

Getting Started with Docker Enterprise Edition on IBM Z

Lydia Parziale

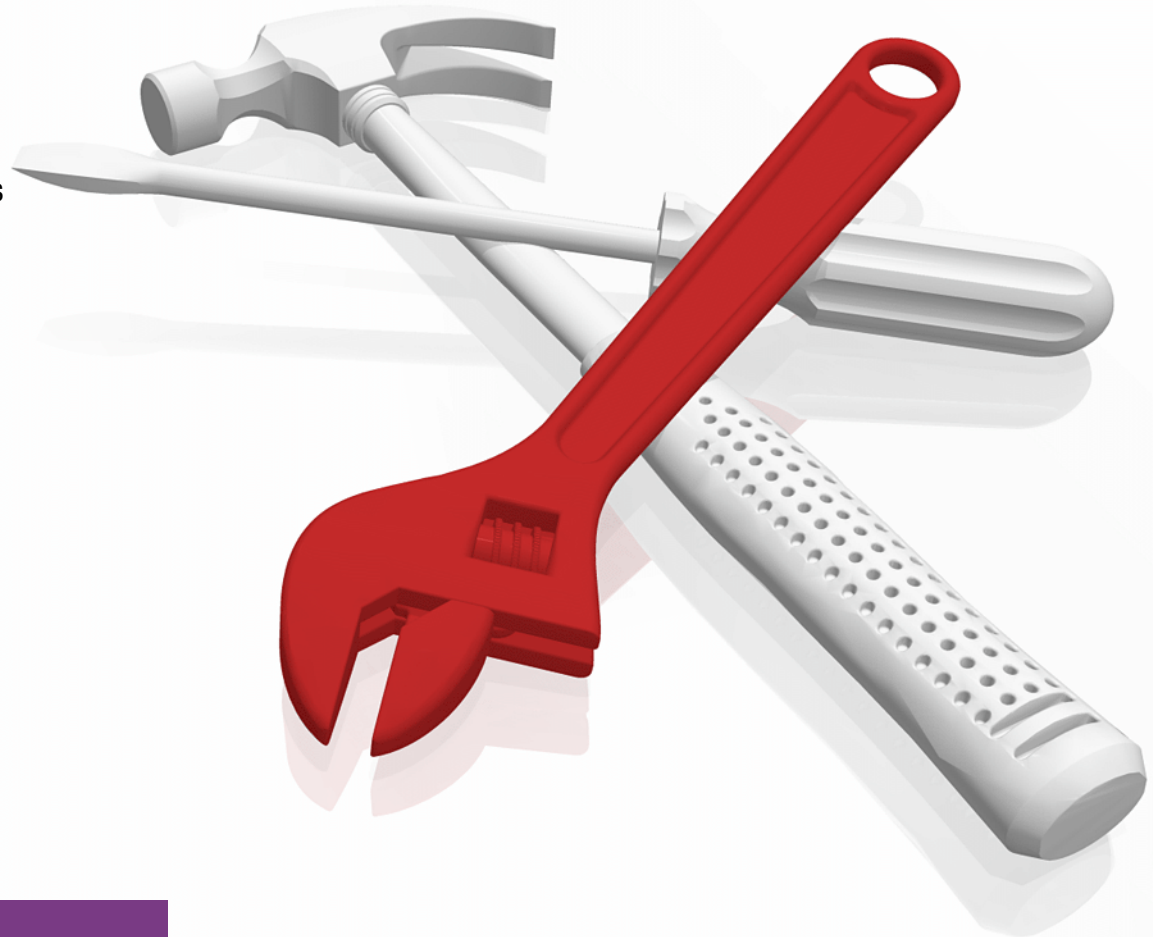
Eduardo Simoes Franco

Robert Green

Eric Everson Mendes Marins

Mariana Roveri

Nilton Carlos Dos Santos



IBM Z



International Technical Support Organization

**Getting Started with Docker Enterprise Edition on IBM
Z**

March 2019

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

First Edition (March 2019)

This edition applies to Docker Enterprise Edition 2.0 engine 17.06 or higher and z/VM 6.4 and 7.1

© Copyright International Business Machines Corporation 2019. All rights reserved.

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

Contents

Notices	vii
Trademarks	viii
Preface	ix
Authors	ix
Now you can become a published author, too!	xi
Comments welcome	xi
Stay connected to IBM Redbooks	xi
Chapter 1. Introduction	1
1.1 Why Docker on IBM Z	2
1.1.1 IBM Z	2
1.1.2 Linux ONE	4
1.1.3 Open Source and LinuxONE	4
1.2 Understanding the concepts of Docker EE on IBM Z	6
1.3 Docker overview	10
1.4 Security	13
1.4.1 Pervasive encryption	14
1.4.2 Linux encryption	16
1.4.3 Secure Service Containers	17
Chapter 2. Planning for Docker Enterprise Edition	19
2.1 Sample of Scenarios for containers on Linux on z	20
2.1.1 Scenario 1: All elements are running in the same subsystem	20
2.1.2 Scenario 2: Same LPAR with different hosts running Docker	20
2.1.3 Scenario 3: Dockers hosts are running on different LPARs and servers	21
2.2 Hardware and software prerequisites on IBM Z	23
2.2.1 Requirements for installing Docker on IBM Z	23
2.3 Lab resources and configuration	25
2.4 z/VM and SSI considerations	25
2.4.1 High availability and Docker	26
2.5 Docker editions	27
2.6 Container orchestration	27
2.6.1 Kubernetes	28
2.6.2 Swarm	31
Chapter 3. Installing and deploying Docker	35
3.1 Installing Docker	36
3.1.1 SuSE Linux	36
3.1.2 Red Hat Linux	41
3.1.3 Ubuntu Linux	46
3.2 Docker storage	48
3.3 Docker verification	61
3.4 Using hardware crypto for application containers	64
3.5 Moving Docker hosts by using z/VM SSI feature	78
3.6 Setting up swarm mode	79
3.7 Setting up Kubernetes	82
3.8 Kubernetes versus Docker Swarm	98
3.8.1 Kubernetes	99

3.8.2 Docker Swarm	99
Chapter 4. Basic Docker operations	101
4.1 Linux on Z commands	102
4.1.1 Overview	102
4.2 Basic Docker commands	105
4.2.1 Docker info	105
4.2.2 Docker ps	106
4.2.3 Docker rmi	107
4.2.4 Docker rm	107
4.2.5 Docker stop	108
4.2.6 Docker pull	108
4.2.7 Docker tag	109
4.2.8 Docker build	109
4.2.9 Docker run	110
4.2.10 Docker container port	110
4.2.11 Docker exec	110
4.2.12 Docker attach	111
4.2.13 Docker stop	111
4.2.14 Docker kill	112
4.2.15 Docker logs	112
4.2.16 Docker diff	113
4.2.17 Docker image	113
4.2.18 Docker network	114
4.2.19 Docker commit	115
4.2.20 Docker inspect	115
4.3 Swarm commands	116
4.3.1 Get information about your cluster	116
4.3.2 List nodes in swarm	117
4.3.3 Demote a node	118
4.3.4 Promote a node	118
4.3.5 Changing the availability of your nodes	119
4.3.6 Find worker join command	121
4.3.7 Find manager join command	121
4.3.8 Leave a swarm	121
4.4 Kubernetes commands	122
4.4.1 Getting the status of the cluster	122
4.5 Backing up a container	124
Chapter 5. Sample use cases	127
5.1 Database management systems	128
5.1.1 MongoDB	128
5.1.2 IBM Db2 database	132
5.2 IBM Watson Explorer Analytical Components	139
5.2.1 Preparing the image	140
5.2.2 Installing Watson Explorer AC	142
5.3 Docker in swarm mode	148
5.3.1 Removing node from swarm	148
5.3.2 Opening firewall ports on manager and worker nodes	149
5.3.3 Updating /etc/hosts file	151
5.3.4 Initializing swarm mode	151
5.3.5 Adding node manager to the cluster	152
5.3.6 Adding worker nodes to the manager node	152

5.3.7	Creating a Docker service.	152
5.3.8	Scaling a container up	154
5.3.9	Changing node availability to simulate a schedule maintenance	155
5.3.10	Promoting a worker node	158
5.3.11	Demoting a manager node	159
5.3.12	Running manager-only nodes	160
5.3.13	Scaling down a service	161
5.3.14	Considerations regarding the number of manager nodes.	161
5.4	Kubernetes	162
5.4.1	Adding a worker node to a Kubernetes cluster	162
5.4.2	Pause (cordon off) a worker node.	164
5.4.3	Temporarily remove a worker node for maintenance	165
5.4.4	Removing a worker node	165
5.5	Web services.	167
5.6	Tying it all together	169
5.6.1	Connecting to the Db2 container	169
5.6.2	Creating a collection	171
5.6.3	Configuring a Db2 crawler.	173
5.6.4	Mining the contents.	179

Notices

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

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

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

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

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

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

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

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

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

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

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

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


COPYRIGHT LICENSE:

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

Trademarks

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

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

DB2®	IBM z14™	System z®
Db2®	Passport Advantage®	Watson™
HiperSockets™	PR/SM™	WebSphere®
IBM®	Processor Resource/Systems Manager™	z/OS®
IBM Cloud™	Redbooks®	z/VM®
IBM LinuxONE™	Redpaper™	z/VSE®
IBM Watson®	Redbooks (logo)  ®	
IBM Z®		

The following terms are trademarks of other companies:

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Preface

What is the difference between a virtual machine and a Docker container?

A virtual machine (VM) is like a house. It is fully contained with its own plumbing and heating and cooling system. If you want another house, you build a new foundation, with new walls, new plumbing, and its own heating and cooling system. VMs are large. They start their own operating systems.

Containers are like apartments in an apartment building. They share infrastructure. They can be many different sizes. You can have different sizes depending on the needs. Containers “live” in a Docker host.

If you build a house, you need many resources. If you build an apartment building, each unit shares resources. Like an apartment, Docker is smaller and satisfies specific needs, is more agile, and more easily changed.

This IBM® Redbooks® publication examines the installation and operation of Docker Enterprise Edition on the IBM Z® platform. It includes the following chapters:

- ▶ Chapter 1, “Introduction” on page 1 explains how to run Docker containers on IBM Z servers.
- ▶ Chapter 2, “Planning for Docker Enterprise Edition” on page 19 provides important information that assists you to build a plan to install the Docker environment that uses an IBM Z and LinuxONE infrastructure.
- ▶ Chapter 3, “Installing and deploying Docker” on page 35 describes the Docker installation steps in two different operating systems (Linux Red Hat and SuSE).
- ▶ Chapter 4, “Basic Docker operations” on page 101 introduces you to some of the useful and frequently used Docker commands for system administrators and developers.
- ▶ Chapter 5, “Sample use cases” on page 127 describes some practical use cases that use Docker containers.

Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Lydia Parziale is a Project Manager for the IBM Redbooks publication team in Poughkeepsie, New York, with domestic and international experience in technology management, including software development, project leadership, and strategic planning. Her areas of expertise include business development and database management technologies. Lydia is a certified PMP and an IBM Certified IT Specialist with an MBA in Technology Management and has been employed by IBM for over 25 years in various technology areas.

Eduardo Simoes Franco is an IT Specialist and Technology Consultant at IBM. He has more than 20 years of experience with IT Solutions, projects, and infrastructure support. He has held technical and management positions at several large corporations where he held a various positions in servers support, as a network analyst, security officer, IT coordinator, and consultant. Currently, he supports large IBM clients worldwide on Docker, virtualization, and Linux on IBM Z platform.

Robert Green is a Systems Architect, supporting the Communications and Consolidated System Integrator accounts in the United States, where he works with customers on their IT infrastructure requirements. Robert has been in the information technology industry working with primarily with mainframe systems for 30 years. He has been with IBM for 22 years in various roles in the labs and in technical sales. Robert holds a B.S. degree in Business Administration with a specialization in Finance, and a Master of Science in Computer and Information Sciences, both from the University of South Alabama. Robert has co-authored the IBM Redbooks publication *IBM Tivoli Directory Server for z/OS*, SG24-7849 and the IBM Redpaper™ publication, *Cloud Computing and the Value of zEnterprise*, REDP-4763.

Eric Everson Mendes Marins is a Senior IT Specialist for IBM Global Technology and Services in Brazil. He has more than 16 years of experience in configuring Linux on IBM Z. He is working as a Linux on IBM Z Architect for IBM Global Account (IGA) in Brazil and implementing several cloud solutions on IBM Z family. He is IBM L3 IT Specialist certified (Thought Leader) and an IBM L3 IT Architect certified (Expert). Also, he holds a degree in Computer Science from Faculdade Ruy Barbosa and a post-graduate degree in Computer Information Systems (Database Management). His areas of expertise include Linux, IBM z/VM®, Docker, high availability, IP networking, and server management. Eric has co-authored several IBM Redbooks publications, including *Advanced Networking Concepts Applied Using Linux on IBM System z*, SG24-7995; *Security for Linux on System z*, SG24-7728; and *Scale up for Linux on IBM Z*, REDP-5461.

Mariana Roveri is a Senior IT Specialist for IBM Global Technology and Services in Brazil. She has more than 9 years of experience working as a Database Administrator. She is working as a Database Administrator for the American Express account. She holds a degree in Information Technology from UNICAMP and post-graduate degree MBA in Database Management. She is IBM L1 IT Specialist and her areas of expertise include databases (Oracle, MongoDB, and IBM DB2®), high availability for databases, and Docker.

Nilton Carlos Dos Santos is a Senior IT Specialist with more than 20 years of experience in different roles, such as Developer, Customer Support, Systems Architect, and Client Technical Specialist. Currently, Nilton is working with IBM Watson® products for IBM Australia, where he moved after 9 years working in IBM Brazil. He holds a bachelor degree in Computer Science and a post-graduate degree in Computer Network Systems. Nilton has also co-authored other IBM Redbooks publications, including *IBM Software Defined Infrastructure for Big Data Analytics Workloads*, SG24-8265; *IBM Platform Computing Solutions for High Performance and Technical Computing Workloads*, SG24-8264; and the IBM Redpaper publication *IBM Platform Computing Cloud Services*, REDP-5214.

Thanks to Tom Ambrosio and William Lamastro of the Competitive Project Office (CPO), and Robert Haimowitz from IBM Redbooks publications for their support and contributions to this publication.

Now you can become a published author, too!

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

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

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



Introduction

As a new Docker user, you must develop an understanding of the software and hardware platform that runs the Docker product. Containers are designed to make better use of hardware and its host operating system.

This chapter introduces you to the concepts of how to run Docker containers on IBM Z servers. This technology is transforming the application world and bringing a new vision to deploy applications in production today.

This chapter includes the following topics:

- ▶ 1.1, “Why Docker on IBM Z” on page 2
- ▶ 1.2, “Understanding the concepts of Docker EE on IBM Z” on page 6
- ▶ 1.3, “Docker overview” on page 10
- ▶ 1.4, “Security” on page 13

1.1 Why Docker on IBM Z

With the adoption of IBM Z Systems and Linux ONE servers, you can take advantage of the benefits from a hardware platform that includes specialized processors, cryptographic cards, availability, flexibility, and scalability. These systems are prepared to handle cloud, mobile, big data, blockchain, and analytics applications and provide a standard platform to run Docker instances. These systems can support up to 2 million Docker containers on a single machine, which maintains the highest levels of security and performance and does not require application server farms that can produce server sprawl.

This configuration requires an open source platform (as such IBM Z) that can provide high throughput and instantaneous data delivery while keeping systems and data secure. This configuration also needs a platform that allows unparalleled flexibility to achieve efficiencies, capabilities, reduced costs, and other benefits that ranked IBM Z technology as leader over several years.

For more information about Docker container technology, see [the Docker website](#).

1.1.1 IBM Z

IBM Z Systems and LinuxONE servers deliver the world's best security and reliability, with millisecond response times, hybrid cloud deployment, and performance at scale. The world's top companies rely on the IBM mainframe to protect their data with impeccable security, run blockchain and machine learning applications, and provide trusted digital experiences. IBM Z leads in several key areas that are described next.

Security

Data breaches are being reported at an increasing frequency for organizations large and small, which leads to a loss of trust for the organization and its customers. Organizations are becoming challenged to not only secure data, but prove to auditors and government regulators that customer data is secure (especially sensitive customer data that might be used for fraud or identify theft). If data is compromised, data must be encrypted so that it becomes worthless outside of the organization without encryption keys.

Pervasive encryption offers the ability to protect all data at rest and in flight, which provides protection from threats internally and externally, without requiring application changes. Encryption is built into the system hardware by design. The Central Processor Assist for Cryptographic Function (CPACF) is built into each core, which enables encryption of data without incurring more software overhead.

Policy-based data encryption enables pervasive or selective encryption, which helps an organization to meet their data security requirements, including General Data Protection Regulations (GDPR) requirements. It also provides a clear view for auditors into what is being encrypted. Whether it be data that is on disk, data that is moving from disk to memory, data that is going across a network, or is on tape for archive storage, data can be encrypted according to organization requirements and standards.

Reliability

In addition to security concerns, organizations need systems that they can rely on to remain up and operational. Any time that a system is unavailable translates to opportunity costs for an enterprise.

Opportunity costs lead to loss of revenue, and possibly added expense when an organization misses targets in terms of Service Level Agreements with their clients. The IBM Z provides record uptime with 99.999% availability (the Z stands for zero downtime), which provides the enterprise with a highly reliable platform.

The design of the IBM Z is focused on removing single points of failure. The failure of a single component does not bring down an entire system in most situations. In addition, most hardware components can be serviced or replaced concurrently, which eliminates downtime for the platform even further.

IBM Z also provides the capability of adding processing capacity concurrently. This addition might be in the form of temporary processing capacity, such as for the day, increasing capacity longer-term for a planned event, or permanently increasing capacity as the organization grows.

Performance

The latest generation of IBM Z, the IBM z14™, features a processor core that is rated at 5.2 GHz (14 nm FINFET Silicon-On-Insulator [SOI]) that does not require any liquid-nitrogen cooling assistance. The z14 can be configured with up to 170 processors cores.

IBM Z also features built-in processors that specially handle I/O operations, offloading this function so that CPUs can be used for operating system and application processing requirements. In addition, many other microprocessors are on various components that offload work from the CPUs for specific functions, such as network routing.

Virtualization

IBM Z servers feature virtualization that is built into the platform, beginning with IBM Processor Resource/System Manager (IBM PR/SM™) and the Logical Partition (LPAR). This most basic hardware virtualization is key to running multiple workloads on a single physical server.

IBM PR/SM allows for a single server to be divided into multiple LPARs (up to 85 on the z14). For each LPAR, an operating system can be installed, such as IBM z/OS®, z/VSE®, z/TPF or Linux, or a hypervisor can be installed. IBM Z supports running z/VM and Kernel-based Virtual Machine (KVM) as hypervisors.

On top of the hypervisor, virtual machines can be installed that are running Linux from distributors, including Red Hat, SuSE, and Ubuntu. z/VM can support hundreds to thousands of virtual machines that are running on a single IBM Z server.

Scalability

Servers are limited by the amount of processing capacity and memory that can be physically installed. Virtualization allows for better utilization of physical resources by allowing multiple servers to access the same resources that might otherwise be underutilized. However, capacity can be quickly exhausted as virtual servers are added to the same physical server in a scale-up fashion.

The only way to increase capacity is to adopt a scale-out strategy. This strategy leads to server sprawl, which uses more environmental factors, such as data center floor space, and power and cooling resources.

IBM Z overcomes this challenge with the ability to physically install 170 processors and up to 32 TB of addressable memory in a single-server footprint that occupies only two floor tiles of data center floor space. The current generation of IBM Z, the IBM z14 allows for 85 physical partitions, each of which can run the z/VM or KVM hypervisor.

z/VM can support hundreds of virtual machines, limited only by the amount of resources that are available in the LPAR. A single instance of z/VM can have up to 2 TB of memory. A single virtual machine that is running under z/VM can have up to 1 TB, which allows for the creation of many small (in terms of memory) virtual machines to large virtual machines.

Note: Memory in the IBM Z can be overcommitted; that is, the total size of all virtual machines can exceed the amount of memory that is installed in the server. However, the total amount of memory that is *in use* at any time by all virtual machines cannot exceed the amount of memory that is installed in the z/VM LPAR.

1.1.2 Linux ONE

IBM introduced a new server that is dedicated to running Linux virtual servers. LinuxONE servers offer the same inherent capabilities for IBM Z servers for organizations who want to run Linux in a scale-up architecture. It is an enterprise server that is designed for large data serving and transactional applications. The LinuxONE server provides the ability to potentially run thousands of Linux virtual servers under the z/VM and KVM hypervisors.

The LinuxONE server has as its foundation the Integrated Facility for Linux (IFL) central processor. IFLs are specialized central processors that are designed to run Linux, z/VM, and KVM only. LinuxONE is available in two models: the Emperor II, which can be configured with up to 170 IFLs and 32 TB of addressable memory, and the smaller Rockhopper II, which can be configured with up to 30 IFLs and 8 TB of addressable memory in a single server. LPARs have the same capabilities as described in “Virtualization” on page 3.

1.1.3 Open Source and LinuxONE

Organizations are embracing Open Source applications and software, and they want platforms that support this strategy. IBM provided the ability to run a Linux kernel distro on IBM Z from various providers, such as Red Hat, SuSE, and Ubuntu, and the ability to compile your own kernel by downloading the correct binaries for the architecture. In addition, middleware, including application servers and databases, were supported for many years on IBM Z and now LinuxONE.

KVM

System administration is an area that is perceived to be over-complicated on IBM Z. Organizations want systems that are agile and can be administered with existing or easily obtainable skills. The KVM is a hypervisor that is simple to use and was used on the x86 architecture platform for several years.

KVM administration skills are easily transferable to LinuxONE and IBM Z servers to enable organizations to more efficiently use their staff. With LinuxONE (and IBM Z), KVM works with QEMU to provide virtualization of hardware resources to virtual machines that are running under KVM, which allows guests to run unmodified.

Note: QEMU is used because no s390x-specific module is available for KVM.

OpenStack

IBM is committed to building its cloud on an open cloud architecture. OpenStack was enabled for z/VM since Juno (10/2013) and continues to be supported for LinuxONE (and IBM Z).

Docker on IBM Z

Docker takes hardware virtualization to the next level. Its strength is the ability to take an application along with any dependencies, such as binaries, and deploy it all as a single package. Instead of installing an operating system in a virtual machine, then installing any dependencies that an application or middleware title might need, and then installing the middleware and application, Docker allows everything to be deployed to a target platform as one complete package in the form of a container.

Containers can be built with multiple architectures included to ease in deployment in environments with multiple platforms. The deployment process is simplified, which allows application developers and system administrators to focus on more productive areas (see Figure 1-1).

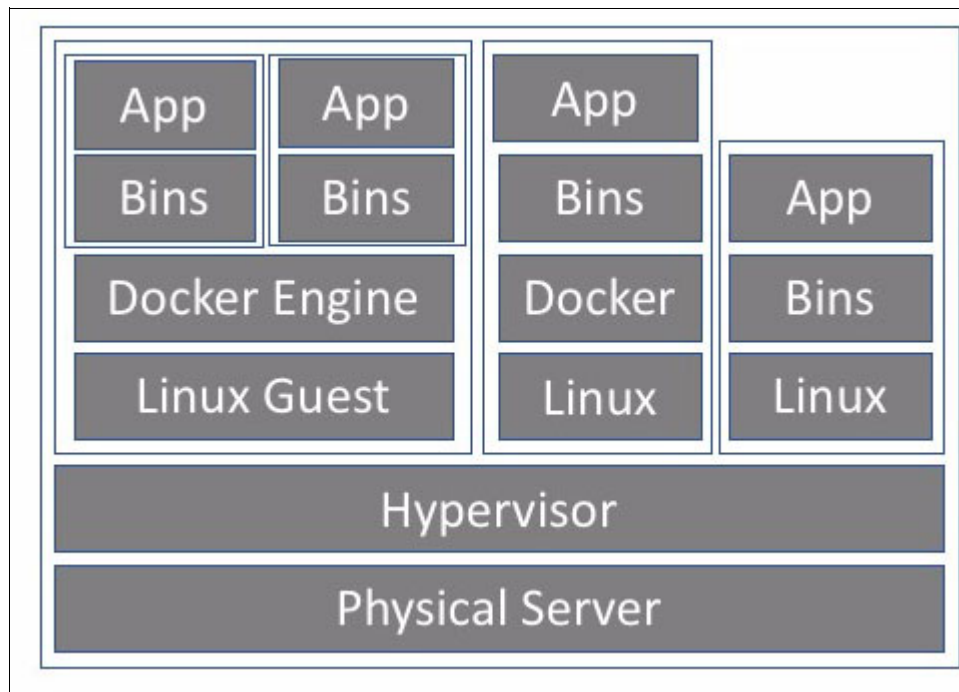


Figure 1-1 Docker Containers on Linux Guests on IBM Z

LinuxONE and IBM Z, together with Docker, provide the following benefits:

- Consolidation

IBM Z servers feature a rich history of running multiple, diverse workloads, all on the same platform, with built-in security features that prevent applications from reading or writing memory that is assigned to other applications that are running on the same box. IBM Z and LinuxONE also can share all resources on the server.

A great degree of scalability is available in the platform, with the ability to run many virtual machines on a single server. With Docker, multiple containers can be installed into a single virtual machine, which enables not only the ability to scale up even further, but allows for fewer virtual machines to be created because containers provide application, memory, and data isolation.

- ▶ Application portability

Combining the application and all associated dependencies allows applications to be developed and compiled to run on multiple architectures. The decision on which platform to deploy an application now becomes based on factors that are inherent to the platform (and by extension, the value a particular platform provides to an organization) rather than which platform a developer chose. Application portability provides a great deal of flexibility when it comes to deploying applications into production, and for choosing which platform for application development.

- ▶ Security

The inherent security features of IBM Z and LinxONE servers in terms of data protection. However, getting even more secure for certain key applications is the Secure Service Container (SSC), a special logical partition that does not have any outside access through outside interfaces, such as SSH. The only access to an application that is running in SSC mode is through remote APIs in the application.

- ▶ Colocation of data

A unique feature of running applications on Linux on IBM Z servers is the ability to access data that is in traditional mainframe systems, such as IBM Db2® for z/OS. Organizations that are undergoing a digital transformation can take advantage of the ease of application development while maintaining their existing database structures.

In addition, multitier applications can take advantage of the virtual networking capabilities of the IBM Z platform because of the scale-up capabilities of IBM Z. Network traffic between guests and LPARs on the same physical Z system does not flow outside of the server, which results in a decrease in network latency and networking cabling infrastructures.

1.2 Understanding the concepts of Docker EE on IBM Z

Docker is lightweight technology that runs multiple containers (group of processes that run in isolation) simultaneously on a single host for improving developer's workflow and speeds up application deployment.

It became rapidly common across organizations around the world and receives attention from large organizations today. It is important to highlight that system administrators and development teams are always looking for solution alternatives that can provide an agile development environment, speed to build, portability, and consistency to quickly deploy applications where increasing demands for complex applications are required.

With Docker technology, organizations can reduce costs, and increase efficiencies profits if the capability of applications can be quickly deployed to improve service. With the new business model (mobile, social, big data, and cloud), a technology can dictate the performance, costs, and response time for critical applications. This ability makes Docker a good technology to deploy new application in containers, a process that is named *containerization*.

With Docker's container platform, an application can also be moved between platforms and run without needing a code update. Also, it allows run the same application, regardless of the environment. For this reason, containers are significantly gaining space and improving the way applications are developed and deployed across the IT infrastructure. This technology is making the IT teams rethinking their processes to adopt containers to bring faster application releases and improve development efficiency.

Containers are not complex or heavy as Virtual Machines (VMs). Docker can quickly deliver the service stack (code, run time, system tools, system libraries, and settings) of the application easily. It also can encapsulate an application in a container with its own operating environment to allow a rapid and consistent application deployment.

Like a hypervisor, Docker allows sharing of resources, such as network, CPU, and disks to run multiple encapsulated workloads. Also, it provides ways to control how many resources can be used by a container to ensure an application runs with all necessary resources and prevents an application from using all resources from the host (see Figure 1-2).

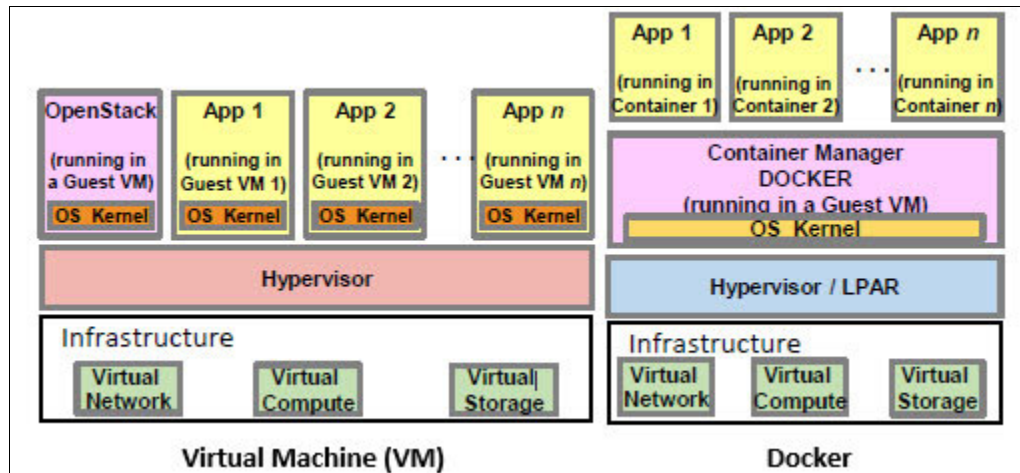


Figure 1-2 Virtual Machine versus Docker

VMs gained early popularity because they enabled higher levels of server utilization and provided better hardware usage. However, this technology is not ideal for every use case. If you must run the same applications in different types of virtual environments, the best choice is containerization and Docker provides this capability. The cost of making the wrong choice is significant and can delay critical deployments.

Additionally, VMs and containers take advantage of hardware and software utilization to make it possible to deploy several applications on the same server. The difference is that VMs provide abstraction from the underlying hardware. Containers provide abstraction at the application layer (or operating system) that packages code and dependencies together.

VM typically needs more resources because it must run a full copy of an operating system for every virtual server. It requires more memory and more CPU to run same application in VM as compared to container. However, containers include only those resources that are required by the application. In fact, containers are faster because a container can be quickly started, whereas a typical Linux process takes to start (few seconds), and containers were proven to be powerful to different types of workloads. Therefore, the decision of selecting the best technology is important and depends on the workload needs.

It is common to deploy each application on its own server (or virtual server), but it might result in an inefficient system without financial benefits. The operational costs and the amount of effort to control this large environment tend to be more expensive and less efficient. Therefore, organizations often face issues to keep the operation from running at top performance because of the sprawl challenges.

The main problems of server sprawl include the following hidden costs:

- ▶ Low resource usage
- ▶ Increased security problems

- ▶ Wasted energy consumption
- ▶ Increased server management workload

The uncontrolled proliferation of virtual machines without adequate IT control can lead to enormous challenges in designing an efficient and cost-effective infrastructure. It becomes more expensive and requires more efforts from support staff to track which servers should be updated, and how to handle security standards. It continues to be a strong focus area in the overall strategic plan of most organizations across the world.

Being intelligent and planning more for the criteria that is required to deploy an application can help avoid many of these problems.

It is easier to spin up a VM to test a new application release, upgrade, or deploy a new operating system. However, it can produce VM sprawl. System administrators and developers are adopting Docker to effectively combat this problem with a better use of server resources.

In addition, Docker can enable organizations to meet growing business needs, without increasing the number of servers or virtual servers. The result is fewer efforts to main the environment.

Docker provides tooling to manage containers and other benefits, which make the infrastructure more flexible and efficient for deployments. Some of those benefits are described next.

No need to run a full copy of an operating system

Unlike VMs, containers run directly on the kernel of hosting operating system, which makes them lightweight and fast. Each container is a regular process that is running on the hosting Linux kernel, with its own configuration and isolation. This design brings more efficiency than VMs in system resources terms and creates an excellent infrastructure that maintains all configurations and application dependencies internally.

Versioning of images

Docker containers provide an efficient mechanism to manage images. Having the necessary controls implemented helps you to easily roll back to a previous version of your Docker image that allows developers to efficiently analyze and solve problems within the application. It provides the necessary flexibility for development teams and allows them to quickly and easily test different scenarios.

Agility to deploy new applications

Developers face challenges when they must deploy new functions into a complex applications today. The environment became more critical, bigger (several components), and has many online users that are connected to the system.

Because of these reasons, developers are transitioning from monolithic applications to distributed microservices to allow organizations to move quickly, meet unexpected surges in mobile and social requests, and scale their business consistently.

The rapidly changing business and technology landscape of today shows the reasons to migrate these types of applications from the older style of monolithic single applications to a new model, microservices.

The use of Docker for managing and deploying this new architecture is an excellent approach to support developers to achieve the agility to deploy applications today. Also, it allows you to quickly make replications and achieve redundancy in case your application requires it. Therefore, starting applications with Docker is as fast as running a machine process.

Isolation

Docker containers use two Linux kernel features (cgroups and Linux namespaces) to enforce and achieve the required isolation and segregation. You can have various containers for separate applications that are running different services while one container does not affect each other on the same Docker host.

Linux namespaces provide resources to allow containers to create its own isolated environment that has its own processes namespace, user namespace, and network namespace. Therefore, each Docker container creates its own workspace as a VM does.

By using cgroups, you can manage system resources, such as CPU time, system memory, and network bandwidth to prevent a container from using all resources. Aside from this feature, Docker effectively permits you to control and monitor the system and each container separately.

Isolation that uses cgroups and Linux namespaces is shown in Figure 1-3.

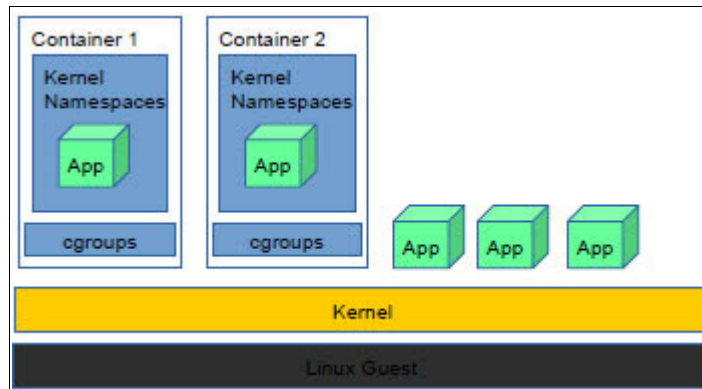


Figure 1-3 Linux Namespaces and cgroups

Better resource utilization

Virtualization technology is the ability for a computer system to share resources so that one physical server can act as many virtual servers. It allows you to make better use of hardware. In the same way, Dockers can share resources and use less resources than a VM because it does not require to load a full copy of an operating system.

Application portability

An application and all its dependencies and configurations can be packaged together into a single isolated container. Therefore, you can move this package across systems and run it without compatibility issues.

1.3 Docker overview

In this section, we introduce several basic Docker components, many of each we reference again in this publication. If you are going to manage Docker in your organization, you must understand these components. A graphical overview of the components is shown in Figure 1-4.

