availability of **line-of-business (LOB)** applications, resilient systems, infinite scalability, fault-tolerance environments, and hybrid computing.

The following are a few more features of virtualization:

- Virtualization is a software which when installed on your IT infrastructure allows you to run more **virtual machines (VMs)** in a single physical server, thereby increasing the density of machines per square feet of area
- Virtualization is not just for enabling more computers, it also allows collaborating all storage devices to form a single large virtual storage space, which can be pooled across machines and provisioned on demand
- It also provides benefits of hybrid computing by enabling you to run different types of **operating systems (OSes)** in parallel, therefore catering to large and varied customers
- It centralizes the IT infrastructure and provides one place to manage machines and cost, execute patch updates, or reallocate resources on demand
- It reduces carbon footprint, cooling needs, and power consumption

Cloud computing is also an implementation of virtualization. Apart from virtualizing the hardware resources, the cloud also promises to offer rich services such as reliability, self-service, and Internet level scalability on a pay-per-use basis.

Due to reduced costs, today's VMs offered by public or private cloud vendors are highly powerful. But are our applications or services utilizing the server capacity effectively? What percentage of compute and storage are the applications actually using? The answer is very low. Traditional applications are not so resource heavy (except a few batch processing systems, big data systems with heavy scientific calculations, and gaming engines that fully utilize the PC's power). In order to provide high scalability and isolation to the customers we end up running many instances of the application in each VM with 10%-30% utilization. And also it takes substantial amounts of time to procure a machine, configure it for the application and its dependencies, make it ready to use, and of course the number of VMs that you can run on your private data center is limited to the physical space you own. Is it really possible to further optimize resource utilization but still have the same isolation and scalability benefits? Can we get more throughput out of our IT infrastructure than we get today? Can we reduce the amount of preparation work required to onboard an application and make it ready to use? Can we run more services using the same physical infrastructure? Yes, all of this is possible, and containerization is our magic wand.

Containerization is an alternative to VM virtualization from which enterprises can benefit from running multiple software components in a single physical/virtualized machine with the same isolation, security, reliability, and scalability benefits. Apart from effective utilization, containerization also promotes rapid application deployment capabilities with options to package, ship, and deploy software components as independent deployment units called **containers**.

In this chapter, we are going to learn:

- Levels of virtualization
- Virtualization challenges
- Containerization and its benefits
- Windows Server Containers
- Hyper-V Containers
- Cluster management
- Terminology and tooling support

Microsoft's history of virtualization

Microsoft's journey with VM/hardware virtualization began with its first hypervisor called **Hyper-V**. In the year 2008, Microsoft released Windows Server 2008 and 2008 R2 with Hyper-V role, which is capable of hosting multiple VMs inside a physical machine. Windows Server 2008 was available in different flavors such as **Standard**, **Enterprise**, and **Datacenter**. They all differ in the number of VMs or **guest OS** that can be hosted for free per server. For example, in Windows Server 2008 Standard edition you can run one guest OS for free and new guest OS licenses have to be purchased for running more VMs. Windows Server 2008 Datacenter edition comes with unlimited Windows guest OS licenses.



Often when talking about virtualization we use words such as **host OS** and guest OS. Host OS is the OS running on a physical machine (or VM if the OS allows nested virtualization) that provides the virtualization environment. Host OS provides the platform for running multiple VMs. Guest OS refers to the OS running inside each VM.

At about the same time, Microsoft also shipped another hypervisor called Hyper-V Server 2008 with a limited set of features, such as Windows Server Core, CLI, and Hyper-V role. The basic difference between a server with role and Hyper-V versions is the licensing norms. Microsoft Hyper-V Server is a free edition and it allows you to run a virtualized environment by using existing Windows Server licenses. But of course you would miss the other coolest OS features of full Windows Server as host OS, such as managing the OS using neat and clean GUI. Hyper-V can only be interacted via remote interfacing and a CLI. Hyper-V server is a trimmed down version for catering to the needs of running a virtualized environment.

In the year 2008, Microsoft announced its cloud platform called **Windows Azure** (now **Microsoft Azure**), which uses a customized Hyper-V to run a multitenant environment of compute, storage, and network resources using Windows Server machines. Azure provides a rich set of services categorized as **Platform as a Service (PaaS)** and **Infrastructure as a Service (IaaS)** using the virtualized infrastructure spread across varied geographical locations.

In August 2012, Windows Server 2012 and 2012 R2 brought significant improvements to the server technology, such as improved multitenancy, private virtual LAN, increased security, multiple live migrations, live storage migrations, less expensive business recovery options, and so on.

Windows Server 2016 marks a significant change in the server OSes from the core. Launched in the second half of the year 2016, Windows Server 2016 has noteworthy benefits, especially for new trends such as containerization and thin OS:

- **Windows Server 2016 with Windows Server Containers and Hyper-V Containers:** Windows Server 2016 comes with a container role that provides support for containerization. With containers role enabled applications can be easily packaged and deployed as independent containers with a high degree of isolation inside a single VM. Windows Server 2016 comes with two flavors of containers: Windows Server Containers and Hyper-V Containers. Windows Server Containers run directly on Windows Server OS while Hyper-V Containers are the new thin VMs that can run on a Hyper-V. Windows Server 2016 also comes with enhanced Hyper-V capabilities such as nested virtualization.
- **Windows Server 2016 Nano Server:** Nano Server is a scaled down version of Windows Server OS, which is around 93% smaller than the traditional server. Nano Servers are designed primarily for hosting modern cloud applications called microservices in both private and public clouds.

Other virtualization platforms from Microsoft:

- Microsoft also offers a hosted virtualization platform called **Virtual PC** acquired from Connectix in 2003. Hosted virtualization is different from regular hypervisor platforms. Hosted virtualization can run on a 32/64-bit system such as traditional desktop PCs from Windows 7 OS and above, whereas the traditional hypervisors run on special hardware and 64-bit systems only.
- A few more virtualization solutions offered by Microsoft are hosted virtualizations called Microsoft Virtual Server 2005 for Windows Server 2003, **Application Virtualization (App-V)**, **MED-V** for legacy application compatibility, terminal services, and **virtual desktop infrastructure (VDI)**.

Understanding virtualization levels

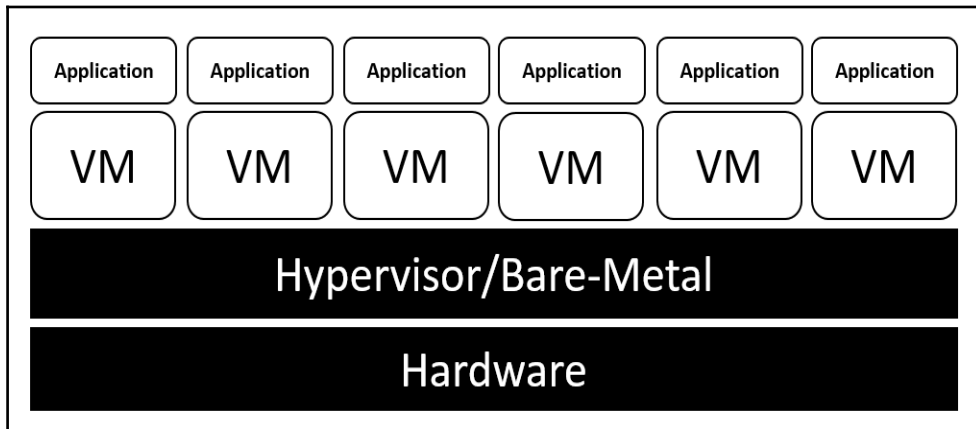
Depending on how the underlying infrastructure is abstracted away from the users and the isolation level, various virtualization technologies have evolved. The following sections discuss a few virtualization levels in brief, which eventually lead to containerization.