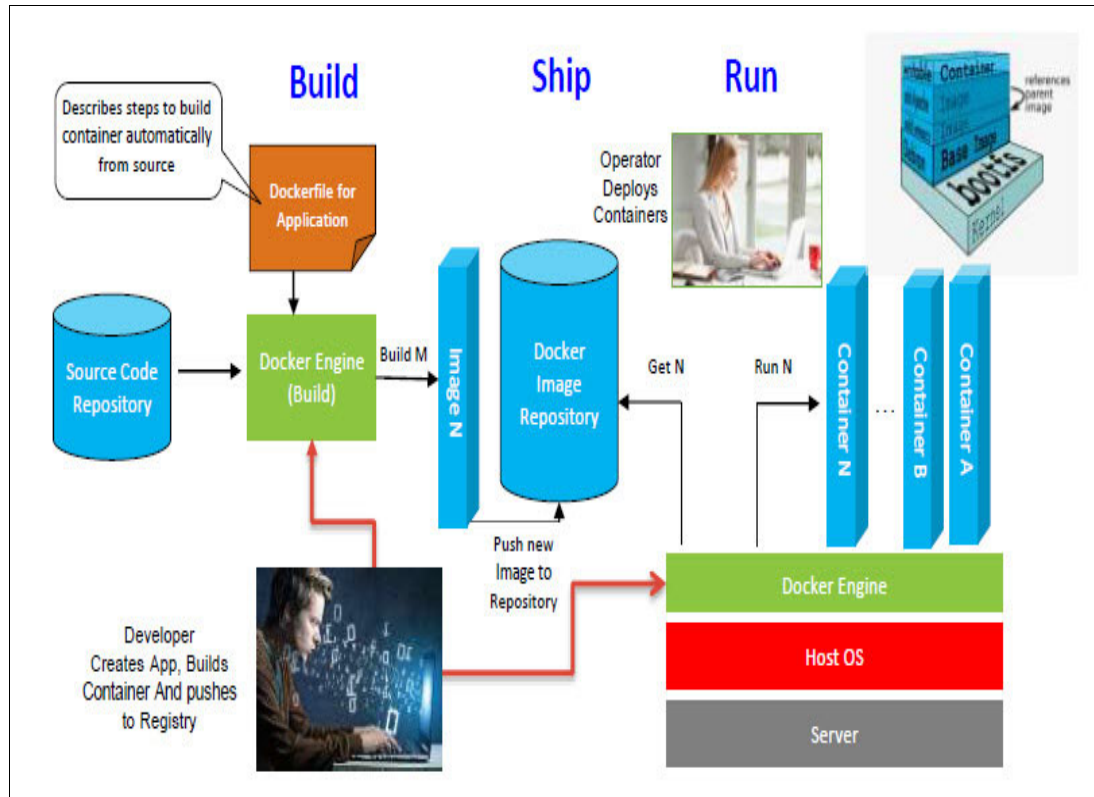


Figure 1-4 Overview of Docker components

Docker image

A Docker image is a read-only snapshot of a container that is stored in Docker Hub or any other repository. As VM golden image, the images are used as a template for building containers. It includes a tarball of layers (each layer is a tarball). To reduce disk space, a Docker image can have intermediate layers, which allows cache.

For more information about Docker images, see the following resources:

- ▶ [IBM Z](#)
- ▶ [IBM Managed Images \(including IBM Z and other platforms\)](#)

Dockerfile

A Dockerfile is a recipe file that contains instructions (or commands) and arguments that are used to define Docker images and containers. In essence, it is a file with a list of commands that are in sequence to build customized images. Figure 1-5 on page 11 and Figure 1-6 on page 11 show more information about a Dockerfile.

Consider the following points regarding Dockerfile:

- ▶ Includes instructions to build a container image
- ▶ Used to instantiate or transfer an Image container

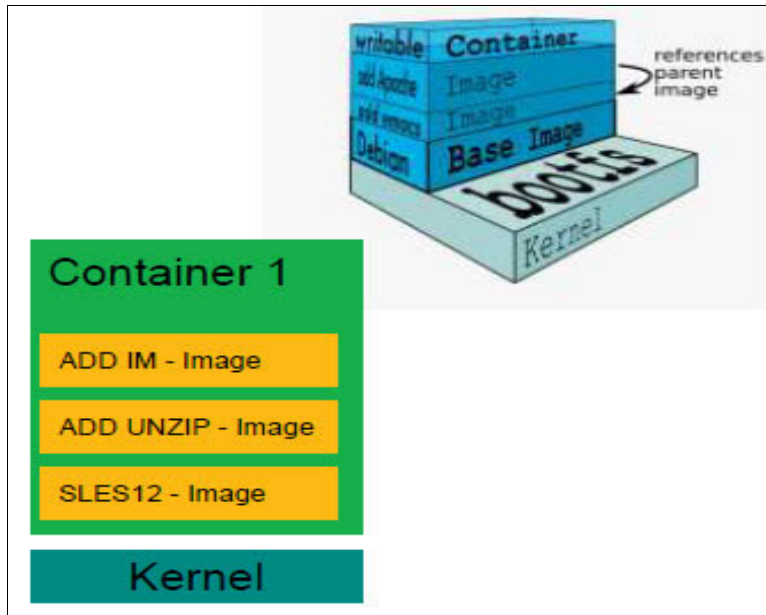


Figure 1-5 Dockerfile Architecture

```
# Dockerfile to build a container with IBM IM
FROM sles12.1:wlp
MAINTAINER Wilhelm Mild "mildw@de.ibm.com"
# Set the Install Repositories – replace defaults
RUN rm /etc/zypp/repos.d/*
COPY server.repo /etc/zypp/repos.d
# install the unzip package
RUN zypper ref && zypper --non-interactive install unzip
# copy package from local into docker image
COPY IM.installer.linux.s390x_1.8.zip tmp/IM.zip
# expose Ports for Liberty and DB2
EXPOSE 9080
EXPOSE 50000
# unzip IM, install it and delete all temp files
RUN cd tmp && unzip IM.zip && ./installc -log
/opt/IM/im_install.log -acceptLicense && rm -rf *
```

Figure 1-6 Sample Dockerfile

Docker container

A Docker container is a running instance of a Docker image and a standard unit in which the application and its dependencies are stored. A container is built by way of a Dockerfile that contains instructions to include necessary components into the new container image. This object often is loaded when you run the **docker run** command to start a new container.

Two docker containers that are running on a Docker Host are shown in Figure 1-7.

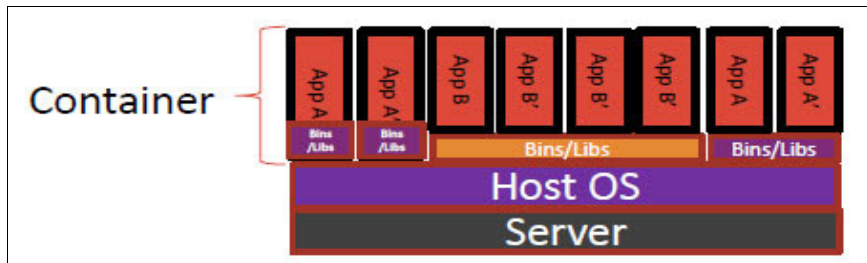


Figure 1-7 Docker containers

Docker registry

A Docker registry is a local repository to store, distribute, and share Docker images. Users can store and retrieve images from these places by way of registry API by using the **docker push** and **docker pull** commands, respectively. An example of Docker Registry (Hub) is shown in Figure 1-8.



Figure 1-8 Docker registry

Although you can create your own private registry to host your Docker images, some public and private registries are available on the cloud where Docker users and partners can share images.

Public registries

Docker organization offers two public registers: Docker Hub and Docker Store, as listed in Table 1-1.

Table 1-1 Docker public registries

Public registries	Description
Docker Hub	Offers for open community a place to store Docker Images. It is popular on the internet.
Docker Store	Offers official Docker Images (Certified content).

For more information about Docker Hub and Docker Store, [see this website](#).

Private registries

If your organization needs an option to use Docker images privately and allow only your users to pull and push images, you can set up your own private registry. Available options for setting up a private registry are listed in Table 1-2.

Table 1-2 Private Docker registries

Private registries	Description
Docker Trusted Registry (DTR)	Enables organizations to use a private internal docker registry. It is a commercial product that is recommended for organizations that want to increase security when storing and sharing their Docker Images.
Open Source Registry	Available for IBM Z and it can be deployed as a Docker container on IBM Z.

Consider the following points regarding Docker registries:

- ▶ Enable sharing and collaboration of Docker images
- ▶ Docker registries are available for private and public repositories
- ▶ Certified base images are available and signed by independent software vendors (ISVs)

Note: For more information about registries, [see this website](#).

Docker Engine

Docker Engine is the program that is used to create (build), ship, and run Docker containers. It is a client/server architecture with a daemon that can be deployed on a physical or virtual host. The client part communicates with the Docker Engine to run the commands that are necessary to manage Docker containers. Docker Engine functions are shown in Figure 1-9.

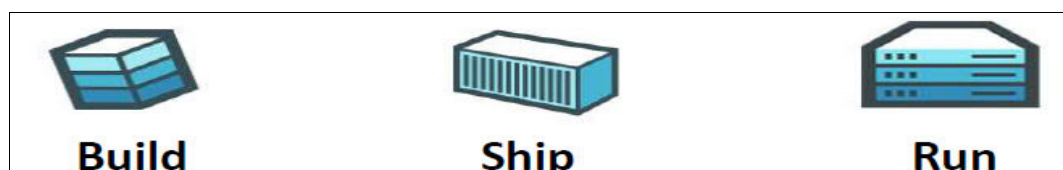


Figure 1-9 Docker Engine functions

Consider the following points regarding Docker Engines:

- ▶ Can run on any physical Logical Partition (LPAR) or virtual Linux server locally
- ▶ Uses Linux kernel cgroups and namespaces for resource management and isolation
- ▶ Uses copy-on-write file system for docker containers (for more information, see 1.4, “Security”)

1.4 Security

Organizations are finding that they must implement encryption in their applications. According to the international digital security company Gemalto, 1,765 data breach incidents were reported in 2017 for an exposure of over 2.6 billion data records¹. No industry is immune from data breaches. Data records include customer data, health data, financial records, and corporate secrets.

¹ For more information, see <https://breachlevelindex.com/>.

Unfortunately, these breaches are occurring because of lax security protocols; however, even with the most ardent practices in place, breaches can still occur. Compliance with government and other regulations is also mandating that safeguards be put in place, including data encryption.

When data that is exposed is not encrypted, the information can be used in various ways. However, with data that is encrypted, compromised data becomes worthless without the keys to decrypt. Therefore, it is increasingly necessary to encrypt data, not only at rest sitting on external storage, but data in flight.

Storage technology vendors offered encryption at the hardware level to apply encryption to any data that is coming on to the storage in files and databases, and then the data must be decrypted by the storage device as the data flowed from the device to the processor memory. Although this process safeguards data if a breach of disk storage or tapes being transported from one location to another occurs, it does not solve the problem with data moving to and from the processor memory, or moving on a network connection within the environment.

1.4.1 Pervasive encryption

IBM introduced pervasive encryption to meet the demand by organizations to encrypt data in-flight and at-rest. Pervasive encryption is policy-driven encryption that allows the organization to encrypt data as it flows through every part of the system. This flow includes data that is sitting on disk or tape at rest, data that is in transit from disk to processor memory, and data that is moving across a network to and from a z System server.

Not all organizations use host-based network security because of the CPU costs that are associated with encrypting data. However, no extra cost is incurred when network traffic is encrypted with the CPACF on the z14 server.

As shown in Figure 1-10 on page 15, the following levels of encryption are available:

- ▶ Full Disk and Tape Encryption

At the base of pervasive encryption is disk and tape hardware encryption. The IBM Z system provides for hardware-level encryption of data at rest by encrypting and decrypting data that is entering and exiting the storage device. As physical storage disks and tapes are moved (for example, for repair or replacement, or to another location), all data on that device remains encrypted. If the device is compromised, the data is meaningless without the encryption keys.

- ▶ File or Dataset Level Encryption

This level is the encryption of data that is moving to the storage device. This level (and the Database Encryption level) is where the encryption capabilities of the z14 server excel.

Data is encrypted by the server by way of the CPACF functionality. An audit and compliance point is to prove that data that is moving between physical devices is secure and by encrypting data on the server this audit point is sufficed.

The data is encrypted as it is moved to storage, and only decrypted when it moves back to the server and into memory for application use. This area is where applications traditionally needed to be modified to provide the algorithms for encryption and decryption. The z14 server provides this function in hardware, with no extra overhead.

- ▶ Database Encryption

The next level is specifically encrypting data that is stored in databases, such as IBM Db2. IBM operating systems, such as z/OS and z/TPF, work closely with the CPACF hardware to encrypt mission-critical databases.

► Application Encryption

At the top of the pyramid is application-based encryption, which is generally the most expensive method in terms of time and processing to perform encryption and decryption operations. This encryption is used when a granular, specialized encryption technique is needed and is not available in the hardware functionality (a rare occurrence with z14).

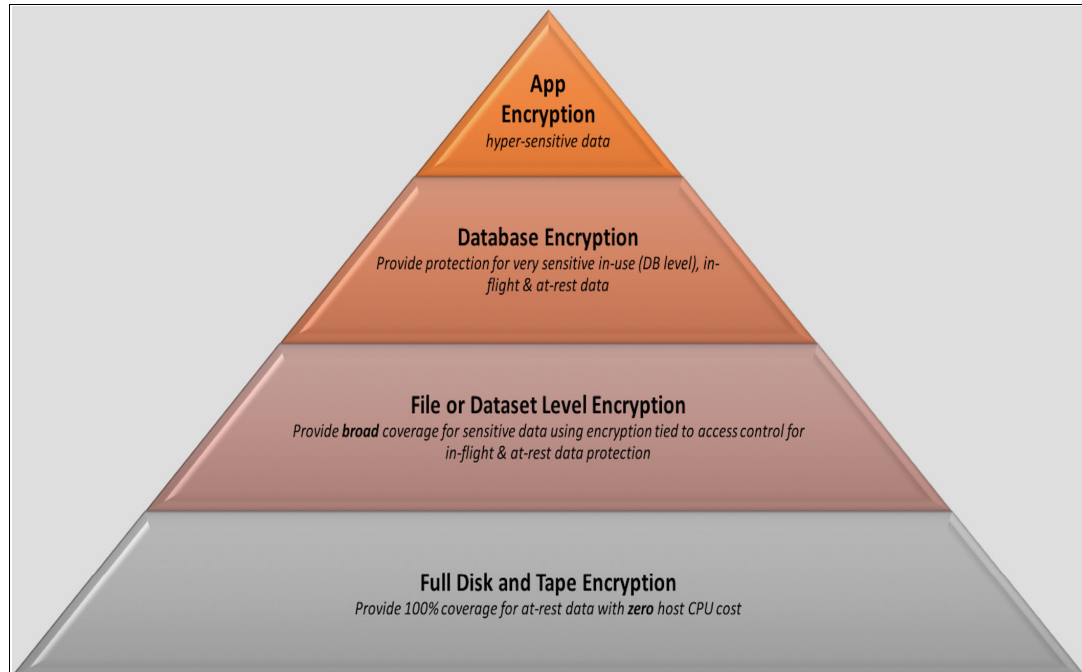


Figure 1-10 Encryption at all levels

Adding encryption traditionally meant that organizations needed open their source code to add the encryption algorithms. Application developers are already operating on tight deadlines to deliver new applications, enhancements, and bug fixes to applications. Changing programs involves function and unit tests, performance tests, QA testing, and sign-off by business units before the application can be moved into production. All of these processes are expensive in terms of money and time.

Pervasive encryption is a paradigm shift from selective encryption (where only data that is required to meet regulatory compliance is encrypted) to all data being encrypted all of the time. Security features are built into the z14 hardware through CPACF, which supports pervasive encryption and provides hardware acceleration for encryption operations.

By having an encryption engine that is built into the central processing core, encryption can now be done externally to applications directly by the operating system. The z14, Rockhopper II, and Emperor II along with the CPACF provide encryption for data without making any changes to the applications. Therefore, applications can run as-is with no changes or extra CPU overhead.

A top concern for organizations is the protection of encryption keys. In large organizations, hackers often target encryption keys, which are routinely exposed in memory as they are used. Only IBM Z and LinuxONE can protect millions of keys (and the process of accessing, generating, and recycling them) in “tamper responding” hardware that causes keys to be invalidated at any sign of intrusion. The keys can then be restored safely.

The IBM Z key management system is designed to meet Federal Information Processing Standards (FIPS) Level 4 standards, whereas the norm for high security in the industry is Level 2. This IBM Z capability can be extended beyond the mainframe to other devices, such as storage systems and servers in the cloud. In addition, IBM Secure Service Container protects against insider threats from contractors and privileged users and provides automatic encryption of data and code in-flight and at-rest, and tamper-resistance during installation and run time.

In summary, the design of the IBM Z enables pervasive encryption by providing the following benefits:

- ▶ Moving cryptographic workload away from central processors
- ▶ Heightening your security level by protecting and securing keys
- ▶ Accelerating encryption and decryption

1.4.2 Linux encryption

Linux running on the z14 server can use the cryptographic features of the hardware. Linux and z14 work closely together to provide encryption in the following areas:

- ▶ **dm-crypt:** Provides transparent file and volume encryption by using industry unique CPACF protected keys.
- ▶ **Network security:** Enterprise scale encryption and handshakes by using z14, Rockerhopper II, Emperor II CPACF, and Single Instruction, Multiple Data (SIMD).
- ▶ **Secure Service Containers:** Automatic protection of data and code for virtual appliances.

The IBM Z and LinuxONE systems provide cryptographic functions that, from an application program perspective, can be grouped into the following categories (see Figure 1-11 on page 17):

- ▶ **Synchronous cryptographic functions, which are provided by the CP Assist for Cryptographic Function (CPACF) or the Crypto Express features when defined as an accelerator:**
 - Support for symmetric and hashing algorithms
 - Included in every central processor (CP) and Integrated Facility for Linux (IFL) processor
 - Pseudo-random number generator
 - TRNG (z14)
- ▶ **Asynchronous cryptographic functions, which are provided by the Crypto Express features:**
 - Asymmetric and hashing algorithm offload
 - Host master-key storage
 - Hardware RNG
 - PKCS #11 Cryptographic Support

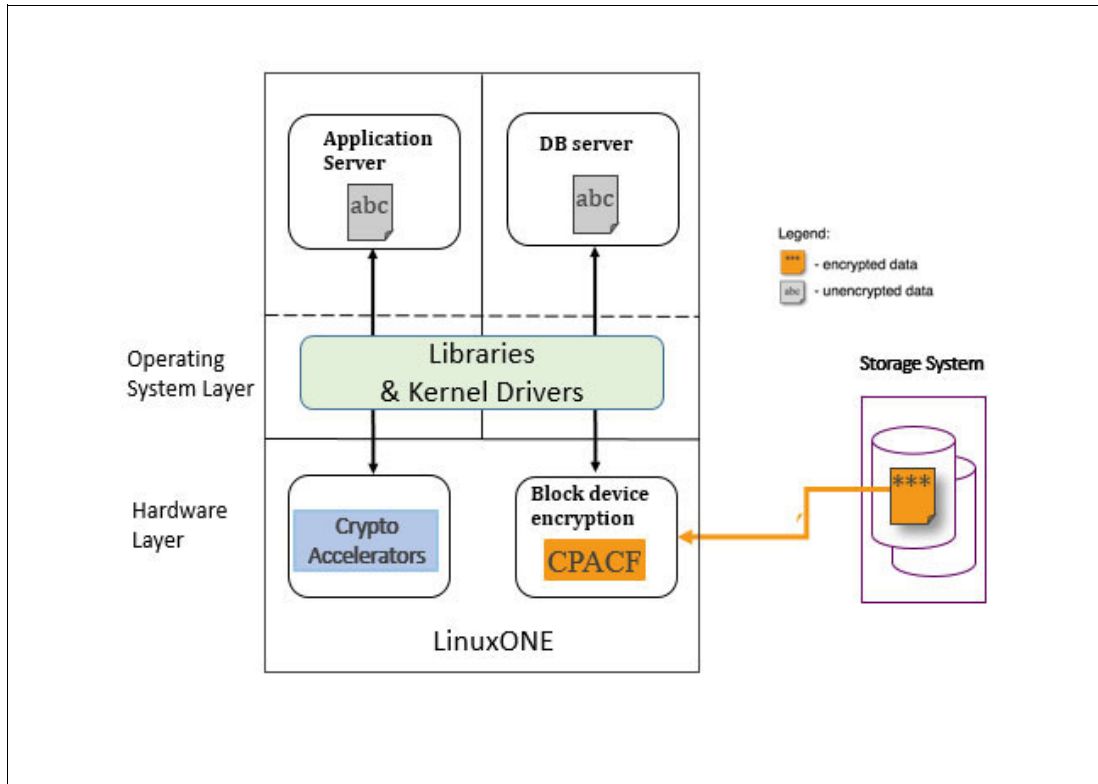


Figure 1-11 IBM Z Cryptographic functions

For more information about how to configure your Docker environment to take advantage of IBM Z crypto features, see 3.4, “Using hardware crypto for application containers” on page 64.

1.4.3 Secure Service Containers

Organizations are using container technology to build and deploy applications quickly as they modernize their environments. Organizations are challenged with the need to provide a higher level of security for mission-critical and highly sensitive data and applications.

The IBM Z and LinuxONE servers introduced a new, specialized type of LPAR that provides for the highly secure, fast deployment, and management of approved, pre-packaged applications, such as blockchain in an appliance model of deployment.

This Secure Service Container, which was started and loaded by using an appliance, does not include direct operating system access by way of SSH or otherwise. Only remote APIs (RESTful) are permitted access to the appliance, which restricts administrator access to workload and especially data. This configuration ensures that only those users with appropriate access credentials can access the application data and code that is deployed in the Secure Service Container.

In addition, memory access is disabled, and disk access is encrypted by default and cannot be disabled. Debugging data in the form of system dumps is also encrypted, which means that support groups can view only the application instructions in the dump, not any data that is included in the dump.

The architecture also provides strong isolation between Secure Service Container instances. This isolation is based on the EAL5+ level of protection and does not require dedicated hardware. An application that is running in one Secure Service Container cannot access an application that is running in another Secure Service Container other than through remote APIs.

IBM announced IBM Services Container for IBM Cloud™ Private. Built on Docker, it allows organizations to directly deploy their own containers into a Secure Services Container. IBM Cloud Private provides a platform for organizations for a cloud-native, cloud-ready management platform. This enhancement extends that capability to include the ability to harden their mission-critical, highly sensitive, Docker-, and Kubernetes-based workloads with data protection capabilities on one of the industry's most trusted, resilient computing platforms: IBM Z and IBM LinuxONE™.



Planning for Docker Enterprise Edition

In the face of the modern IT infrastructure that was brought by the container technology in the IT environment with various software operating environments, it is good practice to create a strategic plan before proceeding with Docker implementation. The Docker platform brings a modern concept of virtualization that is based in microservices that increases the number of layers and consequently the complexity of the application architecture.

Complex internal and external services, servers, and applications must interoperate to provide end-to-end service to customers and internal users alike. Unless it is not imperative, it is recommended to involve different IT departments in the planning process, such as infrastructure and development teams. The IT systems must communicate and integrate in a quick, flexible way with other IT systems within the company.

This chapter provides important information that assists you to build a plan, including basic scenarios, prerequisites for Docker, and links to other reference sources.

The scenarios that are presented in this chapter provide an overview, compare the different schemas, and list the considerations for each scenario.

This chapter includes the following topics:

- ▶ 2.1, “Sample of Scenarios for containers on Linux on z” on page 20
- ▶ 2.2, “Hardware and software prerequisites on IBM Z” on page 23
- ▶ 2.3, “Lab resources and configuration” on page 25
- ▶ 2.4, “z/VM and SSI considerations” on page 25
- ▶ 2.5, “Docker editions” on page 27
- ▶ 2.6, “Container orchestration” on page 27

2.1 Sample of Scenarios for containers on Linux on z

For the Docker architectural standpoint, the Linux on IBM Z administrator has several options for the containers. In this section, we present the logical infrastructure perspective in a common IBM Z configuration, including the following key infrastructure elements:

- ▶ Logical partitions (LPARs) (including hypervisor)

The LPAR is the logical partition that provides the logical segmentation and manages the hardware resources, such as processor and memory for the Linux guests. The hypervisor is installed in the LPAR and provides the operating system and hardware functionality.
- ▶ HOST (Linux)

The HOST is Linux virtual machines that can run in the z/VM or KVM hypervisors, or directly in the LPAR without a hypervisor, which is referred to as *bare metal*.
- ▶ Docker containers
The CONTAINERS are Dockers containers that virtualize the application that is running different workloads.

Next, we provide three hypothetical scenarios, including the characteristics and advantages of each of the architectural models.

2.1.1 Scenario 1: All elements are running in the same subsystem

From an administration perspective, running a unique system can save hours of system administrator work because it requires less workload to perform system maintenance or the effort for problem determination.

The IBM Z hardware provides the vertical scaling because the resources can be dynamically allocated in the LPAR. This feature brings a great benefit to host containers with unexpected workloads performance spikes, such as starting websites or containers with SQL databases.

This scenario includes the following benefits:

- ▶ Improves the vertical scalability on a scale up system, such as IBM Z
- ▶ Facilitates the administration tasks and decreases workload to create the plan
- ▶ Decreases the number of resources to manage (only one OS and hypervisor)

Despite being easier administer, the container system features a single point of failure that is affected by software or hardware maintenance.

Note: In the case of IBM Z, the hardware redundancy and dynamic resource allocation can minimize the effect of a single point of failure. This feature provides high availability, but is not applied to software maintenance that requires system restarts.

2.1.2 Scenario 2: Same LPAR with different hosts running Docker

This scenario covers horizontal scalability with environment segmentation: development, test, production. The system administrator can organize the containers per type of workload or application that is running different workloads (web services, database, and application server for only one application).

The option to segment development, test, and production tasks is widely used in IT environments and involves the administrator network infrastructure. The communication between the hosts is not an issue for IBM Z because a feature that is called IBM HiperSockets™ provides network communication in the memory speed.

For more information about HiperSockets, see *IBM HiperSockets Implementation Guide*, SG24-6816.

In Figure 2-1, the four Linux hosts are running on top of a single hypervisor. During the hypervisor maintenance that requires IPL (or restart), all of the Linux servers and their containers must be recycled.

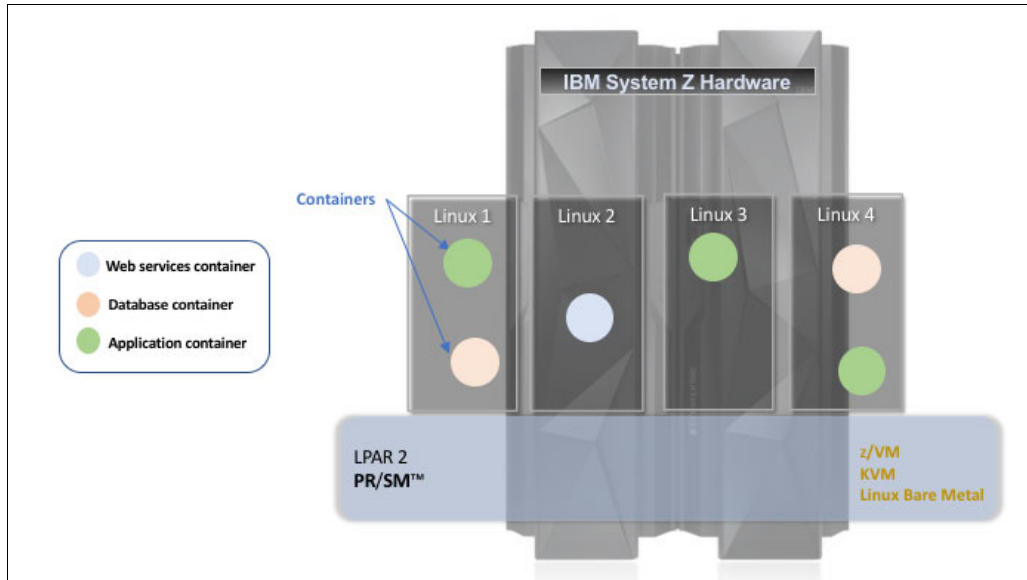


Figure 2-1 Linux hosts running on top of a single hypervisor

This scenario includes the following benefits:

- ▶ Provides vertical and horizontal scalability.
- ▶ Avoids outages during Linux maintenance; for example, the containers can be moved to another host during a kernel upgrade.
- ▶ Supports splitting the containers in different environments (development, testing, and production).
- ▶ Avoids a single point of failure.

2.1.3 Scenario 3: Dockers hosts are running on different LPARs and servers

Despite the workload during plan stage, this more complex scenario provides greater flexibility to system administrators during maintenance and improves the system availability.

In scenarios that involve more than one IBM Z or LinuxONE server, the availability can be obtained at the container level, which includes moving the container between different hosts, and the host level through features, such as, z/VM Live Guest Relocation (LGR) that allows the Linux hosts to be relocated in different LPARS.

For more information about z/VM Live Guest Relocation, see *An Introduction to z/VM Single System Image (SSI) and Live Guest Relocation (LGR)*, SG24-8006.

Note: On IBM Z and LinuxONE machines, the IBM Processor Resource/Systems Manager™ (IBM PR/SM) hardware feature supports the creation of multiple LPARs on a single central processor complex (CPC), which splits server resources across the LPARs. Each LPAR supports an independent operating system or hypervisor: z/VM, Linux, or kVM.

The Figure 2-2 shows multiple hosts and hypervisors instances (LPARs). This configuration works well for hypervisor maintenance to avoid outages because the Linux hosts and containers can be relocated to the other LPAR.

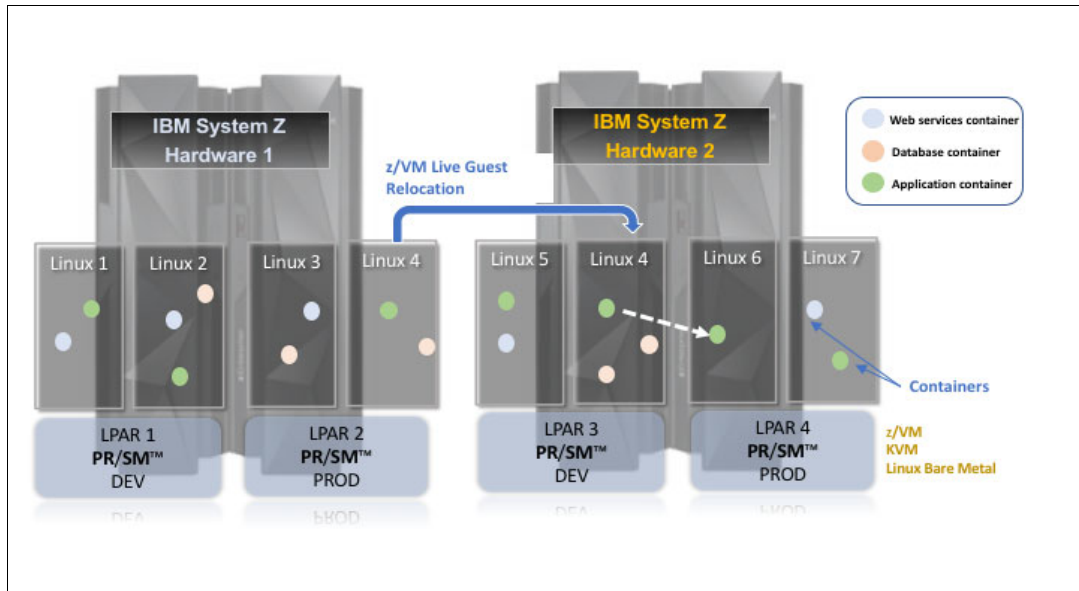


Figure 2-2 Multiple hosts and hypervisors

For more information, see 3.5, “Moving Docker hosts by using z/VM SSI feature” on page 78 (SSI).

This scenario includes the following benefits:

- ▶ Provides vertical and horizontal scalability within different hardware.
- ▶ Avoids outages during software, hardware, or power maintenance¹ tasks. For example, the containers can be moved to another host and the hosts can be moved to another machine during a kernel upgrade, which provides the high availability.
- ▶ Dynamically changes the environment design.
- ▶ Provides flexible designs options, including the disaster recovery situation.

¹ If you are using different machines, it also avoids hardware outages during power maintenance.

2.2 Hardware and software prerequisites on IBM Z

The IBM Z is a business-critical IT platform is fully virtualized in the IBM System z® or LinuxONE hardware. In this section, we discuss the prerequisites to run the containers in IBM Z, including information regarding hypervisor, hosts, and containers.

As full virtualized platform, the IBM Z hardware resources that are managed by the firmware that provides partitioning is known as *IBM Processor Resource/System Manager* (IBM PR/SM). It is the IBM PR/SM functions that are used to create and run LPARs. An LPAR is a subset of the processor hardware that is defined to support an operating system. An LPAR contains resources (processors, memory, and input/output devices) and operates as an independent system. Multiple logical partitions can exist within a mainframe hardware system.

The hypervisors that are supported in the IBM Z are Kernel Virtual Machine and IBM z/VM. These hypervisors provide for the guests operational systems that are the necessary hardware to host the entire environment.

For the next level of Docker container environment, such as production, the project must predict the requirements for the application that is running in the containers and the target audience of the application. In this section, we describe the prerequisites with the minimum requirements that are needed to run Docker containers on IBM Z.

We based this IBM Redbooks publication on a common IBM Z installation where the Linux operating system runs on top of the hypervisors. The Linux that is supported on IBM Z is SLES, RHEL, and Ubuntu Enterprise Edition for s390x architecture.

Because we assume that the hypervisor is installed and the host Linux is running in your environment, we focus on Docker requirements. Before proceeding with the Docker installation process, we recommended that you review the prerequisites and compatibility matrix that is available at [the Docker website](#).

2.2.1 Requirements for installing Docker on IBM Z

Ensure that all the hosts that you want to manage with Docker Enterprise Edition meet the following minimum requirements:

- ▶ Linux with kernel version 3.10 or higher.
- ▶ Docker Enterprise Edition 17.06.2-ee-8 (values of *n* in the -ee- suffix must be 8 or higher)
- ▶ 4 GB of RAM
- ▶ 3 GB of available disk space

The minimum requirements for Docker installation are highly dependent on the following components:

- ▶ Docker Engine (daemon):
 - 18.03: Use this version if you are running Docker Enterprise Edition Engine only.
 - 17.06: Use this version if you use Docker Enterprise Edition 2.0 (Docker Engine, UCP, and DTR).
- ▶ Docker Trusted Registry (DTR)
- ▶ Docker Universal Control Plane (UCP)

The hardware requirements for these components are listed in Table 2-1.

Table 2-1 Hardware requirements

Component	CPU/MEMORY/NETWORK	Disk space
Docker Engine	4 GB	3 GB
DTR	<p>For Dev/Test:</p> <ul style="list-style-type: none"> ▶ 8.00 GB of RAM for manager nodes or nodes running DTR ▶ 4.00 GB of RAM for worker nodes ▶ 2 vCPUs for nodes running <p>For Production:</p> <ul style="list-style-type: none"> ▶ 16 GB of RAM for nodes running DTR ▶ 4 vCPUs for nodes running DTR 	<p>For Dev/Test: 10 GB</p> <p>For Production: 25 - 100 GB of free disk space</p>
UCP	<p>8.00 GB of RAM for manager nodes or nodes running DTR</p> <p>4.00 GB of RAM for worker nodes</p> <p>Static IP address ^a</p>	<p>3 GB</p> <p>25 - 100 GB of free disk space</p>

a. The UCP installation requires an IP address. For more information, see the Docker documentation that is available at [this website](#).

Before you install Docker Enterprise Edition (EE) on SuSE Linux Enterprise Linux (SLES), the following prerequisites must be met:

- ▶ Install a 64-bit version of SLES12 SP3 on your IBM Z machine.
- ▶ The `/var/lib/docker/` is a BTRFS file system. BTRFS is the only storage driver that is supported by Docker EE on SLES.

Docker can be run with the information that is listed in Table 2-1. However, you cannot manage many containers and workloads. For productions installations, consider including more resources to accommodate your workloads and prevent problems with Docker cluster.

For more information about recent updates, see the [Deploy Enterprise Edition on Linux servers page](#) of the Docker website.

2.3 Lab resources and configuration

For the purposes of this book, we built the lab that is running the hypervisor IBM z/VM on top of four LPARs by using the hardware resources that are listed in Table 2-2.

Table 2-2 Hardware resources

Hypervisor	z/VM version	Dockers hosts/IP	Virtual resource	Hardware total
ITSOZVM 1	z/VM 7.1	RHEL: itsoreda/9.76.61.245 itsoredb/9.76.61.246 itsoredc/9.76.61.251	OS: SuSE 12sp3 vCPU: 2 Memory: 16 GB Disk: 20 GB	IBM System z14 Proc: 16(IFL) Mem: 3.008 GB Flash: 1024 GB Disc: 1 TB
ITSOZVM 2		SLES: itsoslea/9.76.61.239 itsosleb/9.76.61.240 itsoslec/9.76.61.241	OS: Redhat 7.5 vCPU: 2 Memory: 16 GB Disk: 20 GB	
ITSOZVM 3	z/VM 6.4	UBUNTU: itsoubua/9.76.61.237 itsoubub/9.76.61.238	OS: Ubuntu 18.04 vCPU: 2 Memory: 16 GB Disk: 20 GB	
ITSOZVM 4				

The Linux Virtual servers on IBM Z can communicate by themselves by using the HiperSockets feature (think of it as an “in-memory” Internet Protocol network). This technology uses memory as its transport media without the need to go out of the server into a real network, which simplifies the need to use cables, routers, or switches to communicate between the virtual servers.

The IBM Z provides excellent vertical (scale up) and horizontal (scale out) scalability. In the lab environment (see Table 2-2), the entire system includes free hardware resources that are available to scale up, which increases virtual resources, such as processor, memory, and disk without interruption.

Note: *Scale out* refers to provisioning new virtual servers with all requirements to run the operating system on top of the hypervisor. The virtual servers coexist in total isolation and share the IBM Z server resources.

2.4 z/VM and SSI considerations

For Linux on System z platform, the packages that are used are compiled to s390x architecture. In this book, we based our installation on Docker Enterprise Edition.

In the IBM Z and LinuxONE environments, it is recommended to use a hypervisor to provide more resilience to the environment. During the planning stage, we decided to use the z/VM to have one mature virtualization platform fit well with the IBM Z hardware.

Also in our lab, we enabled a z/VM SSI feature that is included in the z/VM package that provides the Live Guest Relocation. This feature allows the Linux hosts be moved between different LPARs without any outage. This feature improves availability because the Linux hosts and its containers can be moved before the hypervisor maintenance.

An overview of our lab is shown in Figure 2-3.

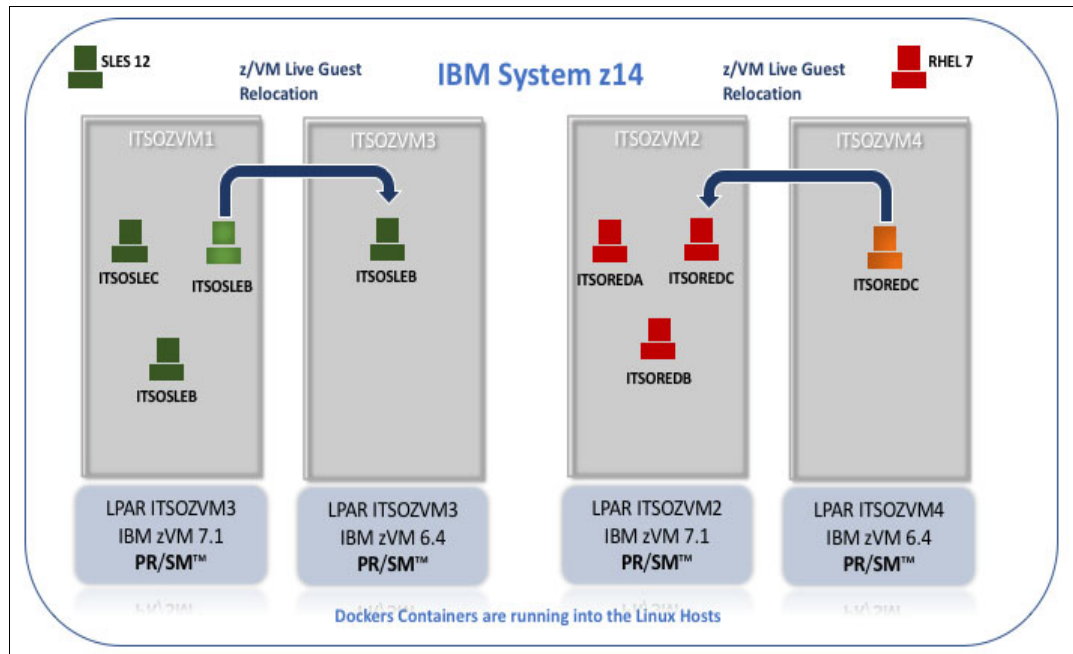


Figure 2-3 Lab overview

The z/VM SSI Cluster features two versions: 7.1 and 6.4. The versions are compatible, which allows moving different guests Linux between the LPARs. This feature is useful for avoiding an outage during z/VM maintenance (for example, upgrading from 6.4 to 7.1).

For more information about SSI, see *An Introduction to z/VM Single System Image (SSI) and Live Guest Relocation (LGR)*, SG24-8006.

2.4.1 High availability and Docker

z/VM SSI and Docker containers are an important part of a high availability (HA) strategy in an organization. Both allow for continued operation of an environment when a system must be updated for configuration changes or maintenance.

SSI is used for moving z/VM guests, especially Linux virtual servers from one z/VM host to another without stopping that server. The move is not apparent for processes and users that are using that Linux server as the move is completed.

For containers, the orchestration strategy is best for moving workloads in containers, and they might not persist across live guest relocations of Linux servers. If the workload was designed with HA in mind, one part of that workload can be taken down during periods of low activity with the work being split across multiple servers. The container orchestration can be used to move the hosting containers from the current Linux server to another server. Then, after that process is completed, the Linux server can be relocated by using Live Guest Relocation.

2.5 Docker editions

An important planning step is to decide which Docker edition you use in your project. In this section, we describe some of the differences between Docker's editions to understand the reason Linux on System z supports only Docker Enterprise Edition (EE).

Docker includes the Community Edition (CE), which is a free version that is available for anyone who wants to get started with Docker. It includes the full Docker platform and is an ideal choice for developers and new users. It also is a good alternative to start building their own container applications. Docker CE is not supported on IBM Z.

For business-critical deployments, such as IBM Z environments, use Docker EE, which provides an optimized Docker version and is supported directly by Docker. It includes certified operating systems and cloud providers and runs certified containers and plug-ins from the Docker store. Docker EE is available in the following tiers:

- ▶ **Basic:** Includes certified infrastructure, with support from Docker company. The containers and plug-ins are certified and available in the Docker store.
- ▶ **Standard:** Includes, beyond basic edition, advanced image, and container management, LDAP/AD user integration, and role-based access control (Docker Datacenter).
- ▶ **Advanced:** Adds Basic and Standard with advanced container management (Docker Datacenter) and Docker Security Scanning.

The IBM Z supports Docker EE on the most popular Linux enterprise distributions: Red Hat Enterprise Linux (RHEL), Ubuntu, and SUSE Linux Enterprise Server (SLES). For more information about installing Docker EE, see Chapter 3, "Installing and deploying Docker" on page 35.

For more information, see the [Docker compatibility matrix](#).

2.6 Container orchestration

On a full virtualized environment, such as IBM Z, we recommend the use of an orchestration tool that is available for Docker EE. Currently, two Docker orchestrators are supported on IBM Z platform: Kubernetes and Swarm. In this section, we discuss the concept of each of these orchestrators to help you plan your Docker environment to become more secure with high availability.

Container orchestration allows users to manage containers across the infrastructure. It defines the initial deployment of the container and the management of multiple containers, which ensures availability, scaling, networking, and portability of applications. Container orchestration ensures that all containers are managed and automated as needed.

It is used to control and perform the automation of several tasks, especially on large and dynamic environments with tasks, such as the following examples:

- ▶ Provision and deployment of containers
- ▶ Load balancing of services
- ▶ Establish network between containers
- ▶ Control and automation of application updates
- ▶ Allocation of resources between containers
- ▶ Monitoring of containers and hosts
- ▶ Scaling up and down by adding and removing containers as needed
- ▶ Keep storage consistent with multiple instances of an application

It is important to understand container management and its configuration, as shown in Figure 2-4.

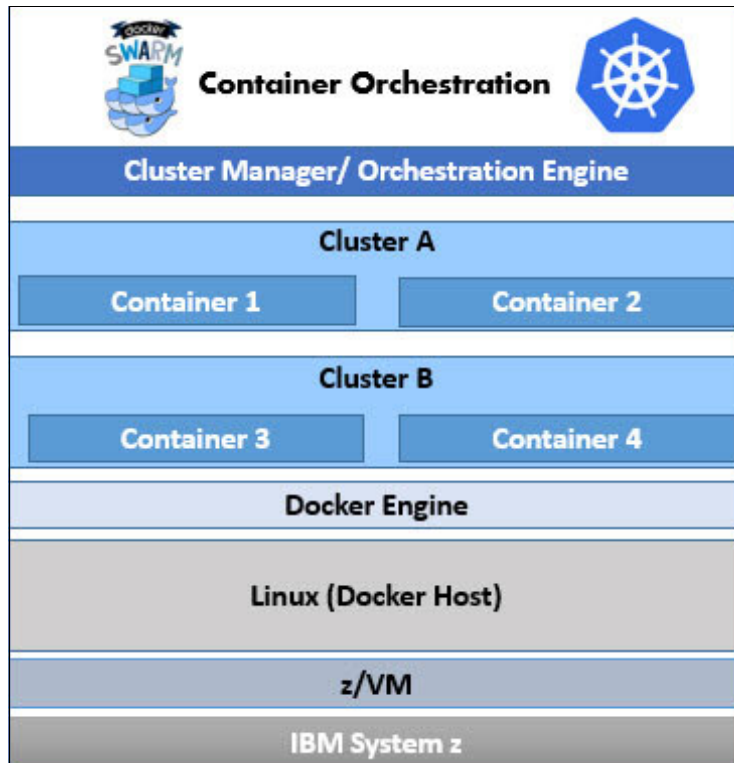


Figure 2-4 Container orchestration

2.6.1 Kubernetes

Also known as k8s, Kubernetes is the leader orchestrator and open source platform to automate the deployment, scaling, and management of containers. It supports a range of containers engines, including Docker. Kubernetes integrates with the Docker engine to coordinate the scheduling and running of Docker containers that use Kubernetes.

The developers community is large, with a powerful container discovery, and is ideal if you have a large amount of undifferentiated compute. It also has a complex installation process because it includes many features. It is recommend that an expert on Kubernetes helps to configure this tool.

Kubernetes components

The architecture of Kubernetes provides a flexible and loosely coupled mechanism for service discovery. A Kubernetes cluster consists of at least one master and multiple compute nodes. To provide high availability, it is necessary to deploy multiple master nodes, as it is recommended for production environments as well.

Kubernetes Master node is the host that contains the control panel components that make decisions about the cluster, such as scheduling pods to run on those nodes.

Note: *Pods* allow you to logically group containers and pieces of our application stacks and are discussed further in this section.

The Kubernetes Master node includes the following key components:

- ▶ API server

This server is the main management point of the cluster. This process validates and configures the data for pods, services, and replication controllers and assigns pods to the nodes. It also performs the synchronization of pod information with the server configuration.

- ▶ Etcd

This component is a high availability key-value that stores the state of all resources on Kubernetes cluster. It stores the state of services, deployments, and pods. The stored data is encrypted.

- ▶ Kube-controller manager

The controller manager server watches the etcd to collect and send information to the API server. Then, it makes changes to bring the status to the wanted state. The controller manager runs different rules, for example:

- Replication controller: Responsible for performing the correct number of pods
- Endpoints controller: Joins services and pods
- Server account controller: Creates default accounts and API access tokens
- Node controller: Responsible for noticing and respond when nodes are down

- ▶ Kube-scheduler

This component watches for newly created pods that are not yet assigned. It defines where to create them, taking into account the resource requirements, policy constraints, deadlines, performance needs, capacity, and workload requirements.

On a non-master node in a cluster, it is necessary to communicate with the Kubernetes Master by using one of the following processes:

- ▶ Kubelet: An agent that runs on each node in the cluster that ensures that containers are started and that they continue to run.
- ▶ Kubelet-proxy: Enables the Kubernetes services abstraction by maintaining network rules.

The Kubernetes architecture is shown in Figure 2-5.

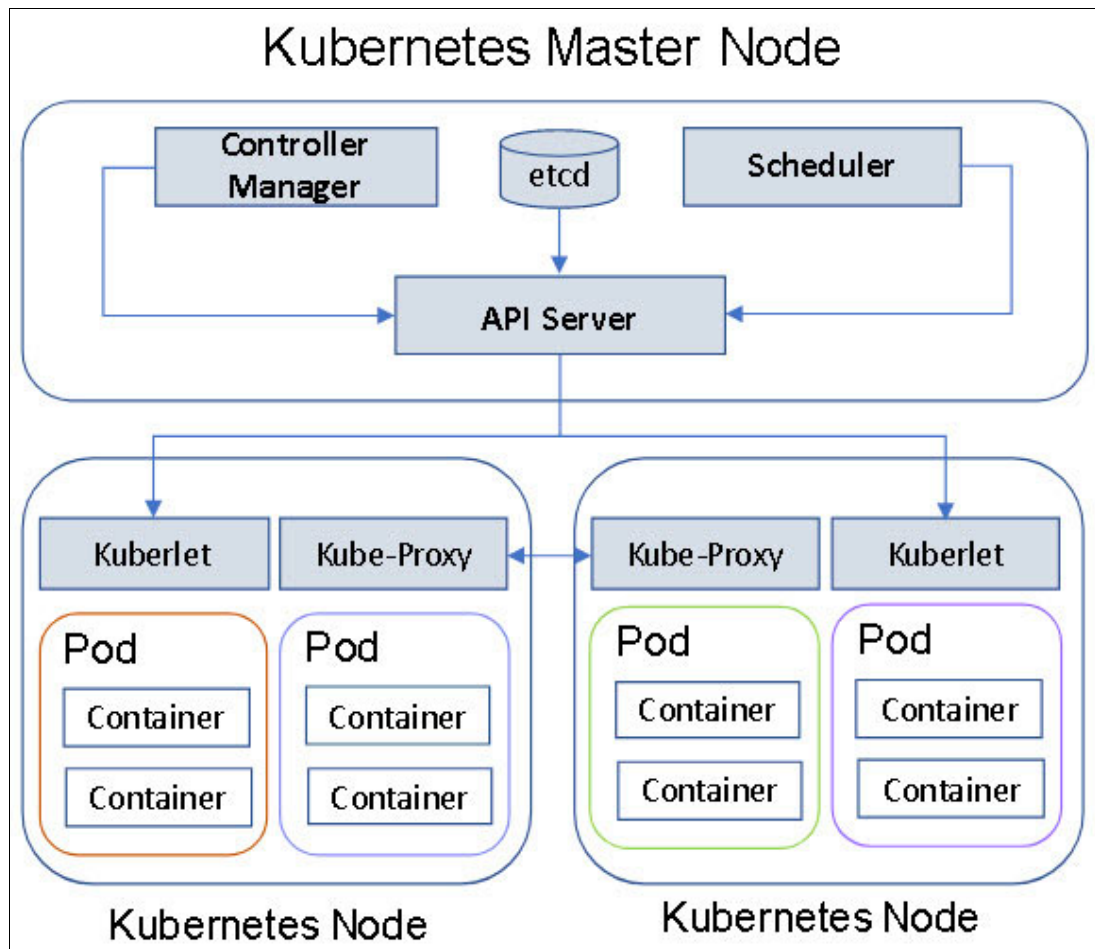


Figure 2-5 Kubernetes architecture

Kubernetes nodes include pods. A *pod* is a group of containers that shares storage and network interfaces. They also share the IP address and port space. A pod represents one or more containers that are controlled together. These containers that operate closely together share a lifecycle and always are on the same node.

As noted earlier in this section, pods allow you to logically group containers and pieces of our application stacks. Also, containers that are on the same pod feature the same host name.

Replication controllers

A replication controller (see Figure 2-6) is a logical entity that creates and manages pods and ensures the high availability of pods. The replication controller ensures that a pod or a set of pods are always up and available.

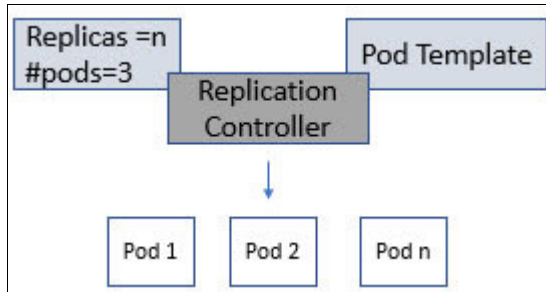


Figure 2-6 Kubernetes replication controller

Replication controllers maintain the wanted number of pods. If more containers than expected are available, it ends or deletes the extra containers. If a pod or host fails, new pods are started to compensate. It also keeps applications running with minor interruptions, gracefully recovers, performs rolling updates, and minimizes the effect on application availability.

Services

A Kubernetes Service defines a logical set of pods by a policy to access them. It works as a basic internal load balancer. A service groups logical collections of pods that perform the same function as a single entity. It is available by using an internally routable IP address.

For more information about the Kubernetes implementation, see Chapter 3, “Installing and deploying Docker” on page 35.

For more information about operational commands, see Chapter 4, “Basic Docker operations” on page 101.

2.6.2 Swarm

Docker provides the following description for swarm:

A swarm consists of multiple Docker hosts, which run in swarm mode and act as managers (to manage membership and delegation) and workers (which run swarm services). A specific Docker host can be a manager, a worker, or perform both roles.

When you create a service, you define its optimal state (number of replicas, network, and storage resources that are available to it, ports the service to the outside world, and more). Docker works to maintain that wanted state. For instance, if a worker node becomes unavailable, Docker schedules that node’s tasks on other nodes. A task is a running container that is part of a swarm service and managed by a swarm manager, as opposed to a stand-alone container.²

The Docker CLI is used to create a swarm, deploy application services to a swarm, and manage swarm behavior. No web interface is used for swarm.

² Source: <https://docs.docker.com/engine/swarm/key-concepts/#what-is-a-swarm>

The following key concepts are related to swarm:

► Node

A node is an Instance of the Docker engine that is participating in the swarm. Multiple nodes can be run on one physical or virtual server. Typically, nodes are spread across multiple servers in a production environment. Nodes in a swarm can be manager or worker nodes:

– Manager node

This node conducts the orchestration and management functions for the swarm. Interaction with the swarm is with a manager node, which in turn dispatches units of work that are called *tasks* to the worker nodes.

Docker recommends the use of an odd number of manager nodes in a swarm for redundancy to maintain a quorum, with seven nodes being the maximum recommended. It is possible to run with only one manager node; however, if this node fails, the swarm continues to operate, but a new cluster must be built to recover.

Note: Increasing the number of manager nodes does not increase performance of the swarm, and doing so might negatively affect the performance of the swarm.

– Worker node

This node receives tasks from the management nodes and runs. Worker nodes inform the manager nodes of their status and the status of the assigned tasks. They exist to run containers. They have no input into load-balancing decisions of the swarm and they do not participate in raft consensus. The status of worker nodes is controlled by the manager.

Note: Manager nodes are worker nodes by default. Manager nodes can be demoted to worker node status by using the `docker node demote` command. Worker nodes can be promoted to manager node status by using the `docker node promote` command.

► Tasks and services

A task carries a Docker container and the commands to run inside the container. Manager nodes assign tasks to worker nodes according to the number of replicas that are set in the service scale. After a task is assigned to a node, it cannot move to another node; it can run on the assigned node only or fail.

A service is the definition of the tasks to run on the manager or worker nodes. It is the central structure of the swarm system and the primary root of user interaction with the swarm. Service definition specifies which container image to use and which commands to run inside running containers.

The Swarm architecture is shown in Figure 2-7.

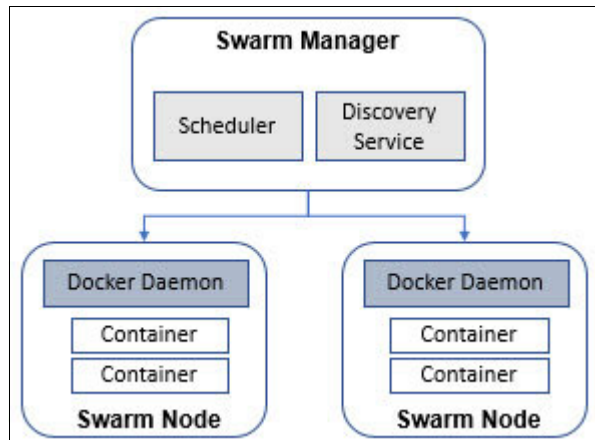


Figure 2-7 Swarm architecture

► Load balancing

The swarm manager uses ingress load balancing to make known the services that are made available externally to the swarm. The swarm manager can automatically assign the service a `PublishedPort`, or a `PublishedPort` can be configured for the service 30000 - 32767 range.

External components, such as cloud load balancers, can access the service on the `PublishedPort` of any node in the cluster whether the node is running the task for the service. All nodes in the swarm route ingress connections to a running task instance.

Swarm mode has an internal DNS component that automatically assigns each service in the swarm a DNS entry. The swarm manager uses internal load balancing to distribute requests among services within the cluster that are based on the DNS name of the service.

For more information about swarm implementation, see Chapter 3, “Installing and deploying Docker” on page 35.

For more information about operational commands, see Chapter 4, “Basic Docker operations” on page 101.

For more information about the difference between Kubernetes and swarm and the usability and best practice for each tool, see Chapter 4, “Basic Docker operations” on page 101.



Installing and deploying Docker

This book is based in Docker on-premises installation, which means that the Docker environment is run inside the infrastructure that is owned by the organization.

This chapter describes the Docker installation steps in two different operating systems (Linux Red Hat and SuSE), and follows the prerequisites that are described in Chapter 2, “Planning for Docker Enterprise Edition” on page 19.

This chapter includes the following topics:

- ▶ 3.1, “Installing Docker” on page 36
- ▶ 3.2, “Docker storage” on page 48
- ▶ 3.3, “Docker verification” on page 61
- ▶ 3.4, “Using hardware crypto for application containers” on page 64
- ▶ 3.5, “Moving Docker hosts by using z/VM SSI feature” on page 78
- ▶ 3.6, “Setting up swarm mode” on page 79
- ▶ 3.7, “Setting up Kubernetes” on page 82
- ▶ 3.8, “Kubernetes versus Docker Swarm” on page 98

3.1 Installing Docker

Running containers on IBM Z benefits from a hardware platform that includes specialized processors, cryptographic cards, availability, flexibility, and scalability. These systems are prepared to handle business-critical applications and provide a standard platform to run your Docker instances. These systems can support up to 2 million Docker containers on a single machine, which maintains the highest levels of security and performance and does not require application server farms. Yet, IBM Z remains largely to support non-production and production applications.

LinuxONE and IBM Z provide the following options to run Docker:

- ▶ Installing Linux directly in the LPAR (LPAR mode)
- ▶ Installing Linux on top of z/VM (as a guest)
- ▶ Installing Linux on top of KVM (as a guest)

Docker released the following editions of Docker to attend to the needs of developer communities and all types of organizations:

- ▶ Community Edition (CE): A no-charge edition that is available for developers and small teams that want to start with Docker. Also, it is supported by the community. Ubuntu is the only available Linux distribution on which to install Docker CE on IBM Z.
- ▶ Enterprise Edition (EE): A licensed edition that is available for developers and organizations that want to run critical business workloads in production. You can install Docker EE on IBM Z with Red Hat Enterprise Linux, SuSE Linux Enterprise Server, or Ubuntu.

The difference between the two editions is related to the number of features that are available for each edition. The EE edition has more features and more advantages than the CE edition. For more information about Docker editions, see [this website](#).

3.1.1 SuSE Linux

Docker EE is available for SuSE Linux and features the following installation options:

- ▶ Manual: The RPM file is downloaded from the Docker URL repository.
- ▶ Zypper: The necessary package and its dependencies are downloaded by using Zypper, which is the preferred method.

In this section, we describe how to install Docker Enterprise Edition (Docker EE) in a SuSE Linux Enterprise Server 12 SP3 by using the Docker official repository by way of Zypper.

Prerequisites

As of this writing, the following prerequisites must be met:

- ▶ SLES 12 SP3 64-bit version
- ▶ `/var/lib/docker` folder is created under a file system that is formatted with BTRFS
- ▶ Docker RPM package is downloaded

Enabling repository and installing

For Docker EE engine installation, it is recommended to set up the Docker repository before you start the installation. In this section, we describe in how to configure the Docker repository.

Note: You can install Docker Engine by downloading the rpm package from the Docker website repository and by using the `rpm` Linux command to install it. However, SuSE Linux provides a useful package manager that is called Zypper to facilitate the installation and to install future software updates. After the Docker repository is enabled in the Zypper configuration, you can quickly install and manage this package.

This IBM Redbooks publication describes the installation of Docker EE 2.0 (Docker Engine 17.06-22) and other components, such as UCP and DTR, as listed on the [Docker compatibility matrix website](#).

Before enabling Docker repository, be aware that every repository in Zypper has its own unique number to identify that repository and a URL where repository exists.

Note: All commands that are listed in this section should be run by using the root ID.

To enable the Docker repository, you need the URL of the Docker EE repository that is associated with your subscription. Complete the following steps to get this information:

1. Go to <https://store.docker.com/editions/enterprise/docker-ee-ibm-z>.
2. Each subscription or trial you have access to is listed. Click **Setup** for Docker Enterprise Edition for SUSE Linux Enterprise Server.
3. Copy the URL from the Copy field and save it to a text file. An example of this data is shown in Example 3-1.

Example 3-1 Sample of URL

`https://storebits.docker.com/ee/z/sub-c80ec936-d198-4b0b-8327-35539629b336`

For more information about the supported Docker EE version, see the [Docker compatibility matrix website](#).

With the URL and Docker version information, you can mount the URL to add the Docker repository in Zypper (see Table 3-1).

Table 3-1 Required information for setting up a new repository

Parameter	Value
<DOCKER-EE-URL>	<code>https://storebits.docker.com/ee/z/sub-c811ec936-d198-4b0b-8327-35539629b336</code>
<ARCHITECTURE>	<code>s390x</code>
<VERSION>	<code>17.06</code>

You must replace the information in the command that is shown in Example 3-2 with the information from Table 3-1.

Example 3-2 URL for the Docker repository

```
zypper addrepo "<DOCKER-EE-URL>/sles/12.3/<ARCHITECTURE>/stable-<VERSION>"  
docker-ee-stable
```

Example 3-3 shows the command after replacing the values from Table 3-1.

Example 3-3 Zypper command to add repository

```
zypper addrepo  
"https://storebits.docker.com/ee/z/sub-c811ec936-d198-4b0b-8327-35539629b336/sles/  
12.3/s390x/stable-17.06" docker-ee-stable
```

Next, run the command that is shown in Example 3-3 to add the new repository. The output of the command is shown in Example 3-4.

Example 3-4 Adding the Docker repository

```
itsoslec:~ # zypper addrepo  
"https://storebits.docker.com/ee/z/sub-c811ec936-d198-4b0b-8327-35539629b336/sles/  
12.3/s390x/stable-17.06" docker-ee-stable  
Adding repository 'docker-ee-stable'  
.....  
.....[done]  
Repository 'docker-ee-stable' successfully added  
Enabled: Yes  
Autorefresh: No  
GPG check: Yes  
URI:  
https://storebits.docker.com/ee/z/sub-c811ec936-d198-4b0b-8327-35539629b336/sles/1  
2.3/s390x/stable-17.06
```

To ensure packages that are downloaded from this repository were assigned by Docker, import the public GPG key from the Docker repository. By using the information that is listed Table 3-1, replace <DOCKER-EE-URL>, as shown in Example 3-5.

Example 3-5 rpm import command

```
rpm --import "<DOCKER-EE-URL>/sles/gpg"
```

The command after replacing the values from Table 3-1 is shown in Example 3-6.

Example 3-6 rpm command to import the GPG key

```
rpm --import  
"https://storebits.docker.com/ee/z/sub-c811ec936-d198-4b0b-8327-35539629b336/sles/  
gpg"
```

Run the command that is shown in Example 3-6 to import the key. No command output is shown if the key was successfully imported.

Also, you can run the command that is shown in Example 3-7 to confirm that key was imported to your system. You must validate the Date field to confirm when key was imported.

Example 3-7 Querying GPG imported keys

```
itsoslec:~ # rpm -qi gpg-pubkey-\  
docker@docker.com  
Architecture: (none)  
Install Date: Fri Oct 5 08:37:31 2018  
Group : Public Keys  
Size : 0
```

```
License      : pubkey
Signature    : (none)
Source RPM   : (none)
Build Date   : Wed Feb 22 14:51:11 2017
Build Host   : localhost
Relocations  : (not relocatable)
Packager     : Docker Release (EE rpm) <docker@docker.com>
Summary      : gpg(Docker Release (EE rpm) <docker@docker.com>)
```

Also, confirm if the repository was added and that a file that is named `docker-ee-stable.repo` was created in the `/etc/zypp/repos.d` folder, as shown in Example 3-8.

Example 3-8 Output of `/etc/zypp/repos.d/docker-ee-stable.repo` file

```
itsoslec:~ # cat /etc/zypp/repos.d/docker-ee-stable.repo
[docker-ee-stable]
name=docker-ee-stable
enabled=1
autorefresh=0
baseurl=https://storebits.docker.com/ee/z/sub-c80ec936-d198-4b0b-8327-35539629b336
/sles/12.3/s390x/stable-17.06
type=NONE
```

With the GPG key imported and the repository enabled, you must run the **zypper refresh** command to update the Zypper package index.

Note: If this is the first time you refreshed the package index since the Docker repositories were added, you are prompted to accept the GPG key, and the key's fingerprint is shown. Verify that the fingerprint matches 77FE DA13 1A83 1D29 A418 D3E8 99E5 FF2E 7668 2BC9 and if so, accept the key.

The output of the command is shown in Example 3-9.

Example 3-9 Output of Zypper refresh on the system

```
itsoslec:~ # zypper refresh
Repository 'SLES12 12-0' is up to date.
Retrieving repository 'docker-ee-stable' metadata
.....
..[done]
Building repository 'docker-ee-stable' cache
.....
.....[done]
All repositories have been refreshed.
```

We ensured that we installed the most recent supported version of Docker EE. A supported version is required to receive official Docker support.

As shown in Example 3-10, we list all available Docker package versions and install `docker-ee-2:17.06.2.ee-16-3`, where `2:17.06.2.ee-16-3` is the latest support version.

Example 3-10 Querying and installing a supported Docker EE version

```

itsoslec:~ # zypper search -s --match-exact -t package docker-ee
Loading repository data...
Reading installed packages...

S | Name      | Type   | Version                                     |
Arch | Repository
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
  | docker-ee | package | 2:17.06.2.ee.16-3                         |
s390x | docker-ee-stable
  | docker-ee | package | 2:17.06.2.ee.15-3                         |
s390x | docker-ee-stable
  | docker-ee | package | 2:17.06.2.ee.14-3                         |
s390x | docker-ee-stable
  | docker-ee | package | 2:17.06.2.ee.13-3                         |
s390x | docker-ee-stable
  | docker-ee | package | 2:17.06.2.ee.12-3                         |
s390x | docker-ee-stable
  | docker-ee | package | 2:17.06.2.ee.11-3                         |
s390x | docker-ee-stable
  | docker-ee | package | 2:17.06.2.ee.10-3                         |
s390x | docker-ee-stable
  | docker-ee | package | 2:17.06.2.ee.9-0.0.dev.git20180425.235429.0.c76576b |
s390x | docker-ee-stable
  | docker-ee | package | 2:17.06.2.ee.8-3                          |
s390x | docker-ee-stable
  | docker-ee | package | 2:17.06.2.ee.7-3                          |
s390x | docker-ee-stable
  | docker-ee | package | 17.06.2.ee.6-3                            |
s390x | docker-ee-stable
  | docker-ee | package | 17.06.2.ee.5-1                            |
s390x | docker-ee-stable
  | docker-ee | package | 17.06.2.ee.4-1                            |
s390x | docker-ee-stable
  | docker-ee | package | 17.06.2.ee.3-1                            |
s390x | docker-ee-stable
  | docker-ee | package | 17.06.1.ee.2-1                            |
s390x | docker-ee-stable
  | docker-ee | package | 17.06.1.ee.1-1                            |
s390x | docker-ee-stable

itsoslec:~ # zypper install docker-ee-2:17.06.2.ee.16-3
Loading repository data...
Reading installed packages...
Resolving package dependencies...

The following 2 NEW packages are going to be installed:
  docker-ee libcgroupl

The following package has no support information from its vendor:
  docker-ee

```

```

2 new packages to install.
Overall download size: 22.8 MiB. Already cached: 0 B. After the operation,
additional 96.6 MiB will be used.
Continue? [y/n/...? shows all options] (y): y
Retrieving package libcgroupl-0.41.rc1-10.9.1.s390x
(1/2), 42.4 KiB (103.6 KiB unpacked)
Retrieving: libcgroupl-0.41.rc1-10.9.1.s390x.rpm
.....
... [done]
Retrieving package docker-ee-2:17.06.2.ee.16-3.s390x
(2/2), 22.7 MiB (96.5 MiB unpacked)
Retrieving: docker-ee-17.06.2.ee.16-3.s390x.rpm
..... [done
(1.7 MiB/s)]
Checking for file conflicts:
.....
..... [done]
(1/2) Installing: libcgroupl-0.41.rc1-10.9.1.s390x
.....
. [done]
(2/2) Installing: docker-ee-2:17.06.2.ee.16-3.s390x
.....
[done]

```

The only supported storage driver for Docker EE on SLES is BTRFS. We recommend that you define a separate file system for hosting `/var/lib/docker` to ensure that logs and other files do not fill up your Docker file system.

In the next sections, we describe how to define a new file system by using Logical Volume Manager (LVM) and mounting it on `/var/lib/docker`. For more information about how to create a BTRFS file system, see “Creating a BTRFS file system for `/var/lib/docker`” on page 58.

Note: After you create the file system for `/var/lib/docker`, see 3.3, “Docker verification” on page 61 for more information about validating your Docker installation.

For more information about the Docker installation on SuSE, see [this website](#).

3.1.2 Red Hat Linux

Although you can download the RPM files and perform the task manually, the best way to install Docker on Red Hat is to enable the official repository and use the Yum tool to perform an easy and quick installation.

In this book, we describe the installation of Docker EE 2.0 with a Docker Engine 17.06.2-ee-16.

Prerequisites

The following prerequisites must be met:

- ▶ Use RHEL 64-bit 7.1 and higher on x86_64, s390x, or ppc64le (not ppc64).
- ▶ Use storage driver overlay2 or devicemapper (direct-lvm mode in production).
- ▶ Disable SELinux on s390x (IBM Z) systems before installing or upgrading.

Enabling and installing repository

To enable the Docker repository, you need the URL of the Docker EE repository that is associated with your subscription. Complete the following steps to get this information:

1. Go to <https://store.docker.com/editions/enterprise/docker-ee-ibm-z>.
2. Each subscription or trial that you can access is listed. Click **Setup** for Docker Enterprise Edition for Red Hat Enterprise Linux.
3. Copy the URL from the Copy field and save it to a text file. A sample URL that provided by Docker is shown in Example 3-11.