Hardware/platform/server virtualization

During the pre-virtualization era, a physical machine was considered a singleton entity that could host one operation system and could contain more than one application. Enterprises that run highly critical businesses or multitenant environments need isolation between applications. This limits from using one server for many applications. Hardware virtualization or VM virtualization helped to scale out single physical servers as they host multiple VMs within a single server where each VM can run in complete isolation. Each VM's CPU and memory needs can be configured as per the application's demand.

A discrete software unit called hypervisor or **Virtual Machine Manager (VMM)** runs on top of virtualized hardware and facilitates server virtualization. Modern cloud platforms, both public and private, are the best examples of hardware virtualization. Each physical server runs an operation system called host OS, which runs multiple VMs each with their own OS called guest OS. The underlying memory and CPU of the host OS is shared across the VMs depending on how the VMs are configured while creating. Server virtualization also enables hybrid computing, which means the guest OS can be of any type, for example, a machine running Windows with Hyper-V role enabled can host VMs running Linux and Windows OSes (for example Windows 10 and Windows 8.1) or even another Windows Server OS. Some examples of server virtualization are VMware, Citrix XenServer, and MS Hyper-V.

In a nutshell, this is what platform virtualization looks like:

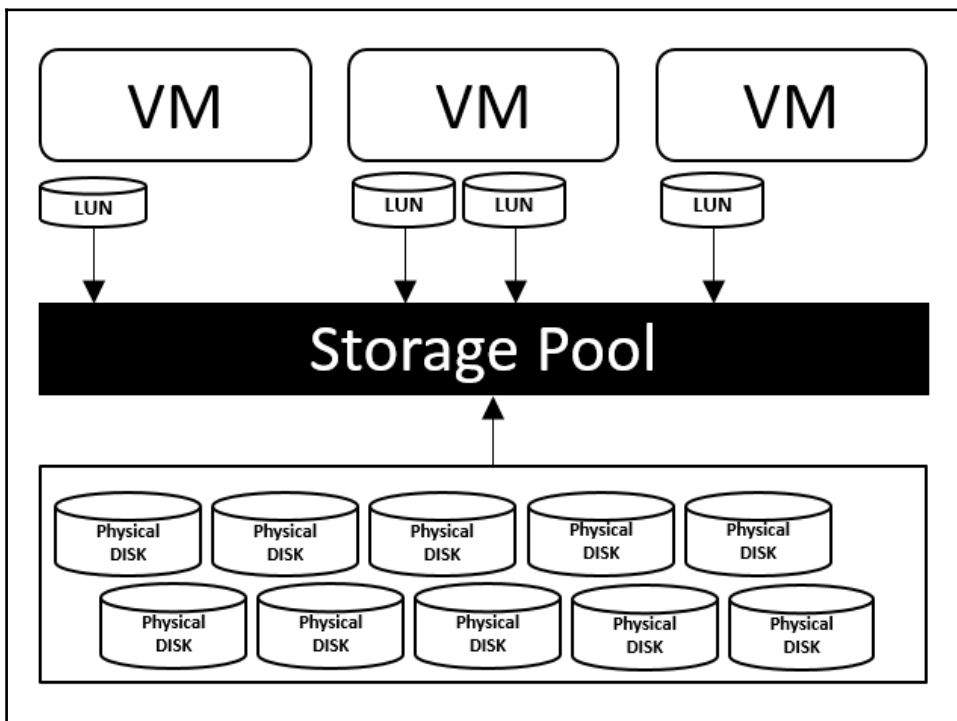


Storage virtualization

Storage virtualization refers to pooling of storage resources to provide a single large storage space, which can be managed from a single console. Storage virtualization offers administrative benefits such as managing backups, archiving, on demand storage allocation, and so on.

For example, Windows Azure VMs by default contain two disk drives for storage, but on demand we can add any number of disk drives to the VM within minutes (limited to the VM tier). This allows instant scalability and better utilization since we are only paying for what we use and expand/shrink as per demand.

This is what storage virtualization looks like:



Network virtualization

Network virtualization is the ability to create and manage a logical network of compute, storage, or other network resources. The components of a virtual network can be remotely located in the same or different physical networks across different geographical locations. Virtual networks help us create custom address spaces, logical subnets, custom network security groups for configuring restricted access to a group of nodes, custom IP configuration (few applications demand static IPs or IPs within a specific range), domain defined traffic routing, and so on.