Example 3-11 Sample of URL provided by Docker

```
https://storebits.docker.com/ee/z/n0a-i1b12ct3-do4en-5fc-6ga7-hr8i19jo0ks11
```

4. Before the installation is started, ensure that no other Docker versions are running and remove any repository for Docker (see Figure 3-1 and Figure 3-2).

```
[root@itsor86a ~]# yum remove docker \  
> docker-client \  
> docker-client-latest \  
> docker-common \  
> docker-latest \  
> docker-latest-logrotate \  
> docker-logrotate \  
> docker-selinux \  
> docker-engine-selinux \  
> docker-engine \  
> docker-ce  
Loaded plugins: langpacks, product-id, search-disabled-repos,  
subscription-manager  
No Match for argument: docker  
No Match for argument: docker-client  
No Match for argument: docker-client-latest  
No Match for argument: docker-common  
No Match for argument: docker-latest  
No Match for argument: docker-latest-logrotate  
No Match for argument: docker-logrotate  
No Match for argument: docker-selinux  
No Match for argument: docker-engine-selinux  
No Match for argument: docker-engine  
No Match for argument: docker-ce  
No Packages marked for removal  
[root@itsor86a ~]#
```

Figure 3-1 Removing all previous docker software

```
[root@itsor86a ~]# rm /etc/yum.repos.d/docker*.repo
```

Figure 3-2 Removing any existent Docker repository

5. Export the personalized URL that you saved as an environment variable, as shown in Figure 3-3.

```
[root@itsor86a ~]# export
DOCKERURL="https://storebits.docker.com/ee/z/n0a-i1b12ct3-do4en-5fc-6ga7-hr8i19
jo0ks11"
```

Figure 3-3 Exporting the URL as an environment variable

6. Store the variable DOCKERURL in a yum variable, as shown in Figure 3-4.

```
[root@itsor86a ~]# echo "$DOCKERURL/rhel" > /etc/yum/vars/dockerurl
```

Figure 3-4 DOCKERURL variable stored for yum

7. Store your operating system version string in /etc/yum/vars/dockerosversion, as shown in Figure 3-5.

```
[root@itsor86a ~]# echo "7.5" > /etc/yum/vars/dockerosversion
```

Figure 3-5 Storing the OS version for Yum

8. Depending on the choice of your storage, install any required packages; for example, device-mapper and lvm2, and yum-utils, for which Red Hat provides the yum-utility-manager (see Figure 3-6).

```
[root@itsor86a ~]# yum install -y yum-utils \
device-mapper-persistent-data \
lvm2
```

Figure 3-6 Required packages

9. Enable the extra Red Hat and Docker repository (see Figure 3-7). We use the DOCKERURL variable that was set in the previous steps. Ensure that the Red Hat repository is configured (including the Extra Packages for Red Hat Enterprise Linux) because Docker might require more software).

```
[root@itsor86a ~]# yum-config-manager --enable rhel-7-server-extras-rpms

[root@itsor86a ~]# yum-config-manager \
--add-repo \
"$DOCKERURL/rhel/docker-ee.repo"
Loaded plugins: langpacks, product-id, subscription-manager
adding repo from:
https://storebits.docker.com/ee/z/sub-66227f0c-e75d-4930-b2b0-7315c8111a96/rhel
/docker-ee.repo
grabbing file
https://storebits.docker.com/ee/z/sub-66227f0c-e75d-4930-b2b0-7315c8111a96/rhel
/docker-ee.repo to /etc/yum.repos.d/docker-ee.repo
repo saved to /etc/yum.repos.d/docker-ee.repo
[root@itsor86a ~]#
```

Figure 3-7 Enabling the repositories

You can now see all of the available Docker versions in the repository by using the `~ yum list docker-ee --showduplicates | sort -r` command, as shown in Figure 3-8. For the purposes of this book, we installed version 17.06.2.

```
[root@itsor86a ~]# yum list docker-ee --showduplicates | sort -r
: manager
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-
Installed Packages
docker-ee.s390x      2:17.06.2.ee.16-3.e17.rhel      docker-ee-stable-17.06
docker-ee.s390x      2:17.06.2.ee.16-3.e17.rhel
@docker-ee-stable-17.06
docker-ee.s390x      2:17.06.2.ee.15-3.e17.rhel      docker-ee-stable-17.06
docker-ee.s390x      2:17.06.2.ee.14-3.e17.rhel      docker-ee-stable-17.06
docker-ee.s390x      2:17.06.2.ee.13-3.e17.rhel      docker-ee-stable-17.06
docker-ee.s390x      2:17.06.2.ee.12-3.e17.rhel      docker-ee-stable-17.06
Available Packages
[root@itsor86a ~]#
```

Figure 3-8 Listing all available Docker versions

10. With the repository set, start the installation (see Figure 3-9, Figure 3-10 on page 45, and Figure 3-11 on page 45).

```
[root@itsor86a ~]# yum -y install docker-ee-17.06.2.ee.16
Resolving Dependencies
--> Running transaction check
---> Package docker-ee.x86_64 2:17.06.2.ee.16-3.e17 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
=====
Package           Arch           Version
Repository         Size
=====
=====
Installing:
```

Figure 3-9 Installing Docker from repository

```

docker-ee          x86_64          2:17.06.2.ee.16-3.e17
docker-ee-stable-17.06          28 M

Transaction Summary
=====
=====
Install 1 Package

Total download size: 28 M
Installed size: 82 M
Downloading packages:
warning:
/var/cache/yum/x86_64/7Server/docker-ee-stable-17.06/packages/docker-ee-17.06.2
.ee.16-3.e17.x86_64.rpm: Header V4 RSA/SHA512 Signature, key ID 76682bc9: NOKEY
Public key for docker-ee-17.06.2.ee.16-3.e17.x86_64.rpm is not installed
docker-ee-17.06.2.ee.16-3.e17.x86_64.rpm
| 28 MB 00:00:03
Retrieving key from
https://storebits.docker.com/ee/z/sub-66227f0c-e75d-4930-b2b0-7315c8111a96/rhel
/gpg

```

Figure 3-10 Installing Docker from repository (cont.)

```

Importing GPG key 0x76682BC9:
  Userid      : "Docker Release (EE rpm) <docker@docker.com>"
  Fingerprint: 77fe da13 1a83 1d29 a418 d3e8 99e5 ff2e 7668 2bc9
  From        :
https://storebits.docker.com/ee/z/sub-66227f0c-e75d-4930-b2b0-7315c8111a96/rhel
/gpg
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : 2:docker-ee-17.06.2.ee.16-3.e17.x86_64
1/1
  Verifying  : 2:docker-ee-17.06.2.ee.16-3.e17.x86_64
1/1

Installed:
  docker-ee.x86_64 2:17.06.2.ee.16-3.e17

Complete!
[root@itsor86a yum.repos.d]#

```

Figure 3-11 Installing Docker from repository (cont.)

If you plan to use device-mapper, you must ensure that you followed the instructions that are available [at this website](#) before you start Docker.

Otherwise, you can start Docker now (see Figure 3-12).

```
[root@itsor86a ~]# systemctl start docker
```

Figure 3-12 Starting Docker

After you complete this process, your Docker environment is up and running. For more information about validating the installation, see 3.3, “Docker verification”.

As a quick test, you can run a simple “hello world” program (see Figure 3-13).

```
[root@itsor86a ~]# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
d1725b59e92d: Pull complete
Digest: sha256:0add3ace90ecb4adbf777e9aacf18357296e799f81cab9fde470971e499788
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.
    (amd64)
 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 ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

[root@itsor86a ~]#
```

Figure 3-13 Testing the Docker installation

3.1.3 Ubuntu Linux

Docker EE is available for Ubuntu 18.04. In this section, we describe how to install Docker EE 2.0 with a Docker engine 17.06.2-ee-16.

Prerequisites

The following prerequisites must be met:

- ▶ Update apt-get repository.
- ▶ Use storage driver for overlay2 or advanced multi-layered unification file system (aufs), which are the supported storage drivers.

Enabling and installing repository

On Ubuntu Linux, the default repository is ready to install Docker. If you must enable the Docker repository, you need the URL of the Docker EE repository that is associated with your subscription to proceed with the steps to enable it. Complete the following steps:

1. Go to <https://store.docker.com/editions/enterprise/docker-ee-ibm-z>.
2. Each subscription or trial that you can access is listed. Click **Setup** for Docker Enterprise Edition for Ubuntu.
3. Copy the URL from the Copy field and save it to a text file (see Example 3-12).

Example 3-12 Sample of URL provided by Docker

```
https://storebits.docker.com/ee/z/n0a-i1bu2ct3-co3in-8fc-6ga7-hr9i17jo0ks34
```

To enable the repository, complete the steps that are provided on [the Docker website](#).

In our lab, we used the default repository for Ubuntu Linux to perform the Docker EE installation (see Example 3-13).

Example 3-13 Output of the default Ubuntu repository

```
root@itsoubua:~# cat /etc/apt/sources.list

deb http://us.ports.ubuntu.com/ubuntu-ports/ bionic main restricted
deb http://us.ports.ubuntu.com/ubuntu-ports/ bionic-updates main restricted
deb http://us.ports.ubuntu.com/ubuntu-ports/ bionic universe
deb http://us.ports.ubuntu.com/ubuntu-ports/ bionic-updates universe
deb http://us.ports.ubuntu.com/ubuntu-ports/ bionic multiverse
deb http://us.ports.ubuntu.com/ubuntu-ports/ bionic-updates multiverse
deb http://us.ports.ubuntu.com/ubuntu-ports/ bionic-backports main restricted
universe multiverse
deb http://ports.ubuntu.com/ubuntu-ports bionic-security main restricted
deb http://ports.ubuntu.com/ubuntu-ports bionic-security universe
deb http://ports.ubuntu.com/ubuntu-ports bionic-security multiverse
```

Before starting the installation, we updated and upgraded the apt package; then, we proceeded with the Docker installation (see Example 3-14).

Example 3-14 Update and upgrade apt package and perform Docker install

```
root@itsoubub:~# cat /etc/issue
Ubuntu 18.04.1 LTS \n \l
root@itsoubub:~# apt-get update
root@itsoubub:~# apt-get upgrade
root@itsoubub:~# apt-get install docker
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed
docker
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 12.9 kB of archives.
After this operation, 44.0 kB of additional disk space will be used.
Get:1 http://us.ports.ubuntu.com/ubuntu-ports bionic/universe s390x docker s390x
1.5-1build1 [12.9 kB]
```

```

Fetched 12.9 kB in 0s (435 kB/s)
Selecting previously unselected package docker.
(Reading database ... 62681 files and directories currently installed.)
Preparing to unpack .../docker_1.5-1build1_s390x.deb ...
Unpacking docker (1.5-1build1) ...
Processing triggers for man-db (2.8.3-2) ...
Setting up docker (1.5-1build1) ...
root@itsoubub:~#
root@itsoubub:~# apt-get install docker-compose
root@itsoubub:~# apt-get install docker-registry

```

After the installation completed, we validated that Docker EE was installed on Ubuntu and was ready to be used (see Example 3-15).

Example 3-15 Validating that Docker is installed on Ubuntu

```

root@itsoubub:~# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
root@itsoubub:~# ps -ef |grep docker
root      16198          1  0 19:11 ?                00:00:00 /usr/bin/dockerd -H fd://
root      16218 16198      0  0 19:11 ?                00:00:00 docker-containerd --config
/var/run/docker/containerd/containerd.toml
docker-+ 17843          1  0 19:12 ?                00:00:00 /usr/bin/docker-registry serve
/etc/docker/registry/config.yml
root      18070 18038      0  0 19:17 pts/0          00:00:00 grep --color=auto docker

```

3.2 Docker storage

Understanding Docker images and the concept of layers and their benefits is vital for system administrators (SAs), developers, or anyone who wants to run and manage Docker containers for critical applications in production environments.

Docker storage drivers (also called *graph drivers*) manage images, containers, and their layers on a Docker host. Layers allow for reuse in more containers per Docker host and faster start-up and download time where base layers are in cache.

A Docker image is a read-only template for the root file systems that is used as baseline to build containers. In the virtual machine (VM) world, the Docker images can be compared as read-only golden images.

Anyone who worked with VMs on IBM Z know a Linux for System z image is a server that contains mini-disks that can be cloned to create VMs to be used for applications and middleware that run the business. As with VM images, a Docker image provides the same purpose to allow you to instantiate new containers, but they have some important differences.

A VM image on IBM Z is a Linux virtual VM that was created with a base set of applications that is installed and customized to make the process to deploy new guests easy and efficient. This process reduces human errors and labor during a Linux server deployment.

Note: The VM images often are accessed in read-only mode to avoid someone inadvertently modify it. A Docker image contains a list of read-only layers that represents differences in the file systems.

When you specify instructions (FROM, RUN, COPY, and so on) by way of Dockerfile, this specification triggers a change to the base image and creates a layer. A *layer* is a collection of files and directories that are generated from these instructions to create an image.

These layers can be shared across images, which reduces the need for disk space on the Docker host server. It also provides more efficient memory usage by allowing the Linux kernel to include only a single copy of a file to be read from several containers.

In summary, the Dockerfile is used to extend an image by adding or modifying files and directories on top of base image. These base images are used as the foundation for building all containerized applications.

Although you have many images that are available at [the Docker Hub website](#) and the [Docker Store website](#), select the certified vendors base images to ensure the support, reliability, and security for your production application deployments.

By using a certified base image, you receive updates, understand the changes that are made with each update, and are fully supported by your vendors. In this book, we built containers that used Red Hat, SuSE, and Ubuntu base images.

Figure 3-14 shows the creation of a container that uses an Ubuntu base image and the modifications by using Dockerfile instructions to create the layers.

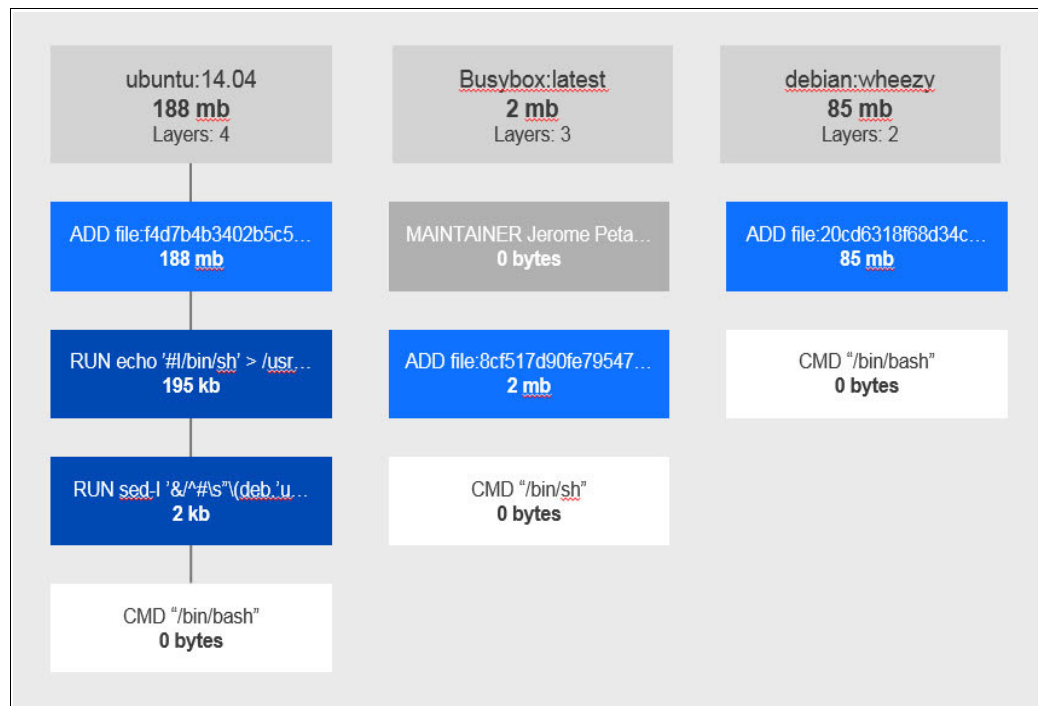


Figure 3-14 Creating a container by using Ubuntu

The output of the `docker history ubuntu` command is shown in Figure 3-15.

```
itsosleb:~ # docker history ubuntu
IMAGE          CREATED          CREATED BY
SIZE          COMMENT
2a1a61f48b0a  6 weeks ago    /bin/sh -c #(nop) CMD ["/bin/bash"]
0B
<missing>     6 weeks ago    /bin/sh -c mkdir -p /run/systemd &&
echo '...    7B
<missing>     6 weeks ago    /bin/sh -c sed -i
's/^#\s*\s*(deb.*universe\... 2.83kB
<missing>     6 weeks ago    /bin/sh -c rm -rf /var/lib/apt/lists/*
0B
<missing>     6 weeks ago    /bin/sh -c set -xe && echo
'#!/bin/sh' >... 745B
<missing>     6 weeks ago    /bin/sh -c #(nop) ADD
file:3990672df51fc41... 83.2MB
itsosleb:~ #
```

Figure 3-15 Output of `docker history` command

The image IDs that are shown in Figure 3-15 are missing because those layers were built on another Docker host and likely were built from another image. Therefore, the history of that image does not have the historical data located locally. This issue is not considered an error and you can ignore it.

If you run the `docker history` command and formatted the output, it might hide other layers that do not have an Image ID. However, this result does not mean that the layer does not exist. The command that is shown in Figure 3-16 shows the output of the `docker images` command that uses the `--format` option to query an Ubuntu image.

```
itsoslec:# docker images --format "{{.ID}}: {{.CreatedSince}}" ubuntu
2a1a61f48b0a: 6 weeks ago
itsoslec:#
```

Figure 3-16 Querying `docker images` by using `format` option

These layers are stacked one over the other (see Figure 3-16) and form the basis of the container root file system. The Docker storage driver, also called *graph drivers*, stacks and maintains the different layers, images, and containers.

Another function that is provided by the storage driver is the sharing management of layers across images. This function makes building, pulling, pushing, and copying images fast and saves on storage.

Layers have no notion of an image or of belonging to an image; instead, they become collections of files and directories that can be shared across images. Layers and images become separated.

However, the layers are named with a randomly generated cache ID. The link between a layer and its cache ID is known only to Docker Engine for security reasons.

The main function of storage driver is to manage these updates and share layers across images. Layers, images, containers, and locally named volumes are stored in the `/var/lib/docker` folder. Its contents vary, depending on the storage driver that Docker selected to manage this data.

to avoid unexpected issues with Docker host, do not attempt to update files in the `/var/lib/docker` folder manually. If you must perform any maintenance on that folder, use Docker commands, as such `docker prune`.

Note: As good practice, back up this folder before any further maintenance is done.

The primary benefit to split off `/var/lib/docker` is the ability to set different mount options and select a different file system type. Another benefit is to prevent certain types of applications to fill up `/var` quickly. Therefore, it is recommended to mount `/var/lib/docker` on a separate partition or volume to not affect the Docker host operating system.

For more information about creating and migrating the content of `/var/lib/docker` directory to separate file systems, see “Creating a BTRFS file system for `/var/lib/docker`” on page 58.

You must verify the different requirements and compatible versions for each Docker distribution that is available in the compatibility matrix to ensure that you receive the correct support from the vendor. For more information, see the Docker compatibility matrix at [this website](#).

After reviewing the information about the compatibility matrix from Docker website, we chose the components that are listed in Table 3-2.

Table 3-2 Overview of ITSO environment

Component	Version
Host Operating System	SuSE Linux Enterprise Server 12 SP3
Storage Driver	overlay2
EE Version	17.06 starting with 17.06.2-ee-16

Docker Community Edition uses overlay2 as default (overlayFS driver). Previously, aufs was used by default. However, aufs is no longer used because of the number of benefits the overlay driver provides. The overlay2 features more improvements, better performance, and the ability to work more efficiently than aufs and also with overlay.

The tables in the [Docker compatibility matrix](#) list the supported component versions that are available for the Docker Enterprise Edition. Although some storage drivers are available on the Linux kernel, the BTRFS is preferred. Otherwise, you can use overlay2 as storage driver for your Docker EE (s390x version).

Note: You must follow the [Docker compatibility matrix](#) to ensure that you receive official support from Docker team.

OverlayFS

OverlayFS is supported on IBM Z by the Docker organization. It is popular in the community and in most of the other distributions, it is the default driver for Docker CE. In this section, we describe this driver and how it works.

As described in Table 3-2, our environment is going to be configured to use BTRFS driver in our Docker host servers. However, the information in this section regarding overlayFS is important for users and system administrators that must support Docker applications.

To check whether the Linux kernel driver for your Docker host supports the overlayFS module, run the `modinfo` Linux command, as shown in Example 3-16. Verify the output and confirm the driver availability supports overlay. In general, they should be available on kernels 3.18 and higher.

Note: The `modinfo` command is used to show more information about a Linux kernel module.

Example 3-16 Output of `modinfo` command for overlay driver

```

itsosleb:~ # modinfo overlay
filename:      /lib/modules/4.4.156-94.57-default/kernel/fs/overlayfs/overlay.ko
alias:        fs-overlayfs
alias:        fs-overlay
license:      GPL
description:  Overlay filesystem
author:       Miklos Szeredi <miklos@szeredi.hu>
srcversion:   5273297588F9E3388FDC816
depends:
supported:   yes
intree:      Y
vermagic:    4.4.156-94.57-default SMP mod_unload modversions
signer:      SUSE Linux Enterprise Secure Boot Signkey
sig_key:     3F:B0:77:B6:CE:BC:6F:F2:52:2E:1C:14:8C:57:C7:77:C7:88:E3:E7
sig_hashalgo: sha256
itsosleb:~ #

```

The overlayFS rapidly gained popularity and it is available for Docker CE and EE. You should find two overlayFS drivers available in Docker: `overlay` and `overlay2`. The latest one is the newer driver and recommended for IBM Z.

When overlayFS is used, the Docker documentation recommends the use of `overlay2` instead of `overlay` because it is more stable, has simpler implementation, features significant improvements over the original `overlay` implementation, and is more efficient in terms of node utilization. Also, if you did not decide which storage driver to use, see the [Docker compatibility matrix](#).

This driver is a type of union file system that combines two or more directories that are called “lowerdir” and “upperdir” to produce a single merged directory that is referenced as “merged” (see Table 3-3).

Table 3-3 OverlayFS Overview

Directory name	Description
lowerdir	Read-only layer that is potentially shared with other containers
upperdir	Writable layer (top layer) to handle file system changes
merged	Combined view of lowerdir and upperdir
workdir	Used internally by OverlayFS driver (a working directory)

If you pull an image from registry repository that contains more than two layers (known as *multi-layered*), the overlayFS driver builds a directory infrastructure under `/var/lib/docker/overlay2` (if `overlay2` is the selected driver) where each image layer has its own directory and creates hard links to reference data that can be shared with lower layers.

Note: Docker supports several storage drivers with different capabilities, technologies, and Linux distribution affinities. For more information, see [this website](#).

Often, the Docker host stores data about its image and container layers at `/var/lib/docker`. This local storage area is populated when images are pulled from registries or when a container is started for the first time.

If you examine the `/var/lib/docker/overlay2` folder, you should see a content of folders that resemble hashes that do not correspond to image layer IDs. In Example 3-17, you can see a list of directories under the `overlay2` folder for our Docker host.

Example 3-17 Output of `ls` command for `/var/lib/docker/overlay2` command

```
itsosleb:~ # ls -l /var/lib/docker/overlay2/
total 4
drwx----- 4 root root    55 Oct 10 16:10
25149503ce77595c096e778293051176234863801dccc4d49d84d3074fcd66a5
drwx----- 4 root root    55 Oct 10 16:10
31fd9ef905b0103c5205e0cac41a26d5f34c2b6caf4baeea9f9009d075597940
drwx----- 4 root root    55 Oct 10 16:10
624f0e2df98c89cbbd64fe659e515f2ff8a8dc7e3fc842512e223c0fe0462559
brw----- 1 root root 254, 0 Oct 10 15:35 backingFsBlockDev
drwx----- 3 root root    30 Oct 10 16:10
c3baff758f92c2d6a1daf071c4898e32a07803060f4599ff6325b0b9797b1fa0
drwx----- 4 root root    55 Oct 10 16:10
f6db0cfa12fca5979fcf2896fd5cb739284b9023174515d587982f7399053a78
drwx----- 2 root root  4096 Oct 10 16:10 l
itsosleb:~ #
```

When the `docker run` command is run to start a new container, this new instance has two layers: one read-only part (lower layer) and another write part (upper layer). These layers merged by the storage driver to create the container file system that is available for this instance.

In addition to the layer folders, an `l` (lowercase L) directory is created to store information about the shortened layer identifiers as symbolic links. According to Docker documentation, these identifiers are used to avoid encountering the page size limitation on arguments to the `mount` command. Therefore, the shorter directory names are easily used and ensure the compatibility for providing shorter directory names shorter provides.

The layers for this new container are shown in Figure 3-17.

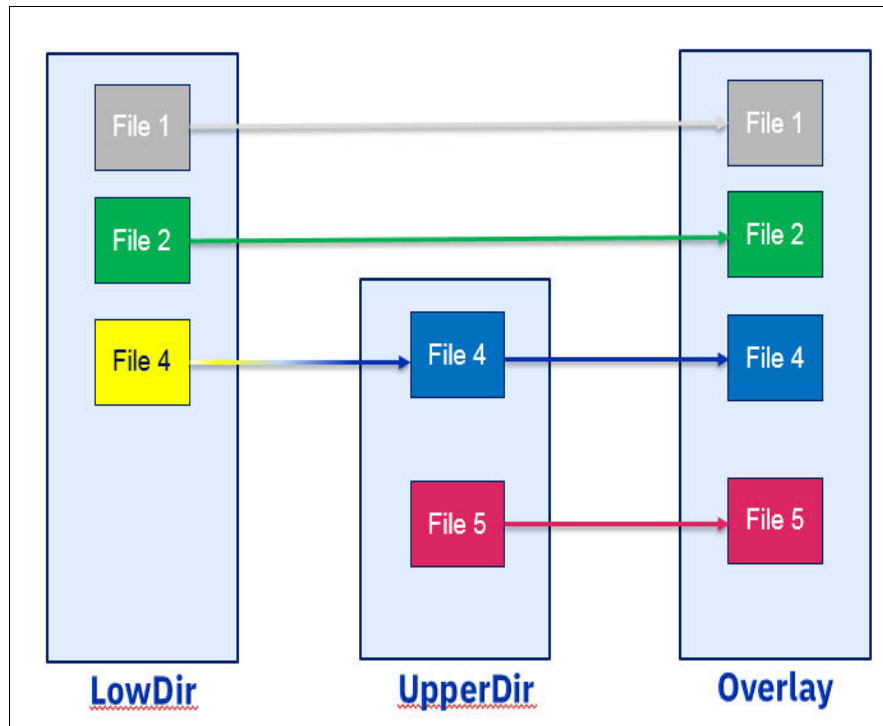


Figure 3-17 OverlayFS architecture

The commands that are shown in Example 3-18 shows how to run a container and list folders that are created by the storage driver.

Example 3-18 Initializing a container and listing the read/write layer

```

itsosleb:~ # docker container run -d nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
599d69132c05: Pull complete
bc12a10c7cc5: Pull complete
54372546fe97: Pull complete
Digest: sha256:9ad0746d8f2ea6df3a17ba89eca40b48c47066dfab55a75e08e2b70fc80d929e
Status: Downloaded newer image for nginx:latest
9a862aea0126b65ef06ad7a4c436451df6a05278fb2c873f8a12881f0c80f67b

itsosleb:~ # docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS            PORTS              NAMES
9a862aea0126       nginx              "nginx -g 'daemon ..." 11 seconds ago
Up 11 seconds      80/tcp            hopeful_hoover

itsosleb:~ # ls -l /var/lib/docker/overlay2/1/ |grep init
lrwxrwxrwx 1 root root 77 Oct 10 16:15 RE4BDZJJBUE3MEXBWD7UFA2JQV ->
../4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07ba5fac5-init/diff

itsosleb:~ # ls -l /var/lib/docker/overlay2/1/ |grep -i
4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07ba5fac5
lrwxrwxrwx 1 root root 72 Oct 10 16:15 HGOHQS2UK3DPUTJQITQZFLQC6F ->
../4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07ba5fac5/diff

```

```
lrwxrwxrwx 1 root root 77 Oct 10 16:15 RE4BDZJJBUE3MEXBWD7UFA2JQV ->
../4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07ba5fac5-init/diff
```

The overlay2 storage driver creates two other directories that include the container ID (and init) to store the updates for the read/write layer for the running container.

As shown in Example 3-19, we ran some commands to connect to a running container with **9a862aea0126** ID and create an empty file that is called `/tmp/123.txt`.

Example 3-19 Connecting to a container and creating an empty file

```
itsosleb:~ # docker ps
CONTAINER ID      IMAGE          COMMAND          CREATED
STATUS           PORTS         NAMES
9a862aea0126     nginx         "nginx -g 'daemon ..." 3 minutes ago
Up 3 minutes     80/tcp       hopeful_hoover
```

```
itsosleb:~ # docker exec -it 9a862aea0126 bash
root@9a862aea0126:/# touch /tmp/123.txt
root@9a862aea0126:/# exit
exit
```

To confirm that the file was created under the read/write layer, you can run the command that is shown in Example 3-20.

Example 3-20 Searching the new file

```
itsosleb:~ # find /var/lib/docker -name \123.txt
/var/lib/docker/overlay2/4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07
ba5fac5/diff/tmp/123.txt
/var/lib/docker/overlay2/4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07
ba5fac5/merged/tmp/123.txt
itsosleb:~ #
```

To list all layers for a container image, you must use the **docker container ls** command passing **-no-trunc** to show the IDs (see Example 3-21).

Example 3-21 Listing full container IDs with --notrunc option

```
itsosleb:~ # docker image inspect nginx -f '{{.RootFS.Layers}}'
[sha256:20d026287261aea4a4623254f82ce85a6e1f94eeae2b31945640d205c2b06123
sha256:fb6654eaa293d79c102fe3e0792aa940d90c0a957924269df152966a718e44b2
sha256:2715cd98b363ae3ff6620c24f87c18f95d063eaa67f1d45d9426f3c11bf58034]
itsosleb:~ #
```

Complete the following steps to get more information about the layers for a container:

1. Find the container ID or container name (see Example 3-22).

Example 3-22 Checking running containers

```
itsosleb:~ # docker container ps -a
CONTAINER ID      IMAGE          COMMAND          CREATED
STATUS           PORTS         NAMES
9a862aea0126     nginx         "nginx -g 'daemon ..." 6 minutes ago
Up 6 minutes     80/tcp       hopeful_hoover
itsosleb:~ #
```

2. Run command that is shown in Example 3-23 to find the container's layers.

Example 3-23 Inspecting a container to list the lowerdir, mergedir, upperdir and workdir

```
# Please note that 9a862aea0126 is the container ID for nginx
itsosleb:~ # docker container inspect 9a862aea0126 -f '{{.GraphDriver.Data}}'

map[LowerDir:/var/lib/docker/overlay2/4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07ba5fac5-init/diff:/var/lib/docker/overlay2/acfcd3955054f504a6efb5da9ab0b323325b05e1b66084858edb6cf91bdfd35c/diff:/var/lib/docker/overlay2/d8f4396a30ec38f26e5dba4406ce72d641b03f6b3f0de1b79076e2963e2e15d2/diff:/var/lib/docker/overlay2/94f898959af836a271f25cf565bc1ddb1e447b3bb7c387518a56770e2a7fa819/diff
MergedDir:/var/lib/docker/overlay2/4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07ba5fac5/merged
UpperDir:/var/lib/docker/overlay2/4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07ba5fac5/diff
WorkDir:/var/lib/docker/overlay2/4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07ba5fac5/work]
itsosleb:~ #
```

Also, you can use command that is shown in Example 3-24 to query for each layer separately. By using **-f**, you can filter for the information for which you are looking.

Example 3-24 Querying each layer separately

```
# Querying for LowerDir
itsosleb:~ # docker container inspect 9a862aea0126 -f
'{{.GraphDriver.Data.LowerDir}}'
/var/lib/docker/overlay2/4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07ba5fac5-init/diff:/var/lib/docker/overlay2/acfcd3955054f504a6efb5da9ab0b323325b05e1b66084858edb6cf91bdfd35c/diff:/var/lib/docker/overlay2/d8f4396a30ec38f26e5dba4406ce72d641b03f6b3f0de1b79076e2963e2e15d2/diff:/var/lib/docker/overlay2/94f898959af836a271f25cf565bc1ddb1e447b3bb7c387518a56770e2a7fa819/diff

# Querying for MergeDir
itsosleb:~ # docker container inspect 9a862aea0126 -f
'{{.GraphDriver.Data.MergedDir}}'
/var/lib/docker/overlay2/4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07ba5fac5/merged

# Querying for UpperDir
itsosleb:~ # docker container inspect 9a862aea0126 -f
'{{.GraphDriver.Data.UpperDir}}'
/var/lib/docker/overlay2/4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07ba5fac5/diff

# Querying for WorkDir
itsosleb:~ # docker container inspect 9a862aea0126 -f
'{{.GraphDriver.Data.WorkDir}}'
/var/lib/docker/overlay2/4a633fc2127434d1ab937d43d8e9147de48da201ddeb534b58a094b07ba5fac5/work
itsosleb:~ #
```

The LowerDir is read-only layer and the UpperDir is read/write layer. For more information, see the Docker documentation or Table 3-3 on page 52.

To find the full container ID for your containers, you can use command that is shown in Example 3-25.

Note: If you specify `--no-trunc`, the full container ID is shown.

Example 3-25 Listing running containers with `--no-trunc`

```
itsosleb:~ # docker container ls --no-trunc
CONTAINER ID           IMAGE
COMMAND                STATUS      PORTS
NAMES
9a862aea0126b65ef06ad7a4c436451df6a05278fb2c873f8a12881f0c80f67b  nginx
"nginx -g 'daemon off;'"  9 minutes ago    Up 9 minutes    80/tcp
hopeful_hoover
itsosleb:~ #
```

BTRFS

For Docker EE on IBM Z, we can also use BTRFS storage driver to manage our container Layers (Files and Directories) of Docker images and containers.

Docker has multiple storage drivers that allow you to work with the underlying storage devices and use the file systems resources. To make efficient use of layered images and containers, a storage driver must be configured to advantage of copy on write. Specifically, Linux kernel contains various file systems modules that can be used to manage storage efficiently.

However, for Docker on IBM Z, the BTRFS must be configured on SuSE Linux to obtain official support from Docker team. Two main advantages of this driver over most driver types are that it uses copy on write extensively and file system level snapshotting.

To check whether the Linux kernel driver for your Docker host supports the BTRFS module, run the `modinfo` Linux command, as shown in Figure 3-18. Verify the output and confirm that the driver availability supports BTRFS (highlighted in bold in Figure 3-18).

```
itsoslec: # modinfo btrfs
filename:      /lib/modules/4.4.156-94.57-default/kernel/fs/btrfs/btrfs.ko
license:      GPL
alias:        devname:btrfs-control
alias:        char-major-10-234
alias:        fs-btrfs
srcversion:   6C9650B4874D020A776AE29
depends:       raid6_pq,xor
supported:   yes
intree:       Y
vermagic:     4.4.156-94.57-default SMP mod_unload modversions
signer:       SUSE Linux Enterprise Secure Boot Signkey
sig_key:      3F:B0:77:B6:CE:BC:6F:F2:52:2E:1C:14:8C:57:C7:77:C7:88:E3:E7
sig_hashalgo: sha256
parm:         allow_unsupported:Allow using feature that are out of supported
scope
```

Figure 3-18 Output of `modinfo` command for BTRFS driver

The storage drivers store image layers and containers in the `/var/lib/docker` folder. If Docker Engine is configured to use BTRFS, each layer is stored in its own BTRFS subvolume that is in the `/var/lib/docker/btrfs/subvolumes` folder.

The command that is shown in Example 3-26 can be used to show some subvolumes that were automatically created for Docker layers. The output can vary, depending on the number of images and containers that are in your server.

Example 3-26 Querying BTRFS subvolumes under /var/lib/docker file system

```
itsoslec: # btrfs subvolume list /var/lib/docker
ID 257 gen 384 top level 5 path
btrfs/subvolumes/3ac7cbd5533a17258aae9a944f9ff0fcd29e786449f1617d5636476cf3493714
ID 261 gen 384 top level 5 path
btrfs/subvolumes/a64f90e4a0b262f7f53baab31662ec3a44a70a40e79f642a1190c3e9f435cd9c
ID 262 gen 344 top level 5 path
btrfs/subvolumes/44236cd0f29fc137c5cb4ff804b4fd9953c618d5ad421d7082010e54d6eac1c1
ID 263 gen 344 top level 5 path
btrfs/subvolumes/46a52bdeebbfce2ef0926d00df21a9864b6a3cfad0d078fdbfb1f6a9e196806
ID 264 gen 344 top level 5 path
btrfs/subvolumes/d3e1e26c3e758418c2be3d0a7532fd327388b80720ca221b06b2a2bf23d04efa
ID 265 gen 386 top level 5 path
btrfs/subvolumes/4e50aa3a669ee3af9a63752f78840e582b231b1066804afbdb1ec38918635935
ID 268 gen 344 top level 5 path
btrfs/subvolumes/328e9f5d0b63d2645c6a89660cd186c5d35533340a40d299326b576d4094d3a4
ID 269 gen 344 top level 5 path
btrfs/subvolumes/6710e5d47247782c82cce38922b05a0a3dc0242e29d92919e427b52bd6144401
ID 306 gen 344 top level 5 path
***** SNIPPET OUTPUT *****
```

Note: The `btrfs` command that is shown in Example 3-26 on page 58 works because our environment includes `/var/lib/docker`, which is mounted in its own BTRFS file system. You can get a list of BTRFS file systems by using the `btrfs filesystem show` command, as shown in Figure 3-19.

```
itsoslec: # btrfs filesystem show
Label: none  uuid: 87ecb261-08e3-4b8f-ab6c-12e7dfd8e2ed
  Total devices 1 FS bytes used 8.02GiB
  devid    1 size 18.44GiB used 11.57GiB path /dev/dasda3

Label: none  uuid: aeb3e08a-610d-4468-835a-d9353718dbf0
  Total devices 1 FS bytes used 4.28GiB
  devid    1 size 26.00GiB used 10.31GiB path
/dev/mapper/dockervg-var_lib_docker
```

Figure 3-19 Showing the BTRFS file systems

For more information about Docker storage drivers, see [this website](#).

Creating a BTRFS file system for /var/lib/docker

By default, the containers are created under the `/var/lib/docker`; therefore, it is recommended to create the containers on a separate logical volume under Logical Volume Manager (LVM) and preferably format it by using BTRFS.

Complete the following steps to define a new file system by using LVM and mount it on `/var/lib/docker`:

1. Add disks to your Linux on Z server. For our environment, we added one disk with disk number 500. The commands that are shown in Example 3-27 indicate that the disk is not active.

Example 3-27 Formatting and partitioning the disks

```
tsoslec:~ # /sbin/dasdfmt -b 4096 -y -f /dev/dasde -m 500
Printing hashmark every 500 cylinders.
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
Finished formatting the device.
Rereading the partition table... ok

itsoslec:~ # fdasd -a /dev/dasde
reading volume label ...: VOL1
reading vtoc .....: ok

auto-creating one partition for the whole disk...
writing volume label...
writing VTOC...
rereading partition table...
itsoslec:~ #
```

2. Format and partition the disk.

The use of LVM is vital to reducing the number of outages to your containers. Also, it helps to increase a file system when needed without taking down all processes. The command that is shown in Example 3-28 shows how to add the disks into the LVM.

Example 3-28 Adding disks into LVM

```
itsoslec:~ # pvcreate /dev/dasde1
Physical volume "/dev/dasde1" successfully created
```

3. After disks are available in LVM, create a Volume Group (VG). As shown in Example 3-29, we create a VG that is named `dockervg` with the new disks.

Example 3-29 Creating a VG called dockervg

```
itsoslec:~ # vgcreate dockervg /dev/dasde1
Volume group "dockervg" successfully created
```

4. Create a Logical Volume (LV). As shown in Example 3-30, we ran the `lvcreate` command to create the file system by using all of the space that is available in `dockervg` volume group.

Example 3-30 Creating the Logical Volume called var_lib_docker

```
itsoslec:~ # lvcreate -l 100%free -n var_lib_docker dockervg
Logical volume "var_lib_docker" created.
```

5. Format your new logical volume as a BTRFS file system. The command and its output is shown in Example 3-31.

Example 3-31 Formatting Logical Volume as BTRFS

```
itsoslec:~ # mkfs.btrfs -f /dev/dockervg/var_lib_docker
btrfs-progs v4.5.3+20160729
See http://btrfs.wiki.kernel.org for more information.
```

Detected an SSD, turning off metadata duplication. Mkfs with -m dup if you want to force metadata duplication.

```
Label:                (null)
UUID:                 c85c3664-ad09-40aa-9a04-7f99dbfa815a
Node size:            16384
Sector size:          4096
Filesystem size:     10.31GiB
Block group profiles:
  Data:               single           8.00MiB
  Metadata:           single           8.00MiB
  System:             single           4.00MiB
SSD detected:         yes
Incompat features:    extref, skinny-metadata
Number of devices:    1
Devices:
  ID      SIZE  PATH
  1      10.31GiB /dev/dockervg/var_lib_docker
```

6. Update the /etc/fstab file so that file system is persistent during server restarts. Append the lines that are shown in Example 3-32 into your file.

Example 3-32 Entries for FSTAB

```
# Docker file system
itsoslec:~ # echo "/dev/dockervg/var_lib_docker /var/lib/docker btrfs defaults 0
0" >> /etc/fstab
```

7. Back up /var/lib/docker first by using the command that is shown in Example 3-33.

Example 3-33 Backing up of /var/lib/docker

```
cp -a /var/lib/docker /var/lib/docker.bkp
```

Mount the file system by using the **mount -a** command. To confirm that the file system is mounted, use the **df -h** command to check whether file system is mounted, as shown in Example 3-34.

Example 3-34 Verifying whether /var/lib/docker is mounted

```
itsoslec:~ # mount -a
itsoslec:~ # df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
devtmpfs	7.9G	0	7.9G	0%	/dev
tmpfs	7.9G	0	7.9G	0%	/dev/shm
tmpfs	7.9G	9.7M	7.9G	1%	/run
tmpfs	7.9G	0	7.9G	0%	/sys/fs/cgroup
/dev/dasdb3	19G	6.3G	12G	36%	/
/dev/dasdb1	194M	24M	160M	13%	/boot/zipl
/dev/dasdb3	19G	6.3G	12G	36%	/home
/dev/dasdb3	19G	6.3G	12G	36%	/var/opt
/dev/dasdb3	19G	6.3G	12G	36%	/tmp

/dev/dasdb3	19G	6.3G	12G	36%	/var/tmp
/dev/dasdb3	19G	6.3G	12G	36%	/boot/grub2/s390x-emu
/dev/dasdb3	19G	6.3G	12G	36%	/var/spool
/dev/dasdb3	19G	6.3G	12G	36%	/opt
/dev/dasdb3	19G	6.3G	12G	36%	/var/crash
/dev/dasdb3	19G	6.3G	12G	36%	/var/lib/mailman
/dev/dasdb3	19G	6.3G	12G	36%	/.snapshots
/dev/dasdb3	19G	6.3G	12G	36%	/var/lib/pgsql
/dev/dasdb3	19G	6.3G	12G	36%	/var/log
/dev/dasdb3	19G	6.3G	12G	36%	/usr/local
/dev/dasdb3	19G	6.3G	12G	36%	/var/lib/named
/dev/mapper/vg_data-lv_data	21G	2.8G	18G	14%	/srv
/dev/loop0	2.8G	2.8G	0	100%	
/srv/ftp/SLE-12-Server-DVD-s390x-GM-DVD1					
tmpfs	1.6G	0	1.6G	0%	/run/user/0
/dev/mapper/dockervg-var_lib_docker	11G	17M	11G	1%	/var/lib/docker

3.3 Docker verification

After the prerequisites and software are installed, it is important to ensure that all components are installed as planned.

Verify the installed version by running the commands that are shown in Example 3-35.

Example 3-35 Checking Docker version

```
itsoslec:~ # docker -v
Docker version 17.06.2-ee-16, build 9ef4f0a
itsoslec:~ #
```

Where:

- ▶ Engine: YY.MM.CC-VV
- ▶ YY = Year
- ▶ MM = Month
- ▶ CC = control
- ▶ VV = Docker Edition

To allow non-privileged users to perform administrative tasks on the Docker host server without prepending *sudo* for each command, add the user ID into the Docker group that is named “docker”, which is automatically created during the Docker installation process.

To include users into the group, run the command as root ID that is shown in Example 3-36.

Example 3-36 Adding a docker group to a Linux non-privileged user ID

```
usermod -aG docker <USERNAME>
```

Where <USERNAME> the name of Linux User ID. Request that the user logout and log back in to ensure that the user ID is updated in the group.

The output and confirmation that user creates with the correct group is shown in Figure 3-20.

```
itsoslec:~ # usermod -aG docker dockeradmin
itsoslec:~ # id dockeradmin
uid=1001(dockeradmin) gid=100(users) groups=482(docker),100(users)
itsoslea:~ #
```

Figure 3-20 Adding a Docker group to a Linux non-privileged user ID

Note: As of this writing, the BTRFS file system is supported by SUSE Linux only. Skip this step if you are running Red Hat Enterprise Linux.

Configure Docker EE to use the BTRFS file system. This step is required only if the file system is not using BTRFS. However, specifically specifying the storage driver has no negative effects.

Edit the `/etc/docker/daemon.json` file (create it if it does not exist) and add the contents that are shown in Figure 3-21.

```
{
  "storage-driver": "btrfs"
}
```

Figure 3-21 Updating storage driver in `/etc/docker/daemon.json` file

The Docker service can now be started by using the `systemctl start docker` command, as seen in Example 3-37.

Example 3-37 Starting Docker Service

```
systemctl start docker
```

The output of the command is shown in Figure 3-22.

```
itsoslec:~ # systemctl start docker
itsoslec:~ # systemctl status docker
â-? docker.service - Docker Application Container Engine
  Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor
  preset: disabled)
  Active: active (running) since Fri 2018-10-05 09:39:59 EDT; 14s ago
  Docs: https://docs.docker.com
  Process: 4112 ExecStartPost=/bin/sh -c awk -v dir="$(docker info -f
  '{{.DockerRootDir}}/{{.Driver}})' '$5 == dir && $7 !~ "shared:" {system("mount
  --make-rshared " dir)}' /proc/self/mountinfo (code=exited, status=0/SUCCESS)
  Main PID: 3968 (dockerd)
  Tasks: 36
  Memory: 68.6M
  CPU: 1.332s
  CGroup: /system.slice/docker.service
         â"â"3968 /usr/bin/dockerd
         â"â"3974 docker-containerd -l
  unix:///var/run/docker/libcontainerd/docker-containerd.sock
  --metrics-interval=0 --start-timeout 2...
         â"â"4206 docker-containerd-shim
  4d42e71d349059a3ec5b4660a7f26bfe41737059315fb4b9dfb6a5b25984af08
  /var/run/docker/libcontainerd/4...
  â"â"4d42e71d349059a3ec5b4660a7f26bfe41737059315fb4b9dfb6a5b25984af08
         â"â"4222 /telemetry
  Oct 05 09:39:58 itsoslea dockerd[3968]:
  time="2018-10-05T09:39:58.959738165-04:00" level=warning msg="Your kernel does
  not support... limit"
  Oct 05 09:39:58 itsoslea dockerd[3968]:
  time="2018-10-05T09:39:58.959758933-04:00" level=warning msg="Your kernel does
  not support... limit"
  Oct 05 09:39:58 itsoslea dockerd[3968]:
  time="2018-10-05T09:39:58.960050578-04:00" level=info msg="Loading containers:
  start."
  Oct 05 09:39:59 itsoslea dockerd[3968]:
```

Figure 3-22 Output of Docker status

To load Docker service during server initialization, run the command that is shown in Example 3-38.

Example 3-38 Enabling Docker service to be loaded during server initialization

```
systemctl enable docker
```

The output of the command is shown Figure 3-23.

```
itsoslec:~ # systemctl enable docker
Created symlink from /etc/systemd/system/multi-user.target.wants/docker.service
to /usr/lib/systemd/system/docker.service.
itsoslec:~ #
```

Figure 3-23 Output of Docker enabled

The easiest way to verify whether you successfully installed and started Docker is by using the **docker container run hello-world** command, which downloads a test image from the Docker hub repository, runs it in a container, and prints a “Hello from Docker” message and exit (see Example 3-39).

Example 3-39 Running the hello-world Docker container

```
docker container run hello-world
```

The output resembles the output that is shown in Figure 3-24.

```
dockeradmin@itsoslec:~> docker container run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
f9c6e5e7797: Pull complete
Digest: sha256:0add3ace90ecb4adbf7777e9aacf18357296e799f81cab9fde470971e499788
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.
    (s390x)
 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 ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

dockeradmin@itsoslec:~>
```

Figure 3-24 Output of docker container run command

Docker EE is now installed and running. Use `sudo` to run Docker commands. Continue to the Linux postinstallation process to configure the graph storage driver, and allow non-privileged users to run Docker commands and for other optional configuration steps.

3.4 Using hardware crypto for application containers

In this section, we describe how to set up containers to use IBM Z’s cryptographic hardware. Also described is how to enable containers to use the cryptographic hardware of IBM Z.

Finally, we use OpenSSL to demonstrate the encryption of data in use. For more information about IBM Z security and encryption, see 1.4, “Security” on page 13.

For the purpose of this demonstration, we completed the following tasks:

- ▶ Enabled cryptocard in Docker host
- ▶ Created a Red Hat crypto-container
- ▶ Tested the hardware crypto functions and Crypto Express adapter support for OpenSSL

Test environment

For our test environment, two Crypto Express cards were enabled and configured for our z14 LPAR. However, the ITSOZVM1 LPAR uses only one crypto adapter that is configured as accelerator.

Also, we defined and installed a Linux server as guest virtual machine to be our Docker host on a z/VM v7.1 LPAR. The Docker EE is running inside this Linux server to provide an infrastructure to run our containers. For more information about our test environment, see Table 3-4 and Figure 3-25.

Table 3-4 Crypto test environment

Component	Version
Docker Host	SuSE Linux Enterprise Server 12 SP3
Docker Container	Red Hat 7.5
z/VM	7.1 (updated with the latest patchset release)

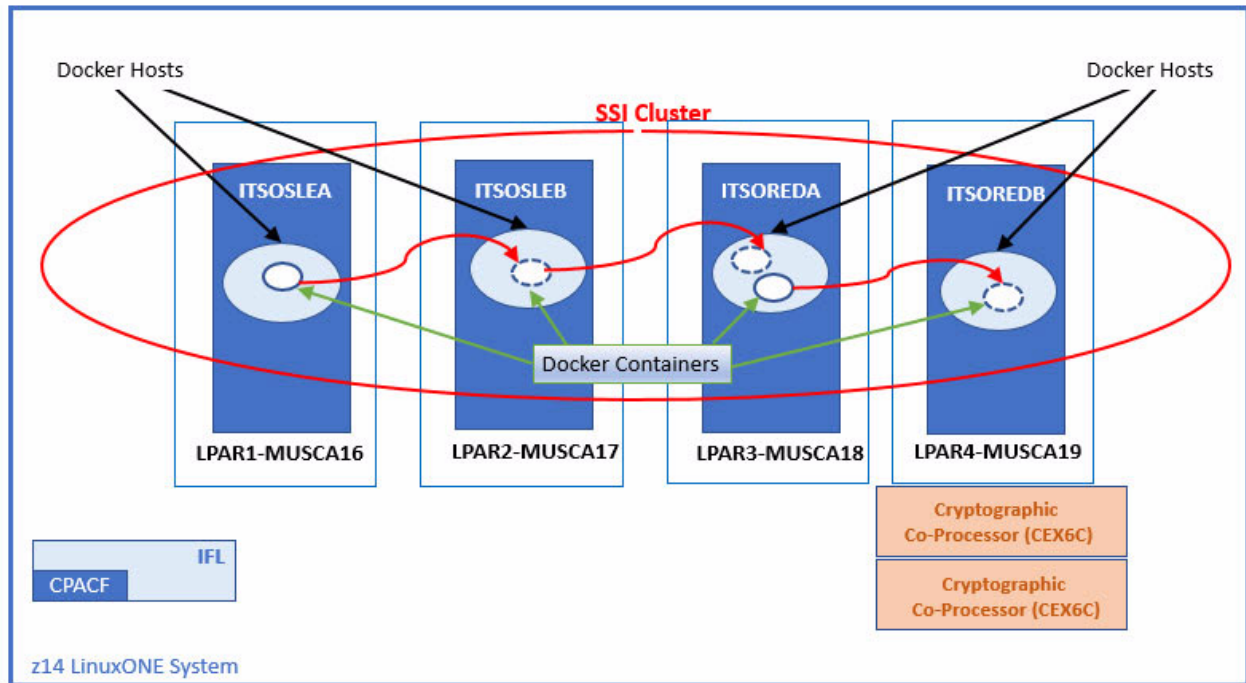


Figure 3-25 Test environment

Enabling Crypto Express in Docker host

Complete the following steps to enable Crypto Express in the Docker host:

1. Access the Crypto Express or accelerators to define the crypto settings in your Docker host. If these definitions are not made, you see the error that is shown in Figure 3-26 on page 66. This error indicates that your Docker host is not configured with Crypto Express.

```

itsoslec:~# docker run --name rhcryptocontainer --device
/dev/prandom:/dev/prandom --device /dev/z90crypt:/dev/z90crypt -d -it
registry.access.redhat.com/rhel7/rhel /bin/bash
4980b0f6e0b7ad7a739428a0efdc0e0ec3537c47827e094c73a27b16c18b6c6f
docker: Error response from daemon: linux runtime spec devices: error gathering
device information while adding custom device "/dev/z90crypt": lstat
/dev/z90crypt: no such file or directory.

```

Figure 3-26 Error message when z90crypt is not active

2. Determine whether your IBM Z LPAR has Crypto Express functions enabled. To make this determination, use HMC or CE¹ to check whether “CP Assist for Crypto Functions” is enabled. You also can access the z/VM LPAR by using x3270 terminal and run the commands as a privileged VM user that are shown in Figure 3-27.

```

q crypto
Crypto Adjunct Processor Instructions are installed
Ready; T=0.01/0.01 09:44:45

q crypto domain
AP 000 CEX6C Domain 013 available shared shared
Ready; T=0.01/0.01 09:45:06

```

Figure 3-27 Verifying the availability of crypto queues

The output of the command that is shown in Figure 3-27 indicates that the z/VM System can access one cryptographic coprocessor and it includes domain 13, which is assigned to handle the requests that come from the logical partition where the z/VM instance is stored.

3. Update the z/VM user ID directory by adding the CRYPTO APVIRT statement to allow the Docker host to access the cryptocard (see Example 3-40).

Example 3-40 z/VM User Directory for ITSOSLEC server

```

USER ITSOSLEC LNX4ITS0 16G 16G BG
  INCLUDE IBMDFLT
  IPL CMS
  MACHINE ESA 4
  OPTION CHPIDV ONE
  CPU 00 BASE
  CPU 01
  CRYPTO APVIRT
  NICDEF 0600 TYPE QDIO LAN SYSTEM VSWITCH1
  NICDEF 0620 TYPE QDIO LAN SYSTEM VSWITCH2
  MDISK 191 3390 0521 0010 IV1LX0 MR READ WRITE MULTI
  MDISK 0500 3390 30051 15025 IVMLX6 MR READ WRITE MULTI
  LINK LNXDISKS 984B 0100 MR
  LINK LNXDISKS 98CB 0200 MR

```

The cryptocard is now enabled for our z/VM LPAR and Linux guest.

¹ Check CPC Details/Instance Information on SE or HMC. The feature is free, but must be installed.

4. Install libica-tools, and enable and start z90crypt driver to the Docker host.

The use of the command that is shown in Example 3-41 installs the libica-tools. This package contains utilities that work with the IBM Z cryptocard.

Example 3-41 Installing libica-tools package using Zypper

```
itsoslec:~ # zypper install libica-tools
Retrieving repository 'SLE-Module-Web-Scripting12-Updates' metadata
..... [done]
Building repository 'SLE-Module-Web-Scripting12-Updates' cache
..... [done]
Retrieving repository 'SLE-SDK12-SP3-Updates' metadata
..... [done]
Building repository 'SLE-SDK12-SP3-Updates' cache
..... [done]
Retrieving repository 'SLES12-SP3-Updates' metadata
..... [done]
Building repository 'SLES12-SP3-Updates'
cache..... [done]
Loading repository data...
Reading installed packages...
Resolving package dependencies...
```

The following 2 NEW packages are going to be installed:

libica3 libica-tools

2 new packages to install.

Overall download size: 85.5 KiB. Already cached: 0 B. After the operation, additional 217.8 KiB will be used.

Continue? [y/n/...? shows all options] (y): y

```
Retrieving package libica3-3.0.2-3.6.s390x
(1/2), 58.2 KiB (146.5 KiB unpacked)
Retrieving: libica3-3.0.2-3.6.s390x.rpm
..... [done]
```

```
Retrieving package libica-tools-3.0.2-3.6.s390x
(2/2), 27.3 KiB (71.3 KiB unpacked)
Retrieving: libica-tools-3.0.2-3.6.s390x.rpm
..... [done]
```

```
Checking for file conflicts:
..... [done]
```

```
(1/2) Installing: libica3-3.0.2-3.6.s390x
..... [done]
```

```
(2/2) Installing: libica-tools-3.0.2-3.6.s390x
..... [done]
```

The z90crypt service can be enabled by running the commands that are shown in Example 3-42.

Example 3-42 Enabling z90crypt service

```
#Enabling z90crypt to start on boot time
itsoslec:~ # systemctl enable boot.z90crypt
boot.z90crypt.service is not a native service, redirecting to systemd-sysv-install
Executing /usr/lib/systemd/systemd-sysv-install enable boot.z90crypt
itsoslec:~ #
```

```

#Starting z90crypt
itsoslec:~ # systemctl start z90crypt
itsoslec:~ #

#Checking the status for the z90crypt service
itsoslec:~ # systemctl status z90crypt
â-? z90crypt.service - LSB: Load the z90crypt module
   Loaded: loaded (/etc/init.d/boot.z90crypt; bad; vendor preset: disabled)
   Active: active (exited) since Tue 2018-10-09 14:24:24 EDT; 20s ago
     Docs: man:systemd-sysv-generator(8)
   Process: 56536 ExecStart=/etc/init.d/boot.z90crypt start (code=exited,
status=0/SUCCESS)

Oct 09 14:24:24 itsoslec systemd[1]: Starting LSB: Load the z90crypt module...
Oct 09 14:24:24 itsoslec boot.z90crypt[56536]: Loading z90crypt module..done
Oct 09 14:24:24 itsoslec systemd[1]: Started LSB: Load the z90crypt module.
itsoslec:~ #

```

Note: The user directory statement CRYPTO APVIRT provides access to the cryptographic hardware and allows the z90crypt device driver to use cryptographic instructions.

Creating a Red Hat crypto container

The command that is shown in Figure 3-28 creates a Red Hat 7 container and the **-d** option is used to keep the container running. Also, the first process the container runs is `/bin/bash`.

```

itsoslec:/docker_config # docker run --name rhcryptocontainer --device
/dev/prandom:/dev/prandom --device /dev/z90crypt:/dev/z90crypt -d -it
registry.access.redhat.com/rhel7/rhel /bin/bash
4980b0f6e0b7ad7a739428a0efdc0e0ec3537c47827e094c73a27b16c18b6c6f

```

Figure 3-28 Creating a Red Hat 7 container

Some required packages are not installed by default in the Docker images. You must install the following packages that are required for encryption, including hardware Cryptographic Support:

- ▶ openssl
- ▶ openssl098e
- ▶ openssl-libs
- ▶ openssl-ibmca

By now, the container and all required packages are installed. The next step is to update the OpenSSL configuration file to use the `ibmca` engine and allow your container to use the IBM Z cryptocard hardware functions.

Note: For more information about how to set up cryptocard, see *Security and Linux on IBM Z*, [REDP-5464](#).

Complete the following steps:

1. Locate the OpenSSL configuration file. In our Red Hat 7.4 distribution, this file is in the following subdirectory:
 - `/etc/pki/tls`

2. Back up the file, as shown in Example 3-43.

Example 3-43 Backing up openssl.cnf file

```
[root@4980b0f6e0b7 /]# cd /etc/pki/tls/

[root@4980b0f6e0b7 tls]# ls
cert.pem certs misc openssl.cnf private

[root@4980b0f6e0b7 tls]# ls -la /etc/pki/tls/openssl.cnf
-rwxr-xr-x 1 1001 1000 10923 Dec 13 2017 /etc/pki/tls/openssl.cnf

[root@4980b0f6e0b7 tls]# cp -p /etc/pki/tls/openssl.cnf
/etc/pki/tls/openssl.cnf.backup

[root@4980b0f6e0b7 tls]# ls -al /etc/pki/tls/openssl.cnf*
-rwxr-xr-x 1 1001 1000 10923 Dec 13 2017 /etc/pki/tls/openssl.cnf
-rwxr-xr-x 1 1001 1000 10923 Dec 13 2017 /etc/pki/tls/openssl.cnf.backup
```

3. Some ibmca-related content must be appended to the openssl.cnf file. This code is included with the openssl-ibmca package. Therefore, find the ibmca package and look for a file that is named openssl.cnf.sample.s390x (see Example 3-44).

Example 3-44 Locating the ibmca sample file

```
[root@4980b0f6e0b7 tls]# rpm -ql openssl-ibmca |grep openssl.cnf.sample.s390x
/usr/share/doc/openssl-ibmca-1.4.0/openssl.cnf.sample.s390x

[root@4980b0f6e0b7 /]# ls -la
/usr/share/doc/openssl-ibmca-1.4.0/openssl.cnf.sample.s390x
-rw-r--r-- 1 root root 1396 Jun 26 11:49
/usr/share/doc/openssl-ibmca-1.4.0/openssl.cnf.sample.s390x
```

4. Append the ibmca-related configuration lines to the OpenSSL configuration file, as shown in Example 3-45.

Example 3-45 Append ibmca specific content to the OpenSSL config file

```
[root@4980b0f6e0b7 /]# tee -a /etc/pki/tls/openssl.cnf <
/usr/share/doc/openssl-ibmca-1.4.0/openssl.cnf.sample.s390x
```

5. Ensure that the ibmca section was appended at the end of the OpenSSL configuration file, as shown in Example 3-46.

Example 3-46 Verifying the append

```
[root@4980b0f6e0b7 /]# grep -n ibmca_section /etc/pki/tls/openssl.cnf
370:ibmca = ibmca_section
373:[ibmca_section]
```

6. The reference to the `ibmca` section in the OpenSSL configuration file must be inserted. Therefore, we insert the line: `openssl_conf = openssl_def`, as shown in Figure 3-47.

Example 3-47 Insert ibmca reference (partial file shown)

```
[root@4980b0f6e0b7 ~]# head /etc/pki/tls/openssl.cnf
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

# This definition stops the following lines choking if HOME isn't
# defined.
HOME      = .
RANDFILE= $ENV::HOME/.rnd

[root@4980b0f6e0b7 ~]# head -n 15 /etc/pki/tls/openssl.cnf
#
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
#

# This definition stops the following lines choking if HOME isn't
# defined.
HOME      = .
RANDFILE= $ENV::HOME/.rnd

openssl_conf = openssl_def #<== line inserted

# Extra OBJECT IDENTIFIER info:
#oid_file= $ENV::HOME/.oid
oid_section= new_oids
```

7. Check the value of the dynamic path variable and adjust it, if needed (see Example 3-48).

Example 3-48 Checking the dynamic path variable

```
[root@4980b0f6e0b7 ~]# grep dynamic.path /etc/pki/tls/openssl.cnf
# Set the dynamic_path to where the libibmca.so engine
dynamic_path = /usr/lib64/openssl/engines/libibmca.so

[root@4980b0f6e0b7 ~]# ls -la /usr/lib64/openssl/engines/libibmca.so
-rwxr-xr-x 1 root root 54360 Jun 26 11:49 /usr/lib64/openssl/engines/libibmca.so
[root@4980b0f6e0b7 ~]#
```

The inserted `ibmca` section is shown in Example 3-49.

Example 3-49 `ibmca` section of the `openssl.cnf` file

```
# OpenSSL example configuration file. This file will load the IBMCA engine
# for all operations that the IBMCA engine implements for all apps that
# have OpenSSL config support compiled into them.
#
# Adding OpenSSL config support is as simple as adding the following line to
# the app:
#
# #define OPENSSL_LOAD_CONF1
#
openssl_conf = openssl_def

[openssl_def]
engines = engine_section

[engine_section]
ibmca = ibmca_section

[ibmca_section]

# The openssl engine path for libibmca.so.
# Set the dynamic_path to where the libibmca.so engine
# resides on the system.
dynamic_path = /usr/lib64/openssl/engines/libibmca.so
engine_id = ibmca
init = 1

#
# The following ibmca algorithms will be enabled by these parameters
# to the default_algorithms line. Any combination of these is valid,
# with "ALL" denoting the same as all of them in a comma separated
# list.
#
# RSA
# - RSA encrypt, decrypt, sign and verify, key lengths 512-4096
#
# RAND
# - Hardware random number generation
#
# CIPHERS
# - DES-ECB, DES-CBC, DES-CFB, DES-OFB, DES-EDE3, DES-EDE3-CBC, DES-EDE3-CFB,
#   DES-EDE3-OFB, AES-128-ECB, AES-128-CBC, AES-128-CFB, AES-128-OFB,
#   AES-192-ECB, AES-192-CBC, AES-192-CFB, AES-192-OFB, AES-256-ECB,
#   AES-256-CBC, AES-256-CFB, AES-256-OFB symmetric crypto
#
# DIGESTS
# - SHA1, SHA256, SHA512 digests
#
default_algorithms = ALL
#default_algorithms = RAND,RSA,CIPHERS,DIGESTS
```

Test Hardware Crypto functions

With the requirements met and all customizations of OpenSSL completed, we now can use the IBM Z hardware cryptographic functions.

Before starting, review whether the dynamic engine loading support is enabled by default and the engine `ibmca` is available and used in our installation, as shown in Example 3-50.

Example 3-50 Check if dynamic engine loading support is enabled and `ibmca` is used

```
[root@4980b0f6e0b7 /]# openssl engine
(dynamic) Dynamic engine loading support
(ibmca) Ibmca hardware engine support
```

Check which algorithm are supported by `ibmca`, as shown in Example 3-51.

Example 3-51 Supported algorithms

```
[root@4980b0f6e0b7 /]# openssl engine -c
(dynamic) Dynamic engine loading support
(ibmca) Ibmca hardware engine support
[RSA, DSA, DH, RAND, DES-ECB, DES-CBC, DES-OFB, DES-CFB, DES-EDE3, DES-EDE3-CBC,
DES-EDE3-OFB, DES-EDE3-CFB, AES-128-ECB, AES-192-ECB, AES-256-ECB, AES-128-CBC,
AES-192-CBC, AES-256-CBC, AES-128-OFB, AES-192-OFB, AES-256-OFB, AES-128-CFB,
AES-192-CFB, AES-256-CFB, id-aes128-GCM, id-aes192-GCM, id-aes256-GCM, SHA1,
SHA256, SHA512]
```

Crypto Express6S card support for OpenSSL

We now check the number of run requests in the cryptographic adapter for your container. A change of this counter is observed if RSA requests that the cryptographic adapter is run. First, note the number of the counter before the test is run (see Example 3-52).

Example 3-52 Displaying the request counter of the Crypto Express6S card

```
[root@4980b0f6e0b7 /]# cat sys/devices/ap/card01/request_count
34205
```

Next, we perform crypto operations that use the cryptographic adapter (especially RSA), as shown in Example 3-53.

Example 3-53 Testing RSA requests

```
[root@4980b0f6e0b7 /]# openssl speed rsa2048 -elapsed
You have chosen to measure elapsed time instead of user CPU time.
Doing 2048 bit private rsa's for 10s: 4588 2048 bit private RSA's in 10.00s
Doing 2048 bit public rsa's for 10s: 6660 2048 bit public RSA's in 10.00s
OpenSSL 1.0.2k-fips 26 Jan 2017
built on: reproducible build, date unspecified
options:bn(64,64) md2(int) rc4(8x,char) des(idx,cisc,16,int) aes(partial) idea(int)
blowfish(idx)
compiler: gcc -I. -I.. -I../include -fPIC -DOPENSSL_PIC -DZLIB -DOPENSSL_THREADS
-D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DKRB5_MIT -m64 -DB_ENDIAN -Wall -O2 -g -pipe -Wall
-Wp,-D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector-strong --param=ssp-buffer-size=4
-grecord-gcc-switches -m64 -march=z196 -mtune=zEC12 -Wa,--noexecstack -DPURIFY
-DOPENSSL_BN_ASM_MONT -DOPENSSL_BN_ASM_GF2m -DRC4_ASM -DSHA1_ASM -DSHA256_ASM -DSHA512_ASM
-DAES_ASM -DAES_CTR_ASM -DAES_XTS_ASM -DGHASH_ASM
          sign    verify    sign/s  verify/s
rsa 2048 bits 0.002180s 0.001502s    458.8    666.0
```

We can now display the number of requests that were run by the Crypto Express card. As shown in Example 3-54, you see that the number is much higher than before, which indicates that the Crypto Express feature was used.

Example 3-54 Display the request counter of the Crypto Express6S card

```
[root@4980b0f6e0b7 /]# cat sys/devices/ap/card01/request_count
68356
```

Another way to verify if Cryptocard processing requests is to run the command that is shown in Example 3-55.

Example 3-55 icastats command

function	hardware			software		
	ENC	CRYPT	DEC	ENC	CRYPT	DEC
RSA-ME		19988			0	
RSA-CRT		14163			0	

While the openssl was running in our tests, we started a new SSH session against the Docker host and ran the `docker stats rhcryptocontainer` command to display a live stream of our crypto container to get resource usage statistics.

As shown in Example 3-56, the CPU time of the Docker host was not affected by the crypto work. Therefore, by using this IBM Z feature, cryptographic cipher calculations can be offloaded from the central processor, which considerably reduces the processor cycles of such operations compared to the cost of having them done through software emulation.

Example 3-56 Output of docker status command

CONTAINER	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O
rhcryptocontainer	0.98%	932KiB / 15.63GiB	0.01%	36MB /
333kB	149MB / 71.3MB	2		

Docker containers can benefit from the use of IBM Z cryptographic hardware and features to protect data without changing or adjusting applications. Also, it supports the use of CPACF and Crypto Express6S (the latest available at the time of this writing) by using in-kernel crypto APIs and the libica cryptographic functions library.

The CPACF encrypts, decrypts, hashes, and supports message authentication and random number generation to protect data in your containers. CPACF on IBM z14 is improved to support pervasive encryption, which provides an envelope of protection around the data that is on IBM Z servers and containers.

The command that is shown in Example 3-57 shows how to determine whether the CPACF feature is enabled on your hardware.

Example 3-57 Verifying if CPACF is enabled on your hardware