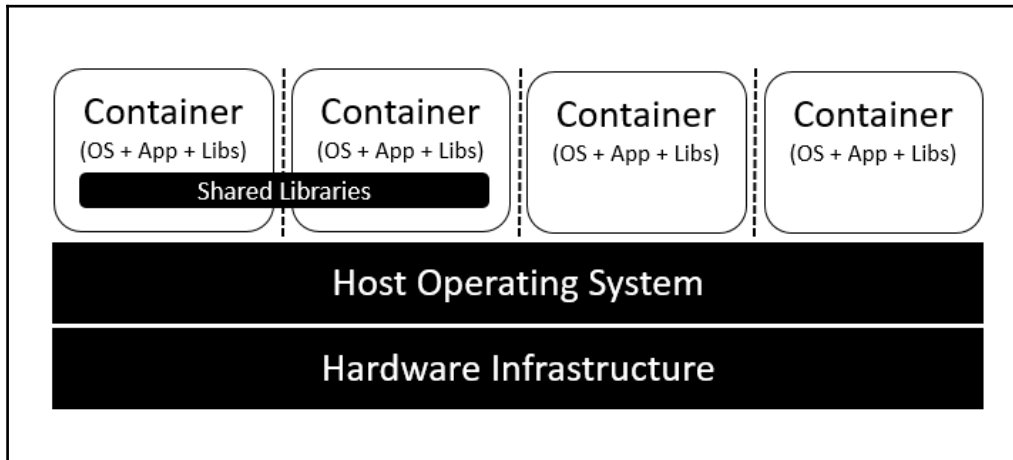
Most of the LOB applications demand logical separation between business components for enhanced security, isolation, and scalability needs. Network virtualization helps build the isolation configuring subnet level security policies, restrict access to logical subnets or nodes using **access control list (ACL)**, and restrict inbound/outbound traffic using custom routing without running a physical network. Public cloud vendors provide network virtualization on pay per use basis for small to medium scale business who cannot afford running a private IT infrastructure. For example, Microsoft Azure allows you to create a virtual network with network security boundaries, secure VPN tunnel to connect to your personal laptops, or on-premise infrastructure, high bandwidth private channels, and so on using pay-per-use pricing. You can run your applications on cloud with tight security among nodes using logical separation without even investing on any network devices.

OS virtualization

The topic of this book is associated with OS virtualization. OS virtualization enables the kernel to be shared across multiple processes inside a single VM with isolation. OS virtualization is also called user-mode or user-space virtualization as it is one level up from the kernel. Individual user-space instances are called containers. The kernel provides all the features for resource management across containers.

This is highly helpful while consolidating a set of services spread across multiple servers into a single server. Few benefits of OS virtualization are high security due to reduced surface of contact for a breach or viruses, better resource management, easy migration of applications or services across hosts, and also instant and dynamic load balancing. OS virtualization does not require any hardware support, so it is easy to implement than other technologies. The most recent implementations of OS virtualization are Linux LXC, Docker, and Windows Server Containers.

This is what OS virtualization looks like:



Today's containers are not yet cross platform, which means a Linux container cannot be directly ported to Windows. Containers being an OS virtualization are tied up to the kernel features, this makes it difficult to pursue cross platform portability.

Limitations of virtualization

There are a few limitations with the hardware or VM virtualization, which leads to containerization. Let's look at a few of them.

Machine turn up time

VMs run a fully-fledged OS. Every time a machine needs to be started, restarted, or shut down it involves running the full OS life cycle and booting procedure. A few enterprises employ rigid policies for procuring new IT resources. All of this increases the time required by the team to deliver a VM or to upgrade an existing one because each new request should be fulfilled by a whole set of steps. For example, a machine provisioning involves gathering the requirements, provisioning a new VM, procuring a license and installing OS, allocating storage, network configuration, and setting up redundancy and security policies.