```
[root@4980b0f6e0b7 /]# cat /proc/cpuinfo
vendor_id       : IBM/S390
# processors    : 2
bogomips per cpu: 18656.00
max thread id   : 0
features: esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs te vx vxd vxe
sie
facilities      : 0 1 2 3 4 6 7 8 9 10 12 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 30 31 32 33 34 35 36 37 38 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55
57 58 59 60 73 74 75 76 77 80 81 82 128 129 130 131 133 134 135 146 147 168 1024
1025 1026 1027 1028 1030 1031 1032 1033 1034 1036 1038 1039 1040 1041 1042 1043
1044 1045 1046 1047 1048 1049 1050 1051 1052 1054 1055 1056 1057 1058 1059 1060
1061 1062 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077
1078 1079 1081 1082 1083 1084 1097 1098 1099 1100 1101 1104 1105 1152 1153 1154
1155 1157 1158 1159 1170 1171 1192
cache0         : level=1 type=Data scope=Private size=128K line_size=256
associativity=8
cache1         : level=1 type=Instruction scope=Private size=128K line_size=256
associativity=8
cache2         : level=2 type=Data scope=Private size=4096K line_size=256
associativity=8
cache3         : level=2 type=Instruction scope=Private size=2048K line_size=256
associativity=8
cache4         : level=3 type=Unified scope=Shared size=131072K line_size=256
associativity=32
cache5         : level=4 type=Unified scope=Shared size=688128K line_size=256
associativity=42
processor 0: version = FF, identification = 167A88, machine = 3907
processor 1: version = FF, identification = 167A88, machine = 3907
```

For the tests with CPACF, it is necessary to install a libica package that contains some utilities to check on the CPACF feature code enablement.

The `icainfo` command displays which CPACF functions are supported by the implementation inside the `libica` library. The command that is shown in Example 3-58 helps to determine which `libica` functions are available for your Linux server or container.

Example 3-58 Show available `icainfo` functions

```
[root@4980b0f6e0b7 /]# icainfo
Cryptographic algorithm support
-----
```

function	hardware	software
SHA-1	yes	yes
SHA-224	yes	yes
SHA-256	yes	yes
SHA-384	yes	yes
SHA-512	yes	yes
SHA3-224	yes	no
SHA3-256	yes	no
SHA3-384	yes	no
SHA3-512	yes	no
SHAKE-128	yes	no
SHAKE-256	yes	no
GHASH	yes	no
P_RNG	yes	yes
DRBG-SHA-512	yes	yes
RSA ME	yes	yes
RSA CRT	yes	yes
DES ECB	yes	yes
DES CBC	yes	yes
DES OFB	yes	no
DES CFB	yes	no
DES CTR	yes	no
DES CMAC	yes	no
3DES ECB	yes	yes
3DES CBC	yes	yes
3DES OFB	yes	no
3DES CFB	yes	no
3DES CTR	yes	no
3DES CMAC	yes	no
AES ECB	yes	yes
AES CBC	yes	yes
AES OFB	yes	no
AES CFB	yes	no
AES CTR	yes	no
AES CMAC	yes	no
AES XTS	yes	no
AES GCM	yes	no

```
-----
Built-in FIPS support: FIPS mode inactive.
```

No specific z/VM authorization or configuration or definitions are required to access CPACF functions from z/VM Random data Cryptographic adapters or accelerators.

To perform some tests, we use OpenSSL tool to operate the CPACF functions on containers in our z14. First, we reset the counter by using the `icastats -r` command. Then, we start the tests. The output and results are shown in Example 3-59.

Example 3-59 Performing tests with CPACF

```
[root@4980b0f6e0b7 /]# icastats -r
[root@4980b0f6e0b7 /]# openssl speed -evp des-ede3-cbc 2>/dev/null | tail -n 3
The 'numbers' are in 1000s of bytes per second processed.
type           16 bytes    64 bytes    256 bytes   1024 bytes   8192 bytes
des-ede3-cbc   94569.01k   345241.06k  561417.25k  669672.11k  709766.94k

[root@4980b0f6e0b7 /]# openssl speed -evp aes-128-cbc 2>/dev/null | tail -n 3
The 'numbers' are in 1000s of bytes per second processed.
type           16 bytes    64 bytes    256 bytes   1024 bytes   8192 bytes
aes-128-cbc    92626.60k   361765.61k  1611765.50k 3215062.73k 4255442.62k

[root@4980b0f6e0b7 /]# openssl speed -evp aes-192-cbc 2>/dev/null | tail -n 3
The 'numbers' are in 1000s of bytes per second processed.
type           16 bytes    64 bytes    256 bytes   1024 bytes   8192 bytes
aes-192-cbc    91298.74k   356671.25k  1483886.51k 2811274.87k 3678246.36k

[root@4980b0f6e0b7 /]# openssl speed -evp aes-256-cbc 2>/dev/null | tail -n 3
The 'numbers' are in 1000s of bytes per second processed.
type           16 bytes    64 bytes    256 bytes   1024 bytes   8192 bytes
aes-256-cbc    89934.02k   351470.49k  1393629.75k 2502938.62k 3195490.98k

[root@4980b0f6e0b7 /]# icastats
function      |             hardware             |             software             |
-----+-----+-----+-----+-----+-----+-----+-----+
              |             ENC  CRYPT  DEC  |             ENC  CRYPT  DEC  |
-----+-----+-----+-----+-----+-----+-----+
          SHA-1 |             484          |             0          |
          SHA-224 |             4          |             0          |
          SHA-256 |             4          |             0          |
          SHA-384 |             4          |             0          |
          SHA-512 |             4          |             0          |
        SHA3-224 |             0          |             0          |
        SHA3-256 |             0          |             0          |
        SHA3-384 |             0          |             0          |
        SHA3-512 |             0          |             0          |
        SHAKE-128 |             0          |             0          |
        SHAKE-256 |             0          |             0          |
           GHASH |            332          |             0          |
           P_RNG |             0          |             0          |
        DRBG-SHA-512 |            676          |             0          |
           RSA-ME |             4          |             0          |
           RSA-CRT |             4          |             0          |
          DES ECB |             0          |             0          |             0          |
          DES CBC |             0          |             0          |             0          |
          DES OFB |             0          |             0          |             0          |
          DES CFB |             0          |             0          |             0          |
          DES CTR |             0          |             0          |             0          |
          DES CMAC |             0          |             0          |             0          |
         3DES ECB |             8          |             8          |             0          |
         3DES CBC |          42520920          |             68          |             0          |
```

3DES OFB	8	8	0	0
3DES CFB	8	8	0	0
3DES CTR	4	4	0	0
3DES CMAC	56	20	0	0
AES ECB	196	12	0	0
AES CBC	182705743	72	0	0
AES OFB	12	12	0	0
AES CFB	24	24	0	0
AES CTR	196	12	0	0
AES CMAC	384	60	0	0
AES XTS	8	8	0	0
AES GCM	120	156	0	0

Final considerations regarding crypto

The IBM Z Hardware provides enhance cryptographic features by using its cryptocard and specific processors. As a result, the Docker containers are enabled to run several crypto functions directly in the hardware, which eliminates bottlenecks that are caused by the software layers. This capability directly improves performance on your Docker environment that runs in a Linux on Z.

In this section, we describe how to use the cryptographic functions on a Linux container to benefit from encrypt data in-flight. We tested the use of IBM Z encryption on Docker containers by using OpenSSL tool to encrypt and decrypt data. During the tests, we did not verify a high usage of the main CPU processor. This characteristic is unique on IBM Z because it has dedicated crypto processors.

The tamper-sensing and tamper-responding Crypto Express features provide acceleration for high-performance cryptographic operations and support up to 85 domains. This specialized hardware performs AES, DES/TDES, RSA, Elliptic Curve (ECC), SHA-1, and SHA-2, and other cryptographic operations.

Note: The importance of encryption to application containers cannot be overstated. It is vital to business applications that depend on security to keep their data secure and protected. This feature must be taken in consideration when you plan or implement crypto in your production applications.

In terms of implementation (as described in this section), the settings and steps to activate the cryptocard devices are simple and well-documented.

The IBM Z platform offers cryptographic engines that provide high-speed cryptographic operations. The cryptographic engines that are described next are available to improve your application to take advantage of Z hardware.

CP Assist for Cryptographic Functions

CP Assist for Cryptographic Functions (CPACF) is a high-performance, low-latency co-processor that performs symmetric key encryption and calculates message digests (hashes) in hardware. Supported algorithms are AES, DES/TDES, SHA-1, SHA-2, and SHA-3. It is provided through a set of instructions that are available in the hardware on every processor unit.

This feature is set up and loaded by way of a z90crypt module in the Docker host. After the Docker host can access CPACF, no other configuration steps must be performed inside the containers.

Random data

To enable operations that involve random data for your container, you must provide the option `--device /dev/prandom:/dev/prandom` by using the **docker run** command. For more information, see Figure 3-28 on page 68.

Cryptographic adapter accelerators

Cryptographic adapter accelerators are functions that are provided through high-security, tamper-responding hardware security modules (HSMs).

To allow containers to access cryptographic adapter or accelerators, you must provide the `--device /dev/z90crypt:/dev/z90crypt` option when using the **Docker run** command. For more information, see Figure 3-28 on page 68.

3.5 Moving Docker hosts by using z/VM SSI feature

The z/VM package includes (at no extra expense) the live guest relocation (LGR) that allows moving Linux virtual servers without disruption to the business. This feature helps you to avoid planned outages. The z/VM systems are aware of each other and can take advantage of their combined resources. LGR enables clients to avoid loss of service because of planned outages by relocating guests from a system that requires maintenance to a system that remains active during the maintenance period.

From sysadmin perspective, this feature is widely used during the z/VM maintenance as SLID or even major upgrades. The containers run while the Linux hosts are moved to different z/VM LPARs. During the move process (see Figure 3-29 on page 79), the Linux hosts run continuity because they were not affected by the move.

```

ITSOVM3 : VMRELOCATE MOVE ITSOSLEB to ITSOZVM1
Relocation of ITSOSLEB from ITSOZVM3 to ITSOZVM1 started
User ITSOSLEB has been relocated from ITSOZVM3 to ITSOZVM1
Ready; T=0.01/0.01 09:46:01
ITSOVM3 : Q CPLEVEL
z/VM Version 6 Release 4.0, service level 1801 (64-bit)
Generated at 09/04/18 11:02:53 EDT
IPL at 10/07/18 10:53:30 EDT
Ready; T=0.01/0.01 09:46:38

```

```

-----
itsosleb:~ # vmcp q userid
ITSOSLEB AT ITSOZVM1
-----

```

Relocation results

```

ITSOVM1 : Q CPLEVEL
z/VM Version 7 Release 1.0, service level 1801 (64-bit)
Generated at 10/07/18 12:28:25 EDT
IPL at 10/07/18 12:53:54 EDT

```

```

-----
itsosleb:~ # vmcp q userid
ITSOSLEB AT ITSOZVM3
-----

```

Figure 3-29 z/VM relocate process

We used two different z/VM versions in our lab and moved the Linux hosts between them. We observed that z/VM LPAR ITSOZVM1 and ITSOZVM3 had different z/VM versions, which were obtained by using the z/VM command **Q CPLEVEL**. During the z/VM upgrade of the z/VM v6.4, the hosts move and the containers run without outages.

3.6 Setting up swarm mode

The examples that are described in this section show how to enable swarm mode. For more information about configuring swarm mode, see [this website](#).

In this environment, we use Linux servers `itsoslea` and `itsosleba` as master nodes.

Configuring the first master node

First, acquire the IP address of the Docker node that is to be used as a swarm master node. This node becomes the leader node; in this case, it is `itsoslea` (see Example 3-60). Acquire the IP address of the Docker node.

Example 3-60 Acquiring the IP address

```

itsoslea:~ # ifconfig
docker0  Link encap:Ethernet  HWaddr 02:42:F1:C0:1E:08
          inet addr:172.17.0.1  Bcast:0.0.0.0  Mask:255.255.0.0
          inet6 addr: fe80::42:f1ff:fec0:1e08/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0

```

```

TX packets:3 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:0 (0.0 b) TX bytes:258 (258.0 b)

eth0 Link encap:Ethernet HWaddr 02:00:01:00:00:05
inet addr:9.76.61.245 Bcast:9.76.61.255 Mask:255.255.255.0
inet6 addr: fe80::1ff:fe00:5/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:1922178 errors:0 dropped:0 overruns:0 frame:0
TX packets:239907 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:2827116305 (2696.1 Mb) TX bytes:17758710 (16.9 Mb)

lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:14 errors:0 dropped:0 overruns:0 frame:0
TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1
RX bytes:1176 (1.1 Kb) TX bytes:1176 (1.1 Kb)

```

The IP address 9.76.61.245 is highlighted in Example 3-60.

Next, use the address to advertise this node as a master node (see Example 3-61).

Example 3-61 Advertising the master node

```

itsoslea:~ # docker swarm init --advertise-addr 9.76.61.245
Swarm initialized: current node (8cteeof8bwra8nqg58grppd0x) is now a manager.

```

To add a worker to this swarm, run the following command:

```

docker swarm join --token
SWMTKN-1-4xfphuac7rgking548h09n6t6sivihp8veji5quwyd9i9oqsyt-0jxkbnywbiy5kfwgduidff
plf 9.76.61.245:2377

```

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

Configuring a second master node

To add a manager node, run the command on the master that was given when the first master node was set up to obtain the token (see Example 3-62).

Example 3-62 Adding the manager node

```

itsoslea:~ # docker swarm join-token manager
To add a manager to this swarm, run the following command:

```

```

docker swarm join --token
SWMTKN-1-4xfphuac7rgking548h09n6t6sivihp8veji5quwyd9i9oqsyt-clsic3pk4oxph1jcw1rcx
uep 9.76.61.245:2377

```

Add `itsosleb` as another manager node by running the output of the `join-token` command (see Example 3-63).

Example 3-63 Adding a second manager node

```
itsosleb:~ # docker swarm join --token
SWMTKN-1-4xfphuac7rgking548h09n6t6sivi hp8veji5quwyd9i9oqsyt-clsic3pk4oxph1jcw1rcx
uep 9.76.61.245:2377
This node joined a swarm as a manager.
```

Confirm that the two manager nodes were added. Because `itsoslea` was the first manager node, it is labeled as the Leader and the second manager node is labeled as Reachable (see Example 3-64).

Example 3-64 Listing the nodes

```
itsoslea:~ # docker node ls
ID                               HOSTNAME          STATUS          AVAILABILITY
MANAGER STATUS
8cteeof8bwra8nqg58grppd0x *    itsoslea          Ready          Active
Leader
u0b2azj6374dbdcksibdxtbvj      itsosleb          Ready          Active
Reachable

itsosleb:~ # docker node ls
ID                               HOSTNAME          STATUS          AVAILABILITY
MANAGER STATUS
8cteeof8bwra8nqg58grppd0x      itsoslea          Ready          Active
Leader
u0b2azj6374dbdcksibdxtbvj *    itsosleb          Ready          Active
Reachable
```

Adding worker nodes

Next, add worker nodes to the swarm. Linux guests `itsoslec` and `itsosleb` are designated as worker nodes for the swarm in this environment. Use the command for adding worker nodes that was given when the first master node was initialized (see Example 3-65).

Example 3-65 Adding worker nodes

```
itsoslec:~ # docker swarm join --token
SWMTKN-1-4xfphuac7rgking548h09n6t6sivi hp8veji5quwyd9i9oqsyt-0jxkbnwybiy5kfwgduidff
plf 9.76.61.245:2377
This node joined a swarm as a worker.
```

View the new swarm with `itsoslec` as a worker (see Example 3-66).

Example 3-66 Viewing the new swarm

```
itsoslea:~ # docker node ls
ID                               HOSTNAME          STATUS          AVAILABILITY
MANAGER STATUS
8cteeof8bwra8nqg58grppd0x *    itsoslea          Ready          Active
Leader
jbb9qtkmfc28k2vvsvhz6ynnu      itsoslec          Ready          Active
u0b2azj6374dbdcksibdxtbvj      itsosleb          Ready          Active
Reachable
```

Running the same command on a worker node (itsos1ec) returns an error because worker nodes do not run worker node commands (see Example 3-67).

Example 3-67 Error message

```
itsos1ec:~ # docker node ls
Error response from daemon: This node is not a swarm manager. Worker nodes can't
be used to view or modify cluster state. Please run this command on a manager node
or promote the current node to a manager.
```

3.7 Setting up Kubernetes

A Kubernetes environment can be set up by using several methods, such as building from source, or the use of an installer to bootstrap the Kubernetes servers. For more information about these methods, see [this website](#).

This book uses the information about how to set up Kubernetes on IBM Z and LinuxONE servers from IBM Techdoc *Installing Kubernetes on Linux on Z and LinuxONE Using the kubeadm Installation Toolkit*, [TD106419](#).

The steps that are described next are taken from [this page](#) of the Kubernetes website.

Installing the base Kubernetes on the master node

First, install kubeadm to bootstrap Kubernetes on the master node. When installing Kubernetes by using this method, kubeadm pulls the files from the GitHub repository, if they do not exist on the host (a Linux Z repository is required).

For Ubuntu hosts, this issue is not a problem because the correct repository is used. For Red Hat, the default is to install on x86-64. The associated .yaml file must be updated before the installation is done. Therefore, download the associated .yaml file and replace the string x86_64 with s390x.

Set up the repository, as shown in Example 3-68.

Example 3-68 Establishing the repository to install kubeadm, kubectl, and kubelet

```
[root@itsoredc ~]# cat <<EOF > /etc/yum.repos.d/kubernetes.repo
> [kubernetes]
> name=Kubernetes
> baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-s390x
> enabled=1
> gpgcheck=1
> repo_gpgcheck=1
> gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
> exclude=kube*
> EOF
```

```
[root@itsoredc ~]# cat /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-s390x
enabled=1
gpgcheck=1
```



```
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kube*
```

Disable SELinux by setting it to passive mode, as shown in Example 3-69.

Example 3-69 Set SELinux in permissive mode

```
[root@itsoredc ~]# setenforce 0
setenforce: SELinux is disabled
```

Now, install kubelet, kubeadm, and kubectl on the master node, as shown in Example 3-70.

Example 3-70 Installing kubelet, kubeadm, and kubectl

```
[root@itsoredc ~]# yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
Loaded plugins: langpacks, product-id, search-disabled-repos, subscription-manager
.
.
.
Installed:
  kubeadm.s390x 0:1.12.1-0                kubectl.s390x 0:1.12.1-0
  kubelet.s390x 0:1.12.1-0

Dependency Installed:
  cri-tools.s390x 0:1.12.0-0                kubernetes-cni.s390x 0:0.6.0-0
  socat.s390x 0:1.7.3.2-2.e17

Complete!
```

Certain ports are needed to be opened on the master node, as shown in Example 3-71.

Example 3-71 Open needed ports for kubeadm and kubernetes

```
[root@itsoredc ~]# firewall-cmd --zone=public --add-port=6443/tcp --permanent
success
[root@itsoredc ~]# firewall-cmd --zone=public --add-port=10250/tcp --permanent
success
[root@itsoredc ~]# firewall-cmd --reload
success
[root@itsoredc ~]# iptables-save | grep 6443
-A IN_public_allow -p tcp -m tcp --dport 6443 -m conntrack --ctstate NEW -j ACCEPT
[root@itsoredc ~]# iptables-save | grep 10250
-A IN_public_allow -p tcp -m tcp --dport 10250 -m conntrack --ctstate NEW -j
ACCEPT
```

Some users on RHEL/CentOS 7 reported issues with traffic being routed incorrectly because iptables was bypassed. Ensure that `net.bridge.bridge-nf-call-iptables` is set to 1 in your `sysctl` configuration file, as shown in Example 3-72.

Example 3-72 Update net.bridge.bridge-nf-call-iptables

```
root@itsoredc ~]# cat <<EOF > /etc/sysctl.d/k8s.conf
> net.bridge.bridge-nf-call-ip6tables = 1
> net.bridge.bridge-nf-call-iptables = 1
> EOF
[root@itsoredc ~]# sysctl --system
* Applying /usr/lib/sysctl.d/00-system.conf ...
net.ipv4.tcp_syncookies = 1
vm.dirty_ratio = 40
kernel.sched_min_granularity_ns = 10000000
kernel.sched_wakeup_granularity_ns = 15000000
kernel.sched_tunable_scaling = 0
kernel.sched_latency_ns = 80000000
kernel.shmmax = 4294967295
kernel.shmall = 268435456
* Applying /usr/lib/sysctl.d/10-default-yama-scope.conf ...
kernel.yama.ptrace_scope = 0
* Applying /usr/lib/sysctl.d/50-default.conf ...
kernel.sysrq = 16
kernel.core_uses_pid = 1
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.default.promote_secondaries = 1
net.ipv4.conf.all.promote_secondaries = 1
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
* Applying /etc/sysctl.d/99-sysctl.conf ...
net.bridge.bridge-nf-call-iptables = 1
* Applying /etc/sysctl.d/k8s.conf ...
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
* Applying /etc/sysctl.conf ...
net.bridge.bridge-nf-call-iptables = 1
[root@itsoredc ~]#
```

Swap must be disabled for Kubernetes to work properly. The easiest option is to install Linux without swap space; therefore, disable it for now, as shown in Example 3-73.

Example 3-73 Disabling swap

```
### Disabling swap, not supported for kubelet
[root@itsoredc ~]# swapon -a
```

Start kubelet, as shown in Example 3-74.

Example 3-74 Starting kubelet

```
[root@itsoredc ~]# systemctl enable kubelet && systemctl start kubelet
Created symlink from /etc/systemd/system/multi-user.target.wants/kubelet.service
to /etc/systemd/system/kubelet.service.
```

Next, get `k8s.gcr.io/kube-proxy-s390x:v1.12.2` and tag this `kube-proxy-s390x` image with `amd64` to trick Kubernetes into thinking `kube-proxy` is installed so that it does not download the `x86_64` version, as shown in Example 3-75.

Example 3-75 Downloading and tagging kube-proxy

```
[root@itsoredc kubernetes]#
[root@itsoredc kubernetes]# docker pull k8s.gcr.io/kube-proxy-s390x:v1.12.2
v1.12.1: Pulling from kube-proxy-s390x
52fb6aa6b72f: Pull complete
7087cdfd2c22: Pull complete
1fb073cf059c: Pull complete
Digest: sha256:b056eb9eef26bf8af3e236d5db783bdcea4faf1d6ef20c0dd6870a4ba556984c
Status: Downloaded newer image for k8s.gcr.io/kube-proxy-s390x:v1.12.1
[root@itsoredc kubernetes]#

[root@itsoredc kubernetes]# docker tag k8s.gcr.io/kube-proxy-s390x:v1.12.1
k8s.gcr.io/kube-proxy:v1.12.2
root@itsoredc kubernetes]#
```

Now, `kubeadm` can be started to bootstrap Kubernetes into the master node, as shown in Example 3-76.

Example 3-76 Bootstrap the master node with kubeadm

```
root@itsoredc ~]# kubeadm init --pod-network-cidr 10.1.0.0/24
[init] using Kubernetes version: v1.12.1
[preflight] running pre-flight checks
.
.
.
Your Kubernetes master has initialized successfully!
```

To start using your cluster, you need to run the following as a regular user:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

You should now deploy a pod network to the cluster.

Run `"kubectl apply -f [podnetwork].yaml"` with one of the options listed at: <https://kubernetes.io/docs/concepts/cluster-administration/addons/>

You can now join any number of machines by running the following on each node as root:

```
kubeadm join 9.76.61.241:6443 --token jwggxe.6lkxc0iloqh040rq
--discovery-token-ca-cert-hash
sha256:cdfb61eeb4a457e08ffe0c1482e7209d60557e8dcf8b30fb085858da0c275614
```

As a regular user and *not* as root, copy the `kubernetes/admin.conf` file to the user's home directory, as shown in Example 3-77.

Example 3-77 Copy kubernetes/admin.conf

```
[lnxadmin@itsoredc ~]$ mkdir -p $HOME/.kube
[lnxadmin@itsoredc ~]$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
[sudo] password for lnxadmin:
[lnxadmin@itsoredc ~]$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
[lnxadmin@itsoredc ~]$
```

Kubernetes is now installed but the networking layer still must be added. Several different networking layers are supported by Kubernetes, including Calico, flannel, and Kube-Router. Care needs to be taken when selecting a networking layer to ensure support for the s390x architecture.

Installing the networking component on the Master Node

Several pod networking add-ons are available and most have a s390x architecture image for Linux on z and LinuxONE installations. For the purposes of this book, we use Kube-Router.

For more information about setting up a network, see *Installing Kubernetes on Linux on Z and LinuxONE Using the kubeadm Installation Toolkit*, [TD106419](#).

Note: Before proceeding with installing Kubernetes on the worker nodes, the networking component must be installed, which is *not* done as part of the installation process.

For more information about how to install Kube-Router, see [this web page](#).

As root, Install the dependences vim and git (if not already installed), as shown in Example 3-78.

Example 3-78 Installing vim and git

```
[root@itsoredc kubernetes]# sudo yum install vim git
Installed:
git.s390x 0:1.8.3.1-13.e17

Dependency Installed:
perl-Error.noarch 1:0.17020-2.e17          perl-Git.noarch 0:1.8.3.1-13.e17
perl-TermReadKey.s390x 0:2.30-20.e17

Complete!
```

Go and gcc are also needed (see Example 3-79).

Example 3-79 Installing Go and gcc

```
root@itsoredc bin]# sudo yum install gcc
Installed:
gcc.s390x 0:4.8.5-28.e17

Dependency Installed:
cpp.s390x 0:4.8.5-28.e17  glibc-devel.s390x 0:2.17-222.e17  glibc-headers.s390x
0:2.17-222.e17  kernel-headers.s390x 0:3.10.0-862.e17
libmpc.s390x 0:1.0.1-3.e17
```

Complete!

```
[lnxadmin@itsoredc ~]$ mkdir go
[lnxadmin@itsoredc ~]$ cd go
[lnxadmin@itsoredc go]$ wget
https://storage.googleapis.com/golang/go1.10.1.linux-s390x.tar.gz
--2018-10-15 10:45:04--
https://storage.googleapis.com/golang/go1.10.1.linux-s390x.tar.gz
Resolving storage.googleapis.com (storage.googleapis.com)... 172.217.1.208,
2607:f8b0:400f:805::2010
Connecting to storage.googleapis.com
(storage.googleapis.com)|172.217.1.208|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 100662960 (96M) [application/octet-stream]
Saving to: 'go1.10.1.linux-s390x.tar.gz'

100%[=====
=====
=====] 100,662,960 33.4MB/s  in 2.9s

2018-10-15 10:45:08 (33.4 MB/s) - 'go1.10.1.linux-s390x.tar.gz' saved
[100662960/100662960]

[lnxadmin@itsoredc go]$ sudo tar -C /usr/local -xzf go1.10.1.linux-s390x.tar.gz
[lnxadmin@itsoredc go]$ export PATH=$PATH:/usr/local/go/bin
[lnxadmin@itsoredc go]$ cd /usr/bin
[lnxadmin@itsoredc bin]$ sudo -ln gcc s390x-linux-gnu-gcc

### Verify Go is properly setup
[lnxadmin@itsoredc bin]$ go version
go version go1.10.1 linux/s390x
[lnxadmin@itsoredc bin]$
```

Download the source for the s390x image of kube-router (see Example 3-80).

Example 3-80 Downloading kube-router

```
[lnxadmin@itsoredc go]$ export GOPATH=~/.go
[lnxadmin@itsoredc go]$ echo $GOPATH
/home/lnxadmin/go
[lnxadmin@itsoredc go]$ git clone
https://github.com/cloudnativelabs/kube-router.git
$GOPATH/src/github.com/cloudnativelabs/kube-router
Cloning into '/home/lnxadmin/go/src/github.com/cloudnativelabs/kube-router'...
remote: Enumerating objects: 59, done.
remote: Counting objects: 100% (59/59), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 32173 (delta 28), reused 31 (delta 17), pack-reused 32114
Receiving objects: 100% (32173/32173), 49.01 MiB | 22.37 MiB/s, done.
Resolving deltas: 100% (12474/12474), done.
[lnxadmin@itsoredc go]$ cd $GOPATH/src/github.com/cloudnativelabs/kube-router
[lnxadmin@itsoredc kube-router]$ ls
build build-image cmd cni contrib CONTRIBUTING.md daemonset dashboard
Dockerfile docs Gopkg.lock Gopkg.toml hack LICENSE MAINTAINER.md Makefile
pkg README.md vendor
[lnxadmin@itsoredc kube-router]$ git checkout v0.2.0
```

Note: checking out 'v0.2.0'.

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-b` with the checkout command again. Example:

```
git checkout -b new_branch_name
```

```
HEAD is now at 7496b00... dont shutdown gobgp server if graceful restart is enabled (#526)
```

```
[lnxadmin@itsoredc kube-router]$
```

Modify `$GOPATH/src/github.com/cloudnativelabs/kube-router/Makefile`, as shown in Example 3-81.

Example 3-81 Modifying `$GOPATH/src/github.com/cloudnativelabs/kube-router/Makefile`

```
@@ -26,6 +26,10 @@ else ifeq ($(GOARCH), arm64)
ARCH_TAG_PREFIX=$(GOARCH)
FILE_ARCH=ARM aarch64
DOCKERFILE_SED_EXPR?=
+else ifeq ($(GOARCH), s390x)
+ARCH_TAG_PREFIX=$(GOARCH)
+FILE_ARCH=IBM S/390
+DOCKERFILE_SED_EXPR?=
else
DOCKERFILE_SED_EXPR?=
FILE_ARCH=x86-64
```

Build the image and binary, as shown in Example 3-82.

Example 3-82 Building Kube-Router image and binary

```
[lnxadmin@itsoredc kube-router]$ make container
Building for GOARCH=s390x
Verifying kube-router gobgp for ARCH=IBM S/390 ...
Starting kube-router container image build.
sudo docker build -t "cloudnativelabs/kube-router-git:s390x-HEAD" -f
Dockerfile.s390x.run .
[sudo] password for lnxadmin:
Sending build context to Docker daemon 89.61MB
Step 1/10 : FROM alpine:3.7
----> e02e5498dc4d
Step 2/10 : RUN apk add --no-cache      iptables      ipset      iproute2
ipvsadm      contrack-tools      curl      bash &&      mkdir -p /var/lib/gobgp
&&      mkdir -p /usr/local/share/bash-completion &&      curl -L -o
/usr/local/share/bash-completion/bash-completion
https://raw.githubusercontent.com/scop/bash-completion/master/bash_completion
----> Running in 0d80e14252de
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/main/s390x/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.7/community/s390x/APKINDEX.tar.gz
(1/26) Installing pkgconf (1.3.10-r0)
(2/26) Installing ncurses-terminfo-base (6.0_p20171125-r1)
```

```

(3/26) Installing ncurses-terminfo (6.0_p20171125-r1)
(4/26) Installing ncurses-libs (6.0_p20171125-r1)
(5/26) Installing readline (7.0.003-r0)
(6/26) Installing bash (4.4.19-r1)
Executing bash-4.4.19-r1.post-install
(7/26) Installing libmnl (1.0.4-r0)
(8/26) Installing libnftnl (1.0.1-r1)
(9/26) Installing libnetfilter_contrack (1.0.6-r0)
(10/26) Installing libnetfilter_cthelper (1.0.0-r0)
(11/26) Installing libnetfilter_cttimeout (1.0.0-r0)
(12/26) Installing libnetfilter_queue (1.0.2-r0)
(13/26) Installing contrack-tools (1.4.4-r0)
(14/26) Installing ca-certificates (20171114-r0)
(15/26) Installing libssh2 (1.8.0-r2)
(16/26) Installing libcurl (7.61.1-r0)
(17/26) Installing curl (7.61.1-r0)
(18/26) Installing libelf (0.8.13-r3)
(19/26) Installing jansson (2.10-r0)
(20/26) Installing libnftnl-libs (1.0.8-r1)
(21/26) Installing iptables (1.6.1-r1)
(22/26) Installing iproute2 (4.13.0-r0)
Executing iproute2-4.13.0-r0.post-install
(23/26) Installing ipset (6.34-r1)
(24/26) Installing libnl (1.1.4-r0)
(25/26) Installing popt (1.16-r7)
(26/26) Installing ipvsadm (1.29-r0)
Executing busybox-1.27.2-r11.trigger
Executing ca-certificates-20171114-r0.trigger
OK: 21 MiB in 39 packages
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 72371  100 72371    0     0  515k      0  --:--:-- --:--:-- --:--:--  515k
---> afbf40c64740
Removing intermediate container 0d80e14252de
Step 3/10 : ADD build/image-assets/bashrc /root/.bashrc
---> a655ca7544c1
Removing intermediate container f818678813aa
Step 4/10 : ADD build/image-assets/profile /root/.profile
---> bf9b3d279d38
Removing intermediate container 83e9567c2c9b
Step 5/10 : ADD build/image-assets/vimrc /root/.vimrc
---> f139fb73193e
Removing intermediate container 025b6a03e3b3
Step 6/10 : ADD build/image-assets/motd-kube-router.sh /etc/motd-kube-router.sh
---> 3193f7a4a5e3
Removing intermediate container 4da6584555f3
Step 7/10 : ADD kube-router gobgp /usr/local/bin/
---> 785de84c348e
Removing intermediate container 4032eaafa0eb
Step 8/10 : RUN cd && /usr/local/bin/gobgp --gen-cmpl --bash-cmpl-file
/var/lib/gobgp/gobgp-completion.bash
---> Running in d072ab18750e
---> ef144b6bd40d
Removing intermediate container d072ab18750e
Step 9/10 : WORKDIR "/root"

```

```
---> 8f38fa9c942e
Removing intermediate container b2d1ecd61091
Step 10/10 : ENTRYPOINT /usr/local/bin/kube-router
---> Running in 6264342516b1
---> b2d389827dd8
Removing intermediate container 6264342516b1
Successfully built b2d389827dd8
Successfully tagged cloudnativelabs/kube-router-git:s390x-HEAD
Finished kube-router container image build.
[lnxadmin@itsoredc kube-router]$
```

Run tests, as shown in Example 3-83.

Example 3-83 Testing kube-router binary

```
[lnxadmin@itsoredc kube-router]$ make test
Building for GOARCH=s390x
Everything is gofmt approved!
go test github.com/cloudnativelabs/kube-router/cmd/kube-router/
github.com/cloudnativelabs/kube-router/pkg/...
ok github.com/cloudnativelabs/kube-router/cmd/kube-router0.075s
? github.com/cloudnativelabs/kube-router/pkg/cmd[no test files]
ok github.com/cloudnativelabs/kube-router/pkg/controllers0.008s
? github.com/cloudnativelabs/kube-router/pkg/controllers/netpol[no test files]
ok github.com/cloudnativelabs/kube-router/pkg/controllers/proxy0.085s [no tests
to run]
ok github.com/cloudnativelabs/kube-router/pkg/controllers/routing2.194s
? github.com/cloudnativelabs/kube-router/pkg/healthcheck[no test files]
? github.com/cloudnativelabs/kube-router/pkg/metrics[no test files]
? github.com/cloudnativelabs/kube-router/pkg/options[no test files]
ok github.com/cloudnativelabs/kube-router/pkg/utis0.055s
[lnxadmin@itsoredc kube-router]$
```

Note: The next tasks in this process are important; therefore, they must be followed carefully.

When starting Kube-Router, the default behavior is to download the x86_64 image of Kube-Router and Kube-Proxy. Kubernetes must be “tricked” into thinking the x86_64 images are present to prevent them from being downloaded. This process is done by tagging the s390x images as “latest”.

As root, download the .yaml file for Kube-Router, as shown in Example 3-84.

Example 3-84 Downloading kubeadm-kuberouter.yaml

```
[root@itsoredc kubernetes]# wget
https://raw.githubusercontent.com/cloudnativelabs/kube-router/master/daemonset/kub
eadm-kuberouter.yaml
--2018-10-15 11:46:38--
https://raw.githubusercontent.com/cloudnativelabs/kube-router/master/daemonset/kub
eadm-kuberouter.yaml
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.68.133
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|151.101.68.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3681 (3.6K) [text/plain]
```


Saving to: 'kubeadm-kuberouter.yaml'

```
100%[=====
=====
=====] 3,681      --.-K/s   in 0s
```

2018-10-15 11:46:39 (95.5 MB/s) - 'kubeadm-kuberouter.yaml' saved [3681/3681]

[root@itsoredc kubernetes]#

Modify the downloaded version of kubeadm-kuberouter.yaml to change imagePullPolicy: Always to imagePullPolicy: IfNotPresent. This change prevents Docker from always downloading the latest image of Kube-Router and replacing the s390x version with an x86_64 version.

Next, tag the Kube-Router Docker image that was previously built to match the name expected in the .yaml file, as shown in Example 3-85.

Example 3-85 Tagging the kube-router docker image

```
[root@itsoredc kubernetes]# docker image tag
cloudnativelabs/kube-router-git:s390x-HEAD cloudnativelabs/kube-router:latest
docker tag gcr.io/google-containers/kube-proxy-s390x:v1.12.1
gcr.io/google-containers/kube-proxy-amd64:v1.12.1
```

Create the /etc/cni/net.d/10-kuberouter.conf file that is needed for Kube-Router to work (this process must be done on each node), as shown in Example 3-86.

Example 3-86 Downloading 10-kuberouter.conf

```
[root@itsoredc kubernetes]# wget -O /etc/cni/net.d/10-kuberouter.conf
https://raw.githubusercontent.com/cloudnativelabs/kube-router/master/cni/10-kubero
uter.conf
--2018-10-15 12:11:47--
https://raw.githubusercontent.com/cloudnativelabs/kube-router/master/cni/10-kubero
uter.conf
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.68.133
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|151.101.68.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 151 [text/plain]
Saving to: '/etc/cni/net.d/10-kuberouter.conf'
```

```
100%[=====
=====
=====] 151      --.-K/s   in 0s
```

2018-10-15 12:11:47 (54.9 MB/s) - '/etc/cni/net.d/10-kuberouter.conf' saved [151/151]

[root@itsoredc kubernetes]#

At this point, Kube-Router is ready to be started. Use the command that is shown in Example 3-87.

Example 3-87 Starting kube-router

```
[root@itsoredc kubernetes]# kubectl apply -f kubeadm-kuberouter.yaml
configmap/kube-router-cfg created
daemonset.extensions/kube-router created
serviceaccount/kube-router created
clusterrole.rbac.authorization.k8s.io/kube-router created
clusterrolebinding.rbac.authorization.k8s.io/kube-router created
```

The status of Kubernetes with Kube-Router can be verified. All containers should have a status of “running”, as shown in Example 3-88. (This process took 2 - 3 minutes in our lab environment.)

Example 3-88 Getting status of environment

```
[lnxadmin@itsoredc kubernetes]$ kubectl get pods --namespace=kube-system
NAME                                READY   STATUS    RESTARTS   AGE
coredns-576cbf47c7-5vz6t           1/1     Running   0          46m
coredns-576cbf47c7-dtcrp           1/1     Running   0          46m
etcd-itsoredc.cpolab.ibm.com        1/1     Running   0          46m
kube-apiserver-itsoredc.cpolab.ibm.com 1/1     Running   0          46m
kube-controller-manager-itsoredc.cpolab.ibm.com 1/1     Running   0          46m
kube-proxy-b78qk                    1/1     Running   0          46m
kube-router-5q2vc                   1/1     Running   0          86s
kube-scheduler-itsoredc.cpolab.ibm.com 1/1     Running   0          46m
[lnxadmin@itsoredc kubernetes]$
```

Installing kubernetes on the worker nodes

At this point, with Kubernetes base set up and the networking overlay installed (in our case, Kube-Router), Kubernetes can now be bootstrapped with kubeadm on the worker nodes.

On the worker node (itsoredd in our environment), repeat the steps that were used on the master node to install kubectl, kubeadm, and kubelet, as shown in Example 3-89.

Example 3-89 Installing kubectl, kubeadm and kubelet on a worker node

```
[root@itsoredc ~]# cat <<EOF > /etc/yum.repos.d/kubernetes.repo
> [kubernetes]
> name=Kubernetes
> baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-s390x
> enabled=1
> gpgcheck=1
> repo_gpgcheck=1
> gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
> exclude=kube*
> EOF

[root@itsoredd ~]#
[root@itsoredd ~]# setenforce 0
setenforce: SELinux is disabled
[root@itsoredd ~]# yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
```

Create the `k8s.conf` configuration file, as shown in Example 3-90.

Example 3-90 Creating `k8s.conf`

```
[root@itsoredd ~]# cat <<EOF > /etc/sysctl.d/k8s.conf
> net.bridge.bridge-nf-call-ip6tables = 1
> net.bridge.bridge-nf-call-iptables = 1
> EOF
[root@itsoredd ~]# sysctl --system
* Applying /usr/lib/sysctl.d/00-system.conf ...
net.ipv4.tcp_syncookies = 1
.
.
.
* Applying /etc/sysctl.d/k8s.conf ...
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
* Applying /etc/sysctl.conf ...
fs.file-max = 6815744
[root@itsoredd ~]#
```

Disable swap and allow ports, as shown in Example 3-91.

Example 3-91 Disabling swap and allow ports

```
[root@itsoredd ~]# swapoff -a
[root@itsoredd ~]# firewall-cmd --zone=public --add-port=6443/tcp --permanent
success
[root@itsoredd ~]# firewall-cmd --zone=public --add-port=10250/tcp --permanent
success
[root@itsoredd ~]# firewall-cmd --zone=public --add-port=30000-32767/tcp
--permanent
success
[root@itsoredd ~]# firewall-cmd --reload
success
[root@itsoredd ~]# iptables-save | grep 6443
-A IN_public_allow -p tcp -m tcp --dport 6443 -m conntrack --ctstate NEW -j ACCEPT
[root@itsoredd ~]# iptables-save | grep 10250
-A IN_public_allow -p tcp -m tcp --dport 10250 -m conntrack --ctstate NEW -j
ACCEPT
[root@itsoredd ~]#
```

Set up `Kubernetes.conf`, as shown in Example 3-92.

Example 3-92 Setting up `kubernetes.conf`

```
wget -O /etc/cni/net.d/10-kubernetes.conf
https://raw.githubusercontent.com/cloudnativelabs/kube-router/master/cni/10-kuberouter.conf
--2018-10-15 14:47:58--
https://raw.githubusercontent.com/cloudnativelabs/kube-router/master/cni/10-kuberouter.conf
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.68.133
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|151.101.68.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 151 [text/plain]
```

Saving to: '/etc/cni/net.d/10-kuberouter.conf'

```
100%[=====
=====
=====] 151      --.-K/s   in 0s
```

2018-10-15 14:47:58 (52.4 MB/s) - '/etc/cni/net.d/10-kuberouter.conf' saved
[151/151]

[root@itsoredd cni]#

Kube-Router is automatically pulled when the worker node is joined to the cluster. The default is to pull the amd64 image. As with the master node, the s390x image must be pulled and tagged. This process can be done by pulling the s390x source and building a container, as was done on the master node.

However, an easier method is to copy the container from the master node over to the worker nodes. To make this copy, tar up the container on the master node and copy it to the worker node, and extract it there (see Example 3-93).

Example 3-93 Copying Kube-Router container to worker node

```
[root@itsoreddc kubernetes]# docker image ls
REPOSITORY          TAG          IMAGE ID
CREATED            SIZE
cloudnativelabs/kube-router-git      s390x-HEAD
b2d389827dd8        2 weeks ago 104MB
cloudnativelabs/kube-router          latest
b2d389827dd8        2 weeks ago 104MB
[root@itsoreddc kubernetes]#
[root@itsoreddc kubernetes]# docker save cloudnativelabs/kube-router >
kube-router.tar
[root@itsoreddc kubernetes]# scp kube-router.tar itsoredb.cpolab.ibm.com:/tmp
The authenticity of host 'itsoredb.cpolab.ibm.com (9.76.61.240)' can't be
established.
ECDSA key fingerprint is SHA256:wdt438eBfCIw8DEsqLwTeVOXBknmATpfxB/oYDnFvA.
ECDSA key fingerprint is MD5:bc:f4:4f:6e:7e:9b:77:39:0d:45:45:40:06:4e:76:10.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'itsoredb.cpolab.ibm.com,9.76.61.240' (ECDSA) to the
list of known hosts.
root@itsoredb.cpolab.ibm.com's password:
kube-router.tar
100% 102MB 51.0MB/s 00:02
[root@itsoreddc kubernetes]#

[root@itsoredb tmp]# docker load < kube-router.tar
6b3831aa6fe1: Loading layer [=====]
4.998MB/4.998MB
c72ba0b3db9b: Loading layer [=====]
2.56kB/2.56kB
46173c62813f: Loading layer [=====]
13.42MB/13.42MB
906ac9dd99db: Loading layer [=====]
3.584kB/3.584kB
```

```

b126c7247f70: Loading layer [=====>]
2.56kB/2.56kB
c99b219e0b58: Loading layer [=====>]
2.56kB/2.56kB
c9df78d2ccbd: Loading layer [=====>]
3.584kB/3.584kB
025a9158e790: Loading layer [=====>]
88.44MB/88.44MB
94a83fb65c42: Loading layer [=====>]
62.46kB/62.46kB
Loaded image: cloudnativelabs/kube-router:latest
[root@itsoredb tmp]#

```

You also must get kube-proxy for s390x, as shown in Example 3-94.

Example 3-94 Download kube-proxy

```

[root@itsoredb kubernetes]# docker pull
gcr.io/google-containers/kube-proxy-s390x:v1.12.1
v1.12.1: Pulling from google-containers/kube-proxy-s390x
52fb6aa6b72f: Pull complete
7087cdfd2c22: Pull complete
1fb073cf059c: Pull complete
Digest: sha256:b056eb9eef26bf8af3e236d5db783bdcea4faf1d6ef20c0dd6870a4ba556984c
Status: Downloaded newer image for
gcr.io/google-containers/kube-proxy-s390x:v1.12.1
[root@itsoredb kubernetes]# docker image tag
gcr.io/google-containers/kube-proxy-s390x:v1.12.1
gcr.io/google-containers/kube-proxy-amd64:v1.12.1
[root@itsoredb kubernetes]#

```

At this point, the worker node is ready to join the cluster. Refer to the `join` command that was provided on the master when `kubeinit` was run (see Example 3-95).

Example 3-95 Joining the cluster from the worker node

```

kubeadm join 9.76.61.241:6443 --token o5voi1.hbg4y2eq6nbmpcwt
--discovery-token-ca-cert-hash
sha256:834d644caeb828f11ca1040123bfcac9692cacef203fa9735c84afadd98a4a3a
[preflight] running pre-flight checks
  [WARNING RequiredIPVSKernelModulesAvailable]: the IPVS proxier will not be
used, because the following required kernel modules are not loaded: [ip_vs_rr
ip_vs_wrr ip_vs_sh ip_vs] or no builtin kernel ipvs support: map[ip_vs:{}
ip_vs_rr:{} ip_vs_wrr:{} ip_vs_sh:{} nf_contrack_ipv4:{}]
you can solve this problem with following methods:
  1. Run 'modprobe -- ' to load missing kernel modules;
  2. Provide the missing builtin kernel ipvs support

[discovery] Trying to connect to API Server "9.76.61.241:6443"
[discovery] Created cluster-info discovery client, requesting info from
"https://9.76.61.241:6443"
[discovery] Requesting info from "https://9.76.61.241:6443" again to validate TLS
against the pinned public key
[discovery] Cluster info signature and contents are valid and TLS certificate
validates against pinned roots, will use API Server "9.76.61.241:6443"
[discovery] Successfully established connection with API Server "9.76.61.241:6443"

```

```
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.12"
ConfigMap in the kube-system namespace
[kubelet] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet] Writing kubelet environment file with flags to file
"/var/lib/kubelet/kubeadm-flags.env"
[preflight] Activating the kubelet service
[tlsbootstrap] Waiting for the kubelet to perform the TLS Bootstrap...
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the
Node API object "itsoredd.cpolab.ibm.com" as an annotation
```

This node has joined the cluster:

- * Certificate signing request was sent to apiserver and a response was received.
- * The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.

Running **kubectl** verifies the nodes on the master, as shown in Example 3-96.

Example 3-96 kubectl get nodes

```
[lnxadmin@itsoreddc kubernetes]$ kubectl get nodes
NAME                                STATUS    ROLES    AGE     VERSION
itsoreddc.cpolab.ibm.com           Ready    master   156m    v1.12.1
itsoredd.cpolab.ibm.com           NotReady <none>   2m35s   v1.12.1
[lnxadmin@itsoreddc kubernetes]$
```

After a few minutes, on the master:

```
NAME                                STATUS    ROLES    AGE     VERSION
itsoreddc.cpolab.ibm.com           Ready    master   3h41m   v1.12.1
itsoredd.cpolab.ibm.com           Ready    <none>   68m     v1.12.1
```

Setting up the Kubernetes dashboard

Kubernetes provides a graphical user interface by way of a web page. It allows users to manage applications that are running in the cluster and troubleshoot them, and manage the cluster. For more information and setup instructions, see [this web page](#).

Setting up on Linux on z and LinuxONE systems includes some important consideration. Otherwise, the default behavior when starting dashboard is to download an x86_64 image that does not work.

Because the dashboard uses the Kubernetes API server (which runs on the master node), the dashboard Docker image must be installed on the master. To have Docker download the s390x image, first download the .yaml file to /etc/kubernetes/kubernetes-dashboard.yaml and make the following change image: value to:

```
k8s.gcr.io/kubernetes-dashboard-s390x:v1.10.0
```

Also, make the following change to allow the dashboard to run on the master:

```
- key: node-role.kubernetes.io/master
  effect: NoSchedule
nodeSelector:
  node-role.kubernetes.io/master: ""
```

The dashboard can be deployed by using the command that is shown in Example 3-97.

Example 3-97 Deploying dashboard

```
[lnxadmin@itsoredc kubernetes]$ kubectl apply -f kubernetes-dashboard.yaml
secret/kubernetes-dashboard-certs created
serviceaccount/kubernetes-dashboard created
role.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
deployment.apps/kubernetes-dashboard created
service/kubernetes-dashboard created
[lnxadmin@itsoredc kubernetes]$
```

Verify that the dashboard is running by using the command that is shown in Example 3-98.

Example 3-98 Verifying that dashboard is operational

```
[lnxadmin@itsoredc kubernetes]$ kubectl get services -n kube-system
NAME                                TYPE                CLUSTER-IP          EXTERNAL-IP          PORT(S)
AGE
kube-dns                            ClusterIP            10.96.0.10          <none>                53/UDP,53/TCP
47h
kubernetes-dashboard                ClusterIP            10.111.198.211     <none>                443/TCP
43h
```

Dashboard uses api-server for access. When kubectl is used, authorization is not set up automatically. By using the command that is shown in Example 3-99, the api-server can be accessed anonymously and thus to dashboard; however, resources cannot be accessed.

Example 3-99 Accessing api-server anonymously

```
[lnxadmin@itsoredc ~]$ vim dashboard-admin.yaml
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: kubernetes-dashboard
  labels:
    k8s-app: kubernetes-dashboard
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: kubernetes-dashboard
  namespace: kube-system
[lnxadmin@itsoredc ~]$ kubectl create -f dashboard-admin.yaml
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
[lnxadmin@itsoredc ~]$
```

Note: Anonymous access to resources in the ITSO internal environment was acceptable at the time of this writing. You should carefully review your organization's own security requirements when deciding how to implement user access in Dashboard.

Obtain the IP address of the master node where Dashboard is running by using the `cluster-info` command, as shown in Example 3-100.

Example 3-100 cluster-info command

```
[lnxadmin@itsoredc ~]$ kubectl cluster-info
Kubernetes master is running at https://9.76.61.241:6443
KubeDNS is running at
https://9.76.61.241:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

Use this IP address to connect to Dashboard (see Figure 3-30), which features the following syntax:

`https://<master-ip>:<apiserver-port>/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/`, where `<master-ip>` is IP address or domain name of the Kubernetes master

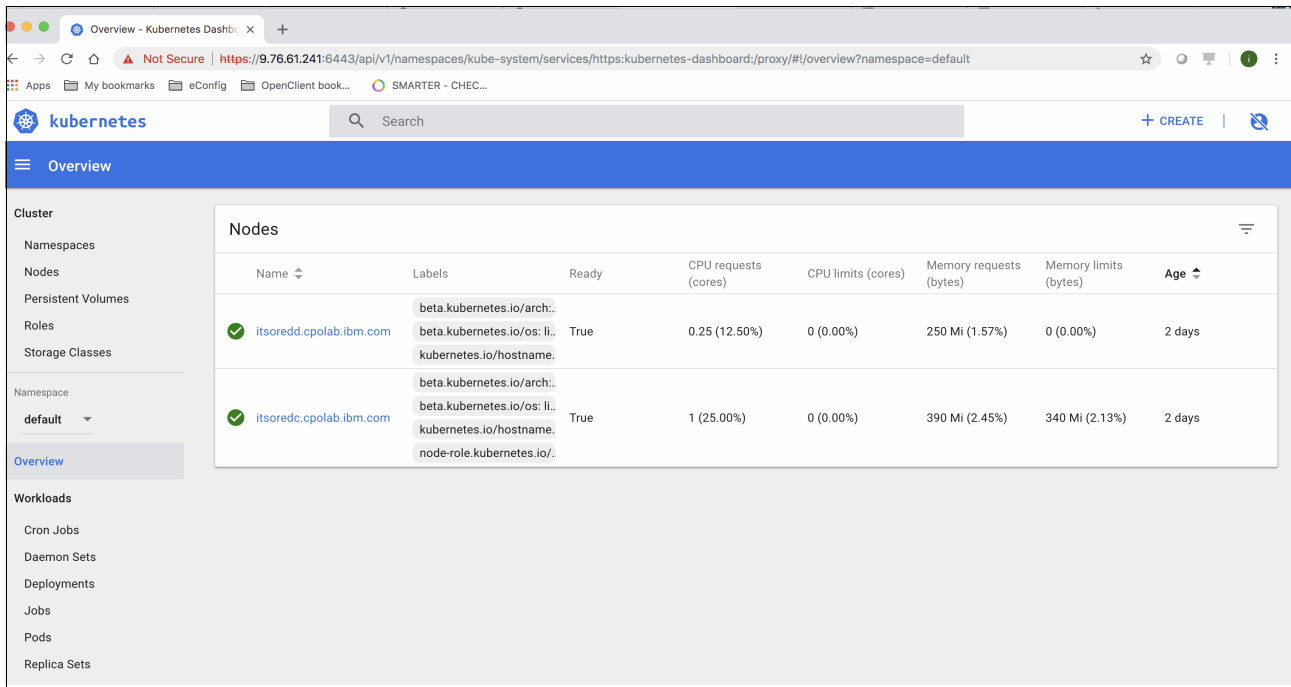


Figure 3-30 Kubernetes Dashboard

3.8 Kubernetes versus Docker Swarm

A simple environment of only a few containers can be managed relatively easily by a Docker administrator who uses personal notes, spreadsheets, and other, informal tracking methods. As the environment grows, it becomes critical to keep track of images. Fortunately, tracking is made possible by Docker Swarm, which is part of Docker, and by using an add-on tool that is available from Kubernetes.

Docker Swarm and Kubernetes are open source, orchestration tools that provide the same functions. The biggest difference between them is how they operate.

3.8.1 Kubernetes

Kubernetes still holds the largest market and uses the concept of pods, as described in 2.6.1, “Kubernetes” on page 28. It has a vast, open source community with many contributors.

It also supports higher demands with more complexity. It is a customizable tool, and the installation can be complex to deploy and manage. However, it is flexible and uses a separate set of tools for management, called kubectl CLI.

3.8.2 Docker Swarm

Docker Swarm has the advantage of being integrated with the Docker product and uses its own API. Because it is developed by Docker, Docker Swarm removes several compatibility and other differences that helps with tool integration. This feature makes the installation and faster.

It is less extensive and less customizable, but it is not well-suited for production deployment and it has a small community. It works well for a simple deployment of containers and it is integrated with Docker Compose and Docker CLI, which are the native Docker tools.

The features of Kubernetes and Docker Swarm are compared in Table 3-5.

Table 3-5 Features comparison

Feature	Kubernetes	Docker Swarm
Networking	Flat network that allow all pods to communicate with one another. This flat network is typically implemented as an overlay.	A node that joins a Docker Swarm cluster to create an overlay network.
Performance and scalability	A complex system. Because of guarantees about the cluster state, container scalability and deployment is slowed.	Can deploy containers much faster and allows faster reaction times to scale on-demand.
High availability	HA is supported, pods are distributed among nodes to provide HA, and load balance service detect unhealthy pods and remove them.	Docker Swarm offers HA, because the services can be replicated in swarm nodes. An odd number of managers is recommended.
Container setup	Uses its own YAML, and can be complex but flexible.	Simple deployment and integrated with Docker Compose and Docker CLI.
Load balancing	Pods are exposed by way of service, which can be used as a load balance for the cluster.	Consists of a DNS that is used for distributing incoming requests to a service name.
Health checks	Two types are available: <ul style="list-style-type: none">▶ Liveness (if the application is responsive)▶ Readiness (if the application responsive but busy because it is preparing, but cannot serve yet)	Limited to services. If a container is not on running state, a new container is started.

Feature	Kubernetes	Docker Swarm
Service discovery	Can be discovered by using the environment or DNS. DNS server is available as an add-on. If DNS is enabled in the entire cluster, pods can use service names that automatically resolve.	Swarm Manager node assigns each service a unique DNS name and load balances running containers.

Kubernetes supports higher demands and offers a more customizable options and extensive use. If a project requires a more complex set up, this option can analyze the plan phase of your project to check which orchestration tool is adequate to a new environment.

If a project needs to set up something quickly, Docker Swarm is available. It is important to understand which features are needed and then check which tool provides more benefit. You can then choose the more adequate and recommended tool to your environment.

For more information about Docker Swarm, see Chapter 5.3, “Docker in swarm mode” on page 148. For more information about Kubernetes, see Chapter 5.4, “Kubernetes” on page 162.



Basic Docker operations

As a new user of Docker on IBM Z, you need to develop an understanding of the hardware platform that runs the Linux operating system. Linux is designed to make full use of the hardware and its many sophisticated peripheral devices and features.

This chapter introduces you to some of the useful and frequently used Docker commands for system administrators and developers. Some of the most commonly used commands can be used to configure the environment and install, configure, and manage containers.

This chapter includes the following topics:

- ▶ 4.1, “Linux on Z commands” on page 102
- ▶ 4.2, “Basic Docker commands” on page 105
- ▶ 4.3, “Swarm commands” on page 116
- ▶ 4.4, “Kubernetes commands” on page 122
- ▶ 4.5, “Backing up a container” on page 124

4.1 Linux on Z commands

In this section, we describe are some of the more common Linux commands that can be used on Linux on IBM Z.

4.1.1 Overview

Some commands include an init script, a configuration file, or both. It is assumed that init scripts are installed in `/etc/init.d/`. You can extract any missing files from the `/etc` subdirectory in the `s390-tools` package.

lscss

The use of the command that is shown in Example 4-1 shows information about all of the subchannels from the Linux virtual file system that are named `sysfs`. The information is displayed in summary format.

Example 4-1 Use `od lscss` display command

```
itsoslec:~ # lscss
Device    Subchan.  DevType CU Type Use  PIM PAM POM  CHPIDs
-----
0.0.0600  0.0.0000  1732/01 1731/01 yes  80  80  ff  00000000 00000000
0.0.0601  0.0.0001  1732/01 1731/01 yes  80  80  ff  00000000 00000000
0.0.0602  0.0.0002  1732/01 1731/01 yes  80  80  ff  00000000 00000000
0.0.0620  0.0.0003  1732/01 1731/01      80  80  ff  01000000 00000000
0.0.0621  0.0.0004  1732/01 1731/01      80  80  ff  01000000 00000000
0.0.0622  0.0.0005  1732/01 1731/01      80  80  ff  01000000 00000000
0.0.0500  0.0.0007  3390/0e 3990/e9 yes  80  80  ff  ff000000 00000000
0.0.0100  0.0.0008  3390/0e 3990/e9 yes  80  80  ff  ff000000 00000000
0.0.0200  0.0.0009  3390/0e 3990/e9 yes  80  80  ff  ff000000 00000000
0.0.000c  0.0.000a  0000/00 2540/00      80  80  ff  ff000000 00000000
0.0.000d  0.0.000b  0000/00 2540/00      80  80  ff  ff000000 00000000
0.0.000e  0.0.000c  0000/00 1403/00      80  80  ff  ff000000 00000000
0.0.0009  0.0.000d  0000/00 3215/00 yes  80  80  ff  ff000000 00000000
0.0.019f  0.0.0011  3390/0e 3990/e9      80  80  ff  ff000000 00000000
0.0.0300  0.0.0015  9336/10 6310/80 yes  80  80  ff  ff000000 00000000
0.0.0301  0.0.0016  9336/10 6310/80 yes  80  80  ff  ff000000 00000000
itsoslec:~ #
```

lsdasd

Example 4-2 shows the command that lists information about DASD devices from `sysfs`.

Example 4-2 Output of `lsdasd` command

```
itsoslec:~ # lsdasd
Bus-ID    Status    Name      Device  Type  BlkSz  Size      Blocks
-----
0.0.0100  active    dasda     94:0    ECKD  4096   21129MB   5409180
0.0.0200  active    dasdb     94:4    ECKD  4096   21129MB   5409180
0.0.0300  active    dasdc     94:8    FBA   512    256MB     524288
0.0.0301  active    dasdd     94:12   FBA   512    512MB     1048576
0.0.0500  active    dasde     94:16   ECKD  4096   10564MB   2704500
itsoslec:~ #
```

modinfo

The command `modinfo` can be used with parameters to show information about the Linux kernel module.

Example 4-3 shows the command that provides kernel module information for the `btrfs` file system.

Example 4-3 Kernel module output for btrfs

```
itsoslec:~ # modinfo btrfs
filename:       /lib/modules/4.4.156-94.57-default/kernel/fs/btrfs/btrfs.ko
license:       GPL
alias:         devname:btrfs-control
alias:         char-major-10-234
alias:         fs-btrfs
srcversion:    6C9650B4874D020A776AE29
depends:        raid6_pq,xor
supported:     yes
intree:        Y
vermagic:      4.4.156-94.57-default SMP mod_unload modversions
signer:        SUSE Linux Enterprise Secure Boot Signkey
sig_key:       3F:B0:77:B6:CE:BC:6F:F2:52:2E:1C:14:8C:57:C7:77:C7:88:E3:E7
sig_hashalgo: sha256
parm:          allow_unsupported:Allow using feature that are out of supported
scope
itsoslec:~ #
```

If an overlay is in use, the use of the `modinfo overlay` command displays the information about the overlay, as shown in Example 4-4.

Example 4-4 Kernel module output for overlay

```
itsosleb:~ # modinfo overlay
filename:       /lib/modules/4.4.156-94.57-default/kernel/fs/overlayfs/overlay.ko
alias:         fs-overlayfs
alias:         fs-overlay
license:       GPL
description:    Overlay filesystem
author:        Miklos Szeredi <miklos@szeredi.hu>
srcversion:    5273297588F9E3388FDC816
depends:
supported:     yes
intree:        Y
vermagic:      4.4.156-94.57-default SMP mod_unload modversions
signer:        SUSE Linux Enterprise Secure Boot Signkey
sig_key:       3F:B0:77:B6:CE:BC:6F:F2:52:2E:1C:14:8C:57:C7:77:C7:88:E3:E7
sig_hashalgo: sha256
itsosleb:~ #
```

dasdfmt

This command is used to format the DASD for a Linux volume. Example 4-5 shows the formatting of the /dev/dasde disk.

Example 4-5 Output of formatting disk /dev/dasde

```
itsoslea:~ # /sbin/dasdfmt -b 4096 -y -f /dev/dasde -m 500
Printing hashmark every 500 cylinders.
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
Finished formatting the device.
Rereading the partition table... ok
```

fdasd

This command is used to create partitions on a specific disk. Example 4-6 shows the automatic creation of one partition that uses the whole disk in non-interactive mode.

Example 4-6 Output of fdasd command

```
itsoslea:~ # fdasd -a /dev/dasde
reading volume label ...: VOL1
reading vtoc .....: ok

auto-creating one partition for the whole disk...
writing volume label...
writing VTOC...
rereading partition table...
itsoslea:~ #
```

pvcreate

Example 4-7 shows the command that is used to create a physical volume.

Example 4-7 Creating a physical volume for disk /dev/dasde1

```
itsoslea:~ # pvcreate /dev/dasde1
Physical volume "/dev/dasde1" successfully created
```

vgcreate

Example 4-8 shows the command that creates a volume group by name and adds at least one physical volume to it.

Example 4-8 Use of the vgcreate command

```
itsoslea:~ # vgcreate dockervg /dev/dasde1
Volume group "dockervg" successfully created
itsoslea:~ #
```

lvcreate

This command creates a logical volume in a volume group. Example 4-9 shows this command, which creates a logical volume in the dokcervg volume group.

Example 4-9 Output of lvcreate command

```
itsoslea:~ # lvcreate -l 100%free -n var_lib_docker dockervg
Logical volume "var_lib_docker" created.
```

mount

This command logically attaches a file system. As shown in Example 4-10, the use of the **mount -a** command causes all file systems that are listed in the `fstab` file to be attached, except for those file systems that are listed as `noauto`.

Example 4-10 Example mount command

```
itsoslea:~ # mount -a
itsoslea:~ # df -h | grep docker
/dev/mapper/dockervg-var_lib_docker 11G 17M 11G 1% /var/lib/docker
```

useradd and usermod

The command **useradd** is used to add a user account to the operating system. The use of the **usermod** command modifies the setting of this user. In Example 4-11, a new user is added and then modified to assign this new user to a group. The next command in the example (**id dockeradmin**) displays the user and group names that are associated with the new user.

Example 4-11 Example of a user set-up

```
itsoslea:~ # useradd -m -c "docker administrator" -d /home/dockeradmin dockeradmin
-p passwordXX
itsoslea:~ # usermod -aG docker dockeradmin
itsoslea:~ # id dockeradmin
uid=1001(dockeradmin) gid=100(users) groups=482(docker),100(users)
itsoslea:~ #
```

4.2 Basic Docker commands

In this section, we introduce some basic Docker commands.

4.2.1 Docker info

This command displays system-wide information about the Docker installation, as shown in Example 4-12.

Example 4-12 Docker info output

```
itsosleb:~ # docker info
Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
Images: 2
Server Version: 17.06.2-ee-16
Storage Driver: overlay2
  Backing Filesystem: xfs
  Supports d_type: true
  Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
  Volume: local
  Network: bridge host macvlan null overlay
```

```

Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 6e23458c129b551d5c9871e5174f6b1b7f6d1170
runc version: 462c82662200a17ee39e74692f536067a3576a50
init version: 949e6fa
Security Options:
  apparmor
Kernel Version: 4.4.156-94.57-default
Operating System: SUSE Linux Enterprise Server 12 SP3
OSType: linux
Architecture: s390x
CPUs: 2
Total Memory: 15.63GiB
Name: itsosleb
ID: PBNE:M6RS:SJHZ:KGRT:TBD4:NJR3:WSHT:5YR5:E36Y:6UBW:ARNM:N6X7
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Experimental: false
Insecure Registries:
  127.0.0.0/8
Live Restore Enabled: false

WARNING: No swap limit support
WARNING: No kernel memory limit support
itsosleb:~ #

```

4.2.2 Docker ps

This command shows all running containers, as shown in Example 4-13. If you include `-a`, it shows all running and non-running (stopped) containers.

Example 4-13 Docker ps output

```

itsosleb:~ # docker ps
CONTAINER ID      IMAGE          COMMAND          CREATED
STATUS           PORTS         NAMES
9a862aea0126     nginx         "nginx -g 'daemon ..." 18 hours ago
Up 18 hours      80/tcp       hopeful_hoover

```

Example 4-14 shows the same `docker ps` command, but the results now show a running container that is listening on a specific network ports. You can see all of the ports that are mapped by using this command.

Example 4-14 Docker ps output with exposed ports

```

itsosleb:~ # docker ps
CONTAINER ID      IMAGE          COMMAND          CREATED          STATUS
PORTS           NAMES
c5e994288ff6     nginx         "/bin/bash"     About an hour ago Up About an
hour            0.0.0.0:80->80/tcp mywebcontainer

```

4.2.3 Docker rmi

A Docker image is an immutable file that is essentially a snapshot of a container. Images are created by using the **build** command and they produce a container when started by using the **run** command. Images are stored in a Docker registry.

The use of the **docker rmi *image name*** command removes one or more images (see Example 4-15).

Example 4-15 Docker rmi output

```
itsosleb:~ # docker rmi nginx
Untagged: nginx:latest
Untagged:
nginx@sha256:9ad0746d8f2ea6df3a17ba89eca40b48c47066dfab55a75e08e2b70fc80d929e
Deleted: sha256:a398f7e872a9fd128e7c96029db9a6ea8166b5f0371243675debd085e9802846
Deleted: sha256:840f95e5227ce83f513117be2f7115a8acab25b8dfc5e8c26487742a58e40a8d
Deleted: sha256:a10e7dbfae84e247fc7de243b303ccc7ed58c1038edca4194f8cdc26ade561d7
Deleted: sha256:20d026287261aea4a4623254f82ce85a6e1f94eeae2b31945640d205c2b06123
itsosleb:~ #
```

Note: To delete all images, run the **docker rmi \$(docker images -q)** command.

If you attempt to remove an image that is being used by a container, an error results, as shown in Example 4-16. In this case, you must determine whether the container is still needed. If the container is not needed not, run the **docker stop <id>** command and then the **docker rm <id>** command to stop and remove the container.

Example 4-16 Error message when trying to remove a Docker image

```
Error response from daemon: conflict: unable to remove repository reference
"nginx" (must force) - container 9a862aea0126 is using its referenced image
a398f7e872a9
```

4.2.4 Docker rm

The use of this command removes a Docker container, as shown in Example 4-17.

Example 4-17 Output of docker rm command

```
itsosleb:~ # docker rm 9a862aea0126
9a862aea0126
```

Note: To remove all containers, issue the **docker rm \$(docker ps -aq)** command. Use caution when you are deleting all of the containers.

4.2.5 Docker stop

The use of this command stops a running container, as shown in Example 4-18.

Example 4-18 Docker stop command

```
itsosleb:~ # docker stop 9a862aea0126  
9a862aea0126
```

4.2.6 Docker pull

The use of this command downloads an image from a Docker repository or registry, as shown in Example 4-19.

Example 4-19 Docker pull command and results

```
itsosleb:~ # docker pull nginx  
Using default tag: latest  
latest: Pulling from library/nginx  
599d69132c05: Pull complete  
bc12a10c7cc5: Pull complete  
54372546fe97: Pull complete  
Digest: sha256:9ad0746d8f2ea6df3a17ba89eca40b48c47066dfab55a75e08e2b70fc80d929e  
Status: Downloaded newer image for nginx:latest  
itsosleb:~ #
```

By default, the use of the **docker pull** command download images or repositories from the [Docker Hub website](#). Alternatively, you can specify where the image is downloaded.

For example, Red Hat provides their own Red Hat Enterprise Linux 7 in a fully featured and supported base image. How to download a container image for Red Hat Enterprise Linux is shown in Example 4-20.