Every time you wish to deploy your application you also have to ensure application specific software requirements such as web servers, database servers, runtimes, and any support software such as plugin drivers are installed on the machine. With teams obliged to deliver at light speed, the current VM virtualization will create more friction and latency.

Low resource utilization

The preceding problem can be partially solved by using the cloud platforms, which offer on-demand resource provisioning, but again public cloud vendors come up with a predefined set of VM configuration and not every application utilizes all allocated compute and memory.

In a common enterprise scenario every small application is deployed in a separate VM for isolation and security benefits. Further for ensuring scalability and availability identical VMs are created and traffic is balanced among them. If the application utilizes only 5-10% of the CPU's capacity, the IT infrastructure is heavily underutilized. Power and cooling needs for such systems are also high, which adds up to the costs. Few applications are used seasonally or by limited set of users, but still the servers have to be up and running. Another important drawback of VMs is that inside a VM OS and supporting services occupy more size than the application itself.

Operational costs

Every IT organization needs an operations team to manage the infrastructure's regular maintenance activities. The team's responsibility is to ensure that activities such as procuring machines, maintaining **SOX Compliance**, executing regular updates, and security patches are done in a timely manner. The following are a few drawbacks that add up to operational costs due to VM virtualization:

- The size of the operations team is proportional to the size of the IT. Large infrastructures require larger teams, therefore more costs to maintain.
- Every enterprise is obliged to provide continuous business to its customers for which it has to employ redundant and recovery systems. Recovery systems often take the same amount of resources and configuration as original ones, which means twice the original costs.
- Enterprises also have to pay for licenses for each guest OS no matter how little the usage may be.

Application packaging and deployment

VMs are not easily shippable. Every application has to be tested on developer machines, proper instruction sets have to be documented for operations or deployment teams to prepare the machine and deploy the application. No matter how well you document and take precautions in many instances the deployments fail because at the end of the day the application runs on a completely different environment than it is tested on which makes it riskier.

Let us imagine you have successfully installed the application on VM, but still VMs are not easily sharable as application packages due to their extremely large sizes, which makes them misfit for DevOps type work cultures. Imagine your applications need to go through rigorous testing cycles to ensure high quality. Every time you want to deploy and test a developed feature a new environment needs to be created and configured. The application should be deployed on the machine and then the test cases should be executed. In agile teams, release happens quite often, so the turnaround time for the testing phase to begin and results to be out will be quite high because of the machine provisioning and preparation work.

Choosing between VM virtualization or containerization is purely a matter of scope and need. It might not always be feasible to use containers. One advantage, for example, is in VM virtualization the guest OS of the VM and the host OS need not be the same. A Linux VM and a Windows VM can run in parallel on Hyper-V. This is possible because in VM virtualization only the hardware layer is virtualized. Since containers share the kernel OS of the host, a Linux container cannot be shipped to a Windows machine. Having said that, the future holds good things for both containers and VMs in both private and public clouds. There might be cases where an enterprise opts to use a hybrid model depending on scope and need.

Introduction to containerization

Containerization is an ability to build and package applications as shippable containers. Containers run in isolation in a user-mode using a shared kernel. A kernel is the heart of the operating system which accepts the user inputs and converts/translates them as processing instructions for CPU. In a shared kernel mode containers do the same as what VMs do to physical machines. They isolate the applications from the underlying OS needs. Let's see a few key implementations of this technology.

A few key implementations of containers

Some of the key implementations of containers are as follows:

- The word *container* has been around since 1982 with the introduction of chroot by Unix, which introduced process isolation. **Chroot** creates a virtual root directory for a process and its child processes, the process running under chroot cannot access anything outside the environment. Such modified environments are also called **chroot jails**.
- In 2000, a new isolation mechanism for FreeBSD (a free Unix like OS) was introduced by R&D Associates, Inc.'s owner, Derrick T. Woolworth, it was named jails. Jails are isolated virtual instances of FreeBSD under a single kernel. Each jail has its own files, processes, users, and super accounts. Each jail is sealed from other jails.
- Solaris introduced its OS virtualization platform called **zones** in the year 2004 with Solaris 10. One or more applications can run within a zone in isolation. Inter-zone communication was also possible using network APIs.
- In 2006, Google launched **process containers**, a technology designed for limiting, accounting, and isolating resource usage. It was later renamed to **control groups (cgroups)** and merged into the Linux kernel 2.6.24.
- In 2008, Linux launched its first out-of-the-box implementation of containers called **Linux containers (LXC)** a derivative of OpenVZ (OpenVZ developed an extension to Linux with the same features earlier). It was implemented using cgroups and **namespaces**. The cgroups allow management and prioritization for CPU, memory, block I/O, and network. Namespaces provided isolation.

Docker

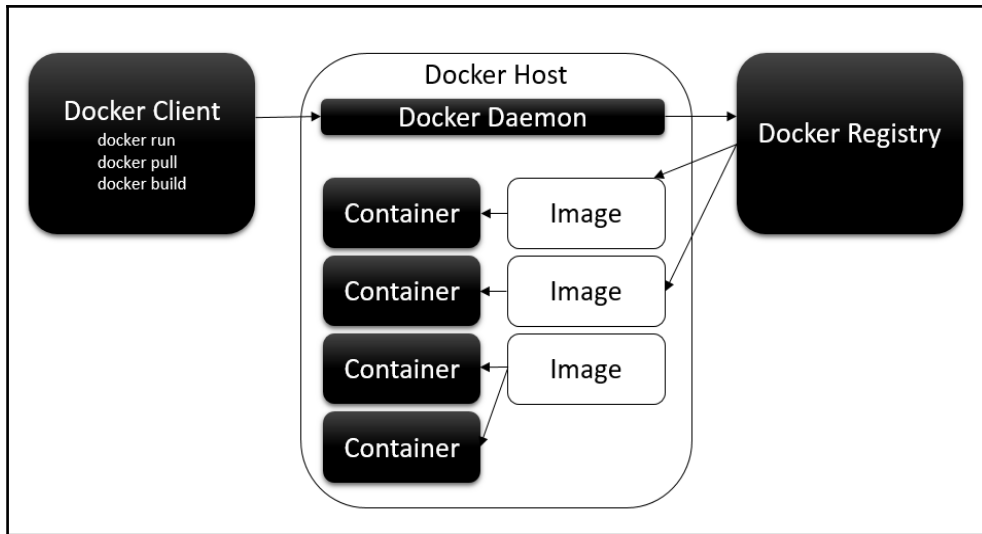
Solomon Hykes, CTO of **dotCloud** a PaaS (Platform as a Service) company, launched Docker in the year 2013, which reintroduced containerization. Before Docker, containers were just isolated processes and application portability as containers across discrete environments was never guaranteed. Docker introduced application packaging and shipping with containers. Docker isolated applications from infrastructure, which allowed developers to write and test the applications on traditional desktop OS and then easily package and ship it to production servers with less trouble.

Docker architecture

Docker uses client-server architecture. The **Docker daemon** is the heart of the Docker platform, and it should be present on every host, as it acts as a server. The Docker daemon is responsible for creating the containers, managing their life cycle, creating and storing images, and many other key things around containers. Applications are designed and developed as containers on developer's desktop and packaged as **Docker images**. Docker images are read-only templates that encapsulate the application and its dependent components. Docker also provides a set of base images that contain a pretty thin OS to start application development. **Docker containers** are nothing but instances of Docker images. Any number of containers can be created from an image within a host. Containers run directly on the Linux kernel in an isolated environment.

The Docker repository is the storage for Docker images. Docker provides both public and private repositories. Docker's public image repository is called **Docker Hub**, anyone can search the images in the public repository from Docker CLI or a web browser, download the image, and customize as per the application's needs. Since public repositories are not well suited for enterprise scenarios, which demand more security, Docker provides private repositories. Private repositories restrict access to your images for users within your enterprise; unlike Docker Hub, private repositories are not free. **Docker registry** is a repository for custom user images, users can pull any publicly available image or push to store and share across other users. The Docker daemon manages a registry per host too, when asked for an image the daemon first searches within the local registry and then the public repositories aka Docker Hub. This mechanism eliminates downloading images from the public repository each time.

Docker uses several Linux features to deliver the functionality. For example, Docker uses namespaces for providing isolation, cgroups for resource management, and union filesystem for making the containers extremely lightweight and fast. **Docker client** is the command-line interface, which is the only user interface for interacting with the Docker daemon. The Docker client and daemon can run out of a single system serving as both client and server. When on the server, users can use the client to communicate with the local server. Docker also provides an API that can be used to interact with remote **Docker hosts**. This can be seen in the following image:



Development life cycle

The Docker development life cycle can be explained with the help of the following steps:

1. Docker container development starts with downloading a base image from Docker Hub. Ubuntu and Fedora are a few images available from Docker Hub. An application can be containerized post application development too. It is not necessary to always start with Dockerizing the application.
2. The image is customized as per application requirement using a few Docker-specific instructions. These sets of instructions are stored in a file called Dockerfile. When deployed the Docker daemon reads the Dockerfile and prepares the final image.
3. The image can then be published to a public/private repository.
4. When users run the following command on any Docker host, the Docker daemon searches for images on the local machine first and then on Docker Hub if the images are not found locally. A Docker container is created if the image is found. Once the container is up and running, `[command]` is called on the running container:

```
$ docker run -i -t [imagename] [command]
```

Docker's success stories

Docker made enterprises life easy, applications can now be contained and easily shipped across hosts and distributed with less friction between teams. Docker Hub today offers 450,000 images publicly available for reuse. A few of the famous ones are nginx web server, Ubuntu, Redis, swarm, MySQL, and so on. Each one is downloaded by more than 10 million users. **Docker Engine** has been downloaded 4 billion times so far and still growing. Docker has enormous community support with 2,900 community contributors and 250 meet up groups. Docker is now available on Windows and Macintosh. Microsoft officially supports Docker through its public cloud platform Azure.

eBay, a major e-commerce giant, uses Docker to run their same day delivery business. eBay uses Docker in their **continuous integration (CI)** process. Containerized applications can be easily moved from a developer's laptop to test and then production machines seamlessly. Docker also enables the application to run alike on developer machines and also on production instances.

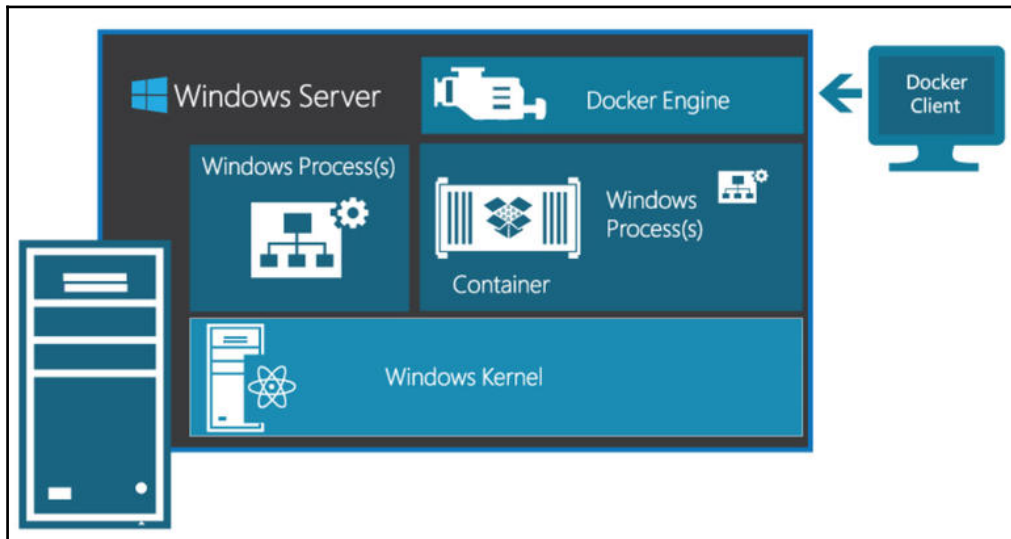
ING, a global finance services organization, faced nightmares with constant changes required to its monolithic applications built using legacy technologies. Implementing each change involved a laborious process of going through 68 documents to move a change to production. ING integrated Docker into its continuous delivery platform, which provided more automation capabilities for test and deployment, optimized utilization, reduced hardware costs, and saved time.

The road ahead for Dockers

Up to Docker release 0.9, containers were built using LXC, which is a Linux centric technology. Docker 0.9 introduced a new driver called **libcontainer** alongside LXC. The libcontainer is now a growing open source, open governed, non-profit library. The libcontainer is written using *Go language* for creating containers using Linux kernel API without relying on any close coupled features such as user-spaces or LXC. This means a lot to companies trending towards containerization. Docker has now moved out of being a Linux centric technology, in the future we might also see Docker adapting other platforms discussed previously, such as **Solaris Zones**, BSD jails, and so on. Libcontainer is openly available with contributions from major tech giants such as Google, Microsoft, Amazon, Red Hat, VMware, and so on. Docker being at its core is responsible for developing the core runtime and container format. Public cloud vendors such as Azure and AWS support Docker on their cloud platforms as first class citizens.

Introduction to Windows Server Containers

Windows Server Containers is an OS virtualization technology from Microsoft that is available from Windows Server 2016 onwards. Windows Server Containers are lightweight processes that are isolated from each other and have a similar view of the OS, processes, filesystem, and network. Technically speaking, Windows Server Containers are similar to Linux containers, which we discussed earlier. The only difference being the underlying kernel and workloads that run on these containers. The following image describes Windows Server Containers:



A little background

Here's a little background about Windows Server Containers:

- Microsoft's research team worked on a program called **Drawbridge** for bringing the famous container technologies to Windows Server ecosystem
- In the year 2014, Microsoft started working with Docker to bring the Docker technology to Windows
- In July 2015, Microsoft joined **Open Container Initiative (OCI)** extending its support for universal container standards, formats, and runtime

- In 2015, Mark Russinovich, CTO at Microsoft Azure, demonstrated the integration of containers with Windows
- In October 2015, Microsoft announced the availability of Windows Server with containers and Docker support in its new Windows Server 2016 Technical Preview 3 and Windows 10

Microsoft adapted Docker in its famous Windows Server ecosystem so developers can now develop, package, and deploy containers to Windows/Linux machines using the same toolset. Apart from support for the Docker toolset on Windows Server, Microsoft is also launching native support to work with containers using PowerShell and Windows CLI. Since OS virtualization is a kernel level feature there is no cross platform portability yet, which means a Linux container cannot be ported to Windows or vice versa.

Windows Server Container versions

Windows Server Containers are available on the following OS versions:

- **Windows Server 2016 Core (no GUI):** A typical Windows Server 2016 server with minimal installation. The Server Core version reduces the space required on disk and the surface of attack. CLI is the only tool available for interacting with the Server Core, either locally/remotely. You can still launch PowerShell using the `start powershell` command from the CLI. You can also call `get-windowsfeature` to see what features are available on Windows Server Core.
- **Windows Server 2016 (full GUI):** This is a full version of Windows Server 2016 with standard **Desktop Experience**. The server roles can be installed using **Server Manager**. It is not possible to convert Windows Server Core to full Desktop Experience. This should be a decision taken during installation only.
- **Windows 10:** Windows Server Containers are available for Windows 10 Pro and Enterprise versions (insider builds 14372 and up). The containers and Hyper-V/virtualization feature should be enabled before working with containers. Creating container and images with Windows 10 will be discussed in following chapters.
- **Nano Server:** Nano Server is a headless Windows Server 30 times or 93% smaller than the Windows Server Core. It takes far less space, requires fewer updates, high security, and boots much faster than the other versions. There is a dedicated chapter in this book on working with Nano Servers and Windows Containers.