Example 4-20 Docker pull command and results for RHEL

```
itsosleb:~ # docker pull registry.access.redhat.com/rhel7  
Using default tag: latest  
latest: Pulling from rhel7  
f8cc1f42a63a: Pull complete  
3d3c2dcd04a1: Pull complete  
Digest: sha256:33963577a88086a7cafc5efbed42c21f16dee63d79a9ebb416a231097aa44cf2  
Status: Downloaded newer image for registry.access.redhat.com/rhel7:latest  
itsosleb:~ #
```

Note: If the **docker pull** command does not work because no internet access is available, you can use the **docker save** and **docker load** commands to download the image, as shown in Example 4-56 on page 124.

4.2.7 Docker tag

Docker tags are aliases to the ID of a Docker image. It is another way to refer to an image. An example of the **docker tag** command is shown in Example 4-21, which assigns the image named `nginx` to the tag `mytag/nginx`.

Example 4-21 Output of docker tag command

```
itsosleb:~ # docker tag nginx mytag/nginx:1.0
```

Because tag clarifies information about the image, ensure that any use who pulls down an image knows what they are getting. It is also a best practice to use version control for images.

4.2.8 Docker build

The use of this command builds an image from the Dockerfile in the current directory, as shown in Example 4-22.

Example 4-22 Output of an image build by using docker build command

```
itsosleb:/docker_config # docker build --rm -t rh_oracle_image .
Sending build context to Docker daemon 3.072kB
Step 1/5 : FROM registry.access.redhat.com/rhel7/rhel
---> 099889565575
Step 2/5 : MAINTAINER Eric Marins <emarins@br.ibm.com>
---> Using cache
---> c938d81527d8
Step 3/5 : ADD DOCKER_START.sh /DOCKER_START.sh
---> Using cache
---> 857d05784ddc
Step 4/5 : RUN chmod +x /DOCKER_START.sh
---> Using cache
---> 4801932fcb05
Step 5/5 : ENTRYPOINT /DOCKER_START.sh
---> Running in 010aec53e093
---> cc1237d2f0cc
Removing intermediate container 010aec53e093
Successfully built cc1237d2f0cc
Successfully tagged rh_oracle_image:latest
```

The content of the Dockerfile that built this image is shown in Example 4-23.

Example 4-23 Dockerfile

```
itsosleb:/docker_config # cat Dockerfile
FROM registry.access.redhat.com/rhel7/rhel
MAINTAINER Eric Marins <emarins@br.ibm.com>

ADD DOCKER_START.sh /DOCKER_START.sh
RUN chmod +x /DOCKER_START.sh

ENTRYPOINT ["/DOCKER_START.sh"]
```

4.2.9 Docker run

The use of this command runs a process in an isolated container. The container process that runs is isolated in that it has its own file system, networking, and isolated process tree that is separate from the host, as shown in Example 4-24.

Example 4-24 Docker run command

```
tsosleb:~ # docker run -d -it --name mywebcontainer -p 80:80 nginx  
86a900286fcb4f26ed379550a83aa9597a026c75505b03743c60dfbba335bd22  
itsosleb:~ #
```

The command options are listed in Table 4-1.

Table 4-1 Docker run options

Option	Description
-d	Run container in background and print container ID.
-it	Connect the container to terminal.
--name	Set a name to your new container.
-p 80:80	Expose port 80 externally and redirect to container port 80.
nginx	The base image from where the container is created.

4.2.10 Docker container port

The use of the command lists the port mappings for a container, as shown in Example 4-25.

Example 4-25 Docker container port command

```
itsosleb:~ # docker container port 86a900286fcb  
80/tcp -> 0.0.0.0:80  
itsosleb:~ #
```

Note: You can use the command that is shown in Example 4-26 to find the container ports.

Example 4-26 Docker container inspect command

```
itsosleb:~ # docker container inspect 86a900286fcb -f "{{json  
.NetworkSettings.Ports}}"  
{ "80/tcp": [{"HostIp": "0.0.0.0", "HostPort": "80"}] }
```

4.2.11 Docker exec

The use of this command creates a process (bash) inside the container and connects it to a terminal. It is used to run a command in a running container, as shown in Example 4-27.

Example 4-27 Docker exec command

```
itsosleb:~ # docker exec -it c5e994288ff6 /bin/bash  
root@c5e994288ff6:/#
```

Note: This command is similar to an SSH connection for a Linux guest.

4.2.12 Docker attach

Use this command to attach your terminal's standard input, output, and error (or any combination of the three) to a running container by using the container's ID or name. The use of this command allows you to view its ongoing output or to control it interactively as though the commands were running directly in your terminal.

If the Docker container was started by using the `/bin/bash` command, you can access it by using `attach`. If it was not started in this manner, you must run the `attach` command to create a bash instance inside the container by using the `docker exec` command.

Note: The `docker attach` command is not used for running a new process in a container; instead, it is for attaching to the running process.

The command `docker exec` is used for running new items in a started container, be it a shell or some other process.

The use of the `docker attach` command is shown in Example 4-28.

Example 4-28 Docker attach command

```
# This container (d98d0460d6d5) was started using /bin/bash, so the attach command works.
itsoslec:~ # docker attach d98d0460d6d5
root@d98d0460d6d5:/#
```

4.2.13 Docker stop

The use of this command stops one or more running containers, as shown in Example 4-29.

Example 4-29 Docker stop command

```
itsosleb:~ # docker stop c5e994288ff6
c5e994288ff6
itsosleb:~ #
```

Note: To stop all containers, run the `docker stop $(docker ps -aq)` command.

4.2.14 Docker kill

The use of the **docker kill** command kills one or more containers. The main process inside the container is sent a SIGKILL signal (default), or the signal that is specified by using the **--signal** option. You can kill a container by using the container's ID (see Example 4-30), ID-prefix, or name.

Example 4-30 Output of docker kill command

```
itsosleb:~ # docker kill c5e994288ff6
c5e994288ff6
itsosleb:~ #
```

Note: To kill all running containers, run the **docker kill \$(docker ps -aq)** command.

4.2.15 Docker logs

The use of this command batch-retrieves logs that are present at the time of execution. As shown in Example 4-31, the last five lines of a container's logs are printed (these lines are the log's output from STDOUT/STDERR).

Example 4-31 Docker logs command for the last five lines

```
itsosleb:~ # docker logs --tail 5 86a900286fcb
2018/10/11 19:57:39 [error] 5#5: *1 open() "/usr/share/nginx/html/favicon.ico"
failed (2: No such file or directory), client: 9.57.215.135, server: localhost,
request: "GET /favicon.ico HTTP/1.1", host: "9.76.61.246"
9.57.215.135 - - [11/Oct/2018:19:57:39 +0000] "GET /favicon.ico HTTP/1.1" 404 153
 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0" "-"
9.57.232.251 - - [11/Oct/2018:19:57:41 +0000] "GET / HTTP/1.1" 200 612 "-"
 "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/68.0.3440.106 Safari/537.36" "-"
2018/10/11 19:57:41 [error] 5#5: *2 open() "/usr/share/nginx/html/favicon.ico"
failed (2: No such file or directory), client: 9.57.232.251, server: localhost,
request: "GET /favicon.ico HTTP/1.1", host: "9.76.61.246", referer:
"http://9.76.61.246/"
9.57.232.251 - - [11/Oct/2018:19:57:41 +0000] "GET /favicon.ico HTTP/1.1" 404 555
"http://9.76.61.246/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/68.0.3440.106 Safari/537.36" "-"
itsosleb:~ #
```

Note: This command is similar to running the **tail -n 5 -f /var/log/messages** command in Linux.

4.2.16 Docker diff

The use of this command lists the changed files and directories between a container's current image and its base image, as shown in Example 4-32.

Example 4-32 Docker diff command

```
itsosleb:~ # docker diff 86a900286fcb
C /run
A /run/nginx.pid
C /var
C /var/cache
C /var/cache/nginx
A /var/cache/nginx/proxy_temp
A /var/cache/nginx/scgi_temp
A /var/cache/nginx/uwsgi_temp
A /var/cache/nginx/client_temp
A /var/cache/nginx/fastcgi_temp
itsosleb:~ #
```

In column 1 of Example 4-32, a symbol is displayed that indicates the type of change that was tracked. Three different types of changes are tracked, as listed in Table 4-2.

Table 4-2 Symbols table

Symbol	Description
A	A file or directory was added
D	A file or directory was deleted
C	A file or directory was changed

4.2.17 Docker image

The use of this command provides a way of managing your Docker images, as shown in Example 4-33.

Example 4-33 List installed Docker images

```
itsosleb:~ # docker image ls
REPOSITORY          TAG          IMAGE ID          CREATED
SIZE
nginx                latest       a398f7e872a9     8 days
ago                115MB
registry.access.redhat.com/rhel7 latest       099889565575     3 weeks
ago                198MB
itsosleb:~ #
```

To list the layers of an image, you can issue the command that is shown in Example 4-34. Column one of the output of this command indicates IMAGE, which is a historical result (pre-Docker v1.10) because whenever a layer was created as a result of a commit action, Docker also created a corresponding image.

Example 4-34 List the history of an image

```

itsosleb:~ # docker image history nginx
IMAGE                CREATED              CREATED BY
SIZE                 COMMENT
a398f7e872a9        8 days ago         /bin/sh -c #(nop)  CMD ["nginx" "-g"
"daem...  0B
<missing>           8 days ago         /bin/sh -c #(nop)  STOPSIGNAL [SIGTERM]
0B
<missing>           8 days ago         /bin/sh -c #(nop)  EXPOSE 80/tcp
0B
<missing>           8 days ago         /bin/sh -c ln -sf /dev/stdout
/var/log/ngi...  22B
<missing>           8 days ago         /bin/sh -c set -x  && apt-get update  &&
a...  56.1MB
<missing>           8 days ago         /bin/sh -c #(nop)  ENV
NJS_VERSION=1.15.5...  0B
<missing>           8 days ago         /bin/sh -c #(nop)  ENV
NGINX_VERSION=1.15...  0B
<missing>           5 weeks ago        /bin/sh -c #(nop)  LABEL maintainer=NGINX
...  0B
<missing>           5 weeks ago        /bin/sh -c #(nop)  CMD ["bash"]
0B
<missing>           5 weeks ago        /bin/sh -c #(nop)  ADD
file:f5f366bce70b148...  59MB
itsosleb:~ #

```

4.2.18 Docker network

The use of this command allows you to manage Docker networks. You can use subcommands to create, inspect, list, remove, prune, connect, and disconnect networks (see Example 4-35).

Example 4-35 Output of docker network command

```

itsosleb:~ # docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
092eeb7e7874       bridge             bridge              local
9064785f0c1c       host               host                local
a2fc0a17ca28       none              null                local
itsosleb:~ #

```

4.2.19 Docker commit

The use of this command commits a container's file changes or settings into a new image. The commit command performs a snapshot of a running container; however, the container that is committed and its processes are paused by default while the image is committed. The commit operation does not include any data that is contained in volumes that are mounted inside the container (see Example 4-36).

Example 4-36 Docker commit command and its output

```
[root@itsoredd oracle]# docker commit db2_v2 db2_image
sha256:203e4e4294d877a76aba58f7654631ab943dd4f801067eda2a7a730eb033bc94
[root@itsoredd oracle]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED
db2_image	latest	203e4e4294d8	4
seconds ago	1.41GB		

Docker system prune

The use of this command remove all unused containers, networks, images (dangling and unreferenced), and optionally, volumes, as shown in Example 4-37.

Example 4-37 Docker system prune command

```
root@itsoubua:~# docker system prune
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all dangling images
  - all build cache

Are you sure you want to continue? [y/N] y
Deleted Containers:
4a5ad28292251b356a8635b4e8ba078551d993d9836f889c24416cd412d43f14
b57a30f38436afaaa23c5f04a07c2136c655084bfd0b6c5bb3a16b2603b4a139
f78301a12eb9eaae87505ed141509eaed325b16429b90c10348fc293e935e5ad

Total reclaimed space: 14B
root@itsoubua:~#
```

4.2.20 Docker inspect

The use of this command returns low-level information about the Docker container. You can check aspects of the Docker container configuration, such as IP address, the type of image, and other useful information about a specific Docker container. The use of this command paired with a search on "IP Address" to obtain one such piece of information is shown in Example 4-38.

Example 4-38 Output of docker inspect to collect the IP Address of a docker container

```
itsos1ec:~ # docker inspect mongoTestv1 | grep "IPAddress"
  "SecondaryIPAddresses": null,
  "IPAddress": "172.17.0.2",
  "IPAddress": "172.17.0.2",
itsos1ec:~ #
```

4.3 Swarm commands

In this section, we describe some Docker commands that are useful when Docker is in swarm mode.

Note: For more information about the initialization of a swarm cluster, see 3.6, “Setting up swarm mode” on page 79.

4.3.1 Get information about your cluster

You can use the `docker info` command to get information about your swarm nodes, as highlighted in Example 4-39.

Example 4-39 Obtaining Swarm information

```
itsoslea:~ # docker info
Containers: 3
  Running: 1
  Paused: 0
  Stopped: 2
Images: 2
Server Version: 17.06.2-ee-16
Storage Driver: btrfs
  Build Version: Btrfs v4.5.3+20160729
  Library Version: 101
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
  Volume: local
  Network: bridge host macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
Swarm: active
NodeID: s19ndwtonxd0ryjslar39m9v8
Is Manager: true
ClusterID: z31h3dw9kfv5ae60lpro0wq3
Managers: 2
Nodes: 2
Orchestration:
  Task History Retention Limit: 5
Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  Election Tick: 10
Dispatcher:
  Heartbeat Period: 5 seconds
CA Configuration:
  Expiry Duration: 3 months
  Force Rotate: 0
Root Rotation In Progress: false
Node Address: 9.76.61.245
Manager Addresses:
  9.76.61.245:2377
  9.76.61.246:2377
```

```

Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 6e23458c129b551d5c9871e5174f6b1b7f6d1170
runc version: 462c82662200a17ee39e74692f536067a3576a50
init version: 949e6fa
Security Options:
  apparmor
Kernel Version: 4.4.156-94.57-default
Operating System: SUSE Linux Enterprise Server 12 SP3
OSType: linux
Architecture: s390x
CPUs: 2
Total Memory: 15.63GiB
Name: itsoslea
ID: FMCX:4PPD:34K4:RXEJ:00QX:L7XE:6SVF:SCXM:SYQF:BPXJ:YNIT:H634
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Registry: https://index.docker.io/v1/
Experimental: false
Insecure Registries:
  127.0.0.0/8
Live Restore Enabled: false

WARNING: No swap limit support
WARNING: No kernel memory limit support
itsoslea:~ #

```

4.3.2 List nodes in swarm

The command that is shown in Example 4-40 is used to list all nodes in a swarm.

Example 4-40 Output of docker node command

```

itsoslea:~ # docker node ls
ID                                HOSTNAME          STATUS          AVAILABILITY
MANAGER STATUS
s19ndwtonxd0ryjslar39m9v8 *      itsoslea         Ready          Active
Leader
s2omwkehmf0qmvj5yxvu13b1y       itsosleb         Ready          Active
Reachable
itsoslea:~ #

```

4.3.3 Demote a node

The **docker node demote** command is used to demote one or more nodes from manager in the swarm, as shown in Example 4-41.

Example 4-41 Output of docker node demote command

```
# Listing current node status in swarm
itsoslea:~ # docker node ls
ID                               HOSTNAME          STATUS            AVAILABILITY
MANAGER STATUS
s19ndwtonxd0ryjslar39m9v8 *    itsoslea         Ready            Active
Leader
s2omwkehmf0qmvj5yxvu13b1y     itsosleb         Ready            Active
Reachable

# Demoting itsosleb
itsoslea:~ # docker node demote itsosleb
Manager itsosleb demoted in the swarm.

# Checking Status of nodes in swarm to confirm itsosleb was demoted.
itsoslea:~ # docker node ls
ID                               HOSTNAME          STATUS            AVAILABILITY
MANAGER STATUS
s19ndwtonxd0ryjslar39m9v8 *    itsoslea         Ready            Active
Leader
s2omwkehmf0qmvj5yxvu13b1y     itsosleb         Ready            Active
itsoslea:~ #
```

4.3.4 Promote a node

As shown in Example 4-42, the use of the **docker node promote** command promotes a worker node to manager.

Example 4-42 Output of docker node command to promote itsosleb

```
# Listing current node status in swarm
itsoslea:~ # docker node ls
ID                               HOSTNAME          STATUS            AVAILABILITY
MANAGER STATUS
s19ndwtonxd0ryjslar39m9v8 *    itsoslea         Ready            Active
Leader
s2omwkehmf0qmvj5yxvu13b1y     itsosleb         Ready            Active

# Promoting itsosleb
itsoslea:~ # docker node promote itsosleb
Node itsosleb promoted to a manager in the swarm.

# Checking Status of nodes in swarm to confirm itsosleb was promoted.
itsoslea:~ # docker node ls
ID                               HOSTNAME          STATUS            AVAILABILITY
MANAGER STATUS
s19ndwtonxd0ryjslar39m9v8 *    itsoslea         Ready            Active
Leader
```

```
s2omwkehmf0qmvj5yxvu13b1y    itsosleb    Ready    Active
Reachable
itsoslea:~ #
```

4.3.5 Changing the availability of your nodes

By default, swarm manager nodes can assign tasks to any ACTIVE node. However, system administrators or developers can update the node status from ACTIVE to DRAIN to prevent a node from receiving new tasks from the swarm manager.

When done, the manager node stops tasks that are running on the node and starts replica tasks on any other node with ACTIVE availability.

You also can pause new tasks to be assigned to a node. However, the existing tasks remain running. How to update change the status of a swarm node is shown in Example 4-43, Example 4-44, and Example 4-45 on page 120.

Example 4-43 Change availability from ACTIVE to drain

```
# Listing current node status in swarm
itsoslea:~ # docker node ls
ID                HOSTNAME          STATUS          AVAILABILITY
MANAGER STATUS
hv091mdcmcm8ancn185yvquo0    itsoslec          Ready           Active
s19ndwtonxd0ryjslar39m9v8 *  itsoslea          Ready           Active
Leader
s2omwkehmf0qmvj5yxvu13b1y    itsosleb          Ready           Active
Reachable

# Changing status of itsoslec server from ACTIVE to drain
itsoslea:~ # docker node update --availability=drain itsoslec
itsoslec

# Checking Status of nodes in swarm to confirm itsoslec is in drain status.
itsoslea:~ # docker node ls
ID                HOSTNAME          STATUS          AVAILABILITY
MANAGER STATUS
hv091mdcmcm8ancn185yvquo0    itsoslec          Ready           Drain
s19ndwtonxd0ryjslar39m9v8 *  itsoslea          Ready           Active
Leader
s2omwkehmf0qmvj5yxvu13b1y    itsosleb          Ready           Active
Reachable
itsoslea:~ #
```

Example 4-44 Change availability from drain to ACTIVE

```
# Listing current node status in swarm
itsoslea:~ # docker node ls
ID                HOSTNAME          STATUS          AVAILABILITY
MANAGER STATUS
hv091mdcmcm8ancn185yvquo0    itsoslec          Ready           Drain
s19ndwtonxd0ryjslar39m9v8 *  itsoslea          Ready           Active
Leader
```

```
s2omwkehmf0qmvj5yxvu13b1y  itsosleb      Ready      Active
Reachable
```

```
# Changing status of itsoslec server from drain to ACTIVE
itsoslea:~ # docker node update --availability=active itsoslec
itsoslec
```

```
# Checking Status of nodes in swarm to confirm itsoslec is in ACTIVE status.
```

```
itsoslea:~ # docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY
MANAGER STATUS			
hv091mdcmcm8ancn185yvquo0	itsoslec	Ready	Active
s19ndwtonxd0ryjslar39m9v8 *	itsoslea	Ready	Active
Leader			
s2omwkehmf0qmvj5yxvu13b1y	itsosleb	Ready	Active

```
Reachable
itsoslea:~ #
```

Example 4-45 Change availability from ACTIVE to pause and returning the previous status

```
# Listing current node status in swarm
```

```
itsoslea:~ # docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY
MANAGER STATUS			
hv091mdcmcm8ancn185yvquo0	itsoslec	Ready	Active
s19ndwtonxd0ryjslar39m9v8 *	itsoslea	Ready	Active
Leader			
s2omwkehmf0qmvj5yxvu13b1y	itsosleb	Ready	Active

```
Reachable
```

```
# Changing status of itsoslec server from ACTIVE to pause
itsoslea:~ # docker node update --availability=pause itsoslec
itsoslec
```

```
# Checking Status of nodes in swarm to confirm itsoslec is in PAUSE status.
```

```
itsoslea:~ # docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY
MANAGER STATUS			
hv091mdcmcm8ancn185yvquo0	itsoslec	Ready	Pause
s19ndwtonxd0ryjslar39m9v8 *	itsoslea	Ready	Active
Leader			
s2omwkehmf0qmvj5yxvu13b1y	itsosleb	Ready	Active

```
Reachable
```

```
# Changing status of itsoslec server from pause to ACTIVE
itsoslea:~ # docker node update --availability=active itsoslec
itsoslec
```

```
# Checking Status of nodes in swarm to confirm itsoslec is in ACTIVE status.
```

```
itsoslea:~ # docker node ls
```

ID	HOSTNAME	STATUS	AVAILABILITY
MANAGER STATUS			
hv091mdcmcm8ancn185yvquo0	itsoslec	Ready	Active

```
s19ndwtonxd0ryjslar39m9v8 * itsoslea Ready Active
Leader
s2omwkehmf0qmvj5yxvu13b1y itsosleb Ready Active
Reachable
itsoslea:~ #
```

Note: It is not recommended to run containers on manager nodes. Therefore, you must set its availability as DRAIN.

4.3.6 Find worker join command

Run the command that is shown in Example 4-46 on a manager node. In our lab, slesa or slesb servers were defined as manager nodes.

Example 4-46 Output of docker swarm command to determine the worker join command

```
itsoslea:~ # docker swarm join-token worker
To add a worker to this swarm, run the following command:

    docker swarm join --token
    SWMTKN-1-081948sx63y81hnpvpuqwf07acbosv4w3fuqug450w0kor2kevp-cn9oqd01ccrearp8qai6qv
    1fb 9.76.61.245:2377

itsoslea:~ #
```

4.3.7 Find manager join command

Run the command that is shown in Example 4-47 on a manager node. In our lab, slesa and slesb servers were defined as manager nodes.

Example 4-47 Output of docker swarm command to determine the manager join command

```
itsoslea:~ # docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token
    SWMTKN-1-081948sx63y81hnpvpuqwf07acbosv4w3fuqug450w0kor2kevp-d70cx02wyuaz2mkamvuulb
    vox 9.76.61.245:2377

itsoslea:~ #
```

4.3.8 Leave a swarm

If node does not need to be part of swarm, you can issue the command that is shown in Example 4-48.

Example 4-48 Output of docker swarm command to remove a node from swarm

```
itsoslec:~ # docker swarm leave
Node left the swarm.
itsoslec:~ #
```

4.4 Kubernetes commands

Many of the Docker commands that were described in this chapter are also available as Kubernetes commands. This section presents basic commands to check the status of a Kubernetes cluster and perform some basic operations.

4.4.1 Getting the status of the cluster

The use of the `kubectl get componentstatus` command shows the status of the parts of Kubernetes, as shown in Example 4-49.

Example 4-49 get componentstatus

```
[lnxadmin@itsoredc ~]$ kubectl get componentstatus
NAME                STATUS    MESSAGE           ERROR
controller-manager  Healthy   ok
scheduler           Healthy   ok
etcd-0              Healthy   {"health": "true"}
```

The use of the `kubectl get nodes` command displays which nodes are part of the cluster and their current status (see Example 4-50).

Example 4-50 get nodes

```
[lnxadmin@itsoredc ~]$ kubectl get nodes
NAME                                STATUS    ROLES    AGE   VERSION
itsoredb.cpolab.ibm.com            Ready    <none>   13m   v1.12.1
itsoredc.cpolab.ibm.com            Ready    master   10d   v1.12.1
itsoredd.cpolab.ibm.com            Ready    <none>   21h   v1.12.1
```

The use of the `kubectl get pods` command show which pods are deployed (see Example 4-51).

Example 4-51 Use of get pods command

```
[lnxadmin@itsoredc ~]$ kubectl get pods --all-namespaces=true
NAMESPACE   NAME                                                                 READY   STATUS
RESTARTS   AGE
kube-system  coredns-576cbf47c7-5vz6t                                           1/1    Running
0           10d
kube-system  coredns-576cbf47c7-dtcrp                                           1/1    Running
0           10d
kube-system  etcd-itsoredc.cpolab.ibm.com                                         1/1    Running
0           10d
kube-system  kube-apiserver-itsoredc.cpolab.ibm.com                             1/1    Running
0           10d
kube-system  kube-controller-manager-itsoredc.cpolab.ibm.com                   1/1    Running
0           10d
kube-system  kube-proxy-b78qk                                                    1/1    Running
0           10d
```

```

kube-system kube-proxy-nwvnf          1/1    Running
0          21h
kube-system kube-proxy-vq8qt          1/1    Running
0          17m
kube-system kube-router-5q2vc         1/1    Running
0          10d
kube-system kube-router-kwcnj          0/1
CrashLoopBackOff 257          21h
kube-system kube-router-v2hxp         0/1
CrashLoopBackOff 8           50m
kube-system kube-scheduler-itsoredc.cpolab.ibm.com 1/1    Running
0          10d
[lnxadmin@itsoredc ~]$

```

The use of the **kubect1 get services** command shows all the services that are running (see Example 4-52).

Example 4-52 Use of get services command

```

[lnxadmin@itsoredc ~]$ kubect1 get services
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
kubernetes    ClusterIP     10.96.0.1     <none>         443/TCP    10d
[lnxadmin@itsoredc ~]$

```

The use of the **kubect1 cluster-info** command shows the status of the kubernetes cluster (see Example 4-53).

Example 4-53 Use of cluter-info command

```

[lnxadmin@itsoredc ~]$ kubect1 cluster-info
Kubernetes master is running at https://9.76.61.241:6443
KubeDNS is running at
https://9.76.61.241:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

```

To further debug and diagnose cluster problems, use 'kubect1 cluster-info dump'.
[lnxadmin@itsoredc ~]\$

4.5 Backing up a container

Images of the containers that exist can be created to save a customized image, back it up, or copy a container to a different host. After an image is updated and customized per your business requirements or needs, you can commit your container and create an image to your repository.

Complete the following steps:

1. Confirm that your container is available, as shown in Example 4-54.

Example 4-54 Confirming containers available on your host

```
[root@itsoredd ~]# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	
CREATED	STATUS	PORTS	NAMES
de63be560f7b	registry.access.redhat.com/rhel7	"/bin/bash"	3

2. Perform a commit of your container after the configuration is completed by using the **docker commit** command, as shown in Example 4-55.

Example 4-55 Commit the Docker image

```
[root@itsoredd ~]# docker commit rh_oracle_v10 rh_oracle12c
sha256:ac5c4cb7f9e571cf0fed264af157cfca1966f5715dfff8238266163464a69f75
[root@itsoredd ~]# docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED
rh_oracle12c	latest	ac5c4cb7f9e5	9
seconds ago	690MB		

3. After the new image is created, create a container that is based on this image or save the container by using the **docker save** command and send it to a different host. Then, use the **docker load** command to load the new image to the other host (in this case, copied from server itsoredd to itsoreddc), as shown in Example 4-56.

Example 4-56 Saving Docker image into a new file, transferring, then loading

```
[root@itsoredd tmp]# docker save rh_oracle12c > /oracle/oracle12image.tar
[root@itsoredd tmp]# scp /tmp/sles12sp2.tar itsoslea.cpolab.ibm.com:/oracle
[root@itsoreddc tmp]# docker load < /oracle/oracle12image.tar
ca3c5eec1300: Loading layer [=====]
155.6MB/155.6MB
Loaded image: oracle12image
itsoredd:/oracle # docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED
oracle12image	1.0.2	2511879f104c	2 days ago
152MB			
hello-world	latest	ca91c0e8c6ca	4 weeks ago
1.97kB			

```
[root@itsoreddc tmp]# /oracle #
```

This approach that uses the **docker save** and **docker load** commands can be used to download images to assist when a server lacks internet access. If your server lacks internet access and the **docker pull** command does not work, you can use a server with internet access to download the docker image.

Then, use the **docker save** command as shown in Example 4-56 on page 124 and copy the image to the server that lacks internet access and use the **docker load** command to make the image available on that server.

Another option to back up a Docker container is the use of the **export** or **import** options on Docker, as shown in Example 4-57.

Example 4-57 Export the current image, then import it using docker commands

```
[root@itsoredd oracle]# docker ps -a
CONTAINER ID        IMAGE                                     COMMAND
CREATED            STATUS                                PORTS          NAMES
3c44208f7e7f       registry.access.redhat.com/rhe17      "/bin/bash"    5
hours ago          Up 4 hours                             rh_oracle_v11
```

```
[root@itsoredd oracle]# docker export rh_oracle_v11 | gzip > rh_oracle_v11.gz
```

```
[root@itsoredd oracle]# ls -ltr rh_oracle_v11.gz
-rw-rw-r-- 1 oracle lnxadmin 70536778 Oct 10 14:55 rh_oracle_v11.gz
[root@itsoredd oracle]#
```

```
[root@itsoredd oracle]# zcat rh_oracle_v11.gz | docker import - rh_oracle12c_v1
sha256:9b342e55a9b3061d094bec62efb01c8d1571fcfd079c978852da162e7d610549
[root@itsoredd oracle]#
```

```
[root@itsoredd oracle]# docker image ls
REPOSITORY          TAG          IMAGE ID          CREATED
SIZE
rh_oracle12c        latest      ac5c4cb7f9e5     21
minutes ago        690MB
```

Several options for backup containers are available. Check which option is best for your scenario.



Sample use cases

This chapter describes some practical cases that use Docker containers, including installation and configuration steps.

Upon completion of this chapter, you should be able to create a container by using an application on IBM Z and use orchestrations tools.

This chapter includes the following topics:

- ▶ 5.1, “Database management systems” on page 128
- ▶ 5.2, “IBM Watson Explorer Analytical Components” on page 139
- ▶ 5.3, “Docker in swarm mode” on page 148
- ▶ 5.4, “Kubernetes” on page 162
- ▶ 5.5, “Web services” on page 167
- ▶ 5.6, “Tying it all together” on page 169

5.1 Database management systems

Docker Enterprise Edition (Docker EE) is a great tool for development databases. It provides environment isolation that is ideal for development and test tasks. In this section, we provide the following cases that demonstrate database management that is running on Docker containers:

- ▶ MongoDB
- ▶ IBM Db2

In this section, we describe the practical installation and configuration steps for each database management system. Although various databases are available in the Docker images repository, you can create an image that is suited to your needs.

5.1.1 MongoDB

In this example, we tested a Docker container that uses MongoDB on an RHEL 7 image. We also created a database, added new data, and interacted with it by using Docker EE.

MongoDB is an open source documented database that provides high performance, high availability, and automatic scaling. It is available in the following editions:

- ▶ Community Edition: Open-source release of MongoDB
- ▶ Enterprise Edition: provides more administration, authentication, and monitoring features.

In our case, we installed MongoDB Enterprise Edition version 3.4.8.

Setting up MongoDB in Docker EE

MongoDB for IBM Z is supported on RHEL and CentOS 6. For more information, see the [MongoDB website](#).

A Docker image available for MongoDB. For more information, see [this website](#).

Complete the following steps to set up MongoDB on IBM Z:

1. Download the Docker image to your host server. The command that is used to perform this task and its resulting output is shown in Example 5-1.

Example 5-1 Pulling mongoDB Docker image

```
itsoslec:~ # docker pull sinenomine/mongodb-s390x
Using default tag: latest
latest: Pulling from sinenomine/mongodb-s390x
7c5b25b1702b: Pull complete
f7e2beee24dc: Pull complete
f77a8563476a: Pull complete
Digest: sha256:e2c2f0a83664b31d4148b35e6c31895fd8397efec55830823379efa227682bc2
Status: Downloaded newer image for sinenomine/mongodb-s390x:latest
```

2. After the image is downloaded, confirm that the image is available to be used on your host by running the command that is shown in Example 5-2.

Example 5-2 Checking images available on your Docker

```

itsoslec:~ # docker image ls
REPOSITORY                                TAG          IMAGE ID
CREATED          SIZE
sinenomine/mongodb-s390x                latest      c23b42b446b7    13
months ago          658MB
itsoslec:~ #

```

3. Process a new container from the MongoDB image downloaded that was from the Docker repository by using the **docker run** command, as shown in Example 5-3.

Example 5-3 Processing a new Docker container

```

itsoslec: # docker run -d -v /mongodb/data:/mongodb/data -p 27017:27017 -p 28017:28107 --name mongodb sinenomine/mongodb-s390x 01b1b48d7502e819f1e85660edd0ba0be6e90c2d62ecb821b505ec586d240b3b

```

After the new container is up and running, you can interact with the MongoDB container by using the **docker exec** command, as shown in Example 5-4.

Example 5-4 Running commands on a Docker container

```

itsoslec:/mongodb/data # docker exec -it mongodb /bin/bash
bash-4.2# ps -ef | grep mongod
root      1      0  0 14:42 ?          00:00:00 mongod --dbpath /mongodb/data
--ipv6 --httpinterface --rest --smallfiles --noprealloc --logappend --logpath
/mongodb/data/mongodb.log
root      36     31  0 14:43 pts/0    00:00:00 grep mongod

```

After opening a bash shell by using the **docker exec** command, you can interact with MongoDB database and start it on a specific port, as shown in Example 5-5.

Example 5-5 Starting MongoDB in background process on port 1523 inside the Docker container

```

bash-4.2# mongod --port 1523 &
[1] 159
bash-4.2# 2018-10-09T15:34:57.964+0000 I CONTROL [initandlisten] MongoDB starting
: pid=159 port=1523 dbpath=/data/db 64-bit host=7d31f903553b
2018-10-09T15:34:57.964+0000 I CONTROL [initandlisten] db version v3.4.8
2018-10-09T15:34:57.964+0000 I CONTROL [initandlisten] git version:
8ef456f89f63ab12941fe6b5352b20cff2522da3
2018-10-09T15:34:57.964+0000 I CONTROL [initandlisten] OpenSSL version: OpenSSL
1.0.1e-fips 11 Feb 2013
2018-10-09T15:34:57.964+0000 I CONTROL [initandlisten] allocator: tcmalloc
2018-10-09T15:34:57.964+0000 I CONTROL [initandlisten] modules: enterprise
2018-10-09T15:34:57.964+0000 I CONTROL [initandlisten] build environment:
2018-10-09T15:34:57.964+0000 I CONTROL [initandlisten] distmod: rhel72
2018-10-09T15:34:57.964+0000 I CONTROL [initandlisten] distarch: s390x
2018-10-09T15:34:57.964+0000 I CONTROL [initandlisten] target_arch: s390x
2018-10-09T15:34:57.964+0000 I CONTROL [initandlisten] options: { net: { port:
1523 } }
2018-10-09T15:34:57.964+0000 W - [initandlisten] Detected unclean shutdown
- /data/db/mongod.lock is not empty.

```

```

2018-10-09T15:34:57.969+0000 I - [initandlisten] Detected data files in
/data/db created by the 'wiredTiger' storage engine, so setting the active storage
engine to 'wiredTiger'.
2018-10-09T15:34:57.969+0000 W STORAGE [initandlisten] Recovering data from the
last clean checkpoint.
2018-10-09T15:34:57.969+0000 I STORAGE [initandlisten] wiredtiger_open config:
create,cache_size=7492M,session_max=20000,eviction=(threads_min=4,threads_max=4),c
onfig_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,com
pressor=snappy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_size
=2GB),statistics_log=(wait=0),
2018-10-09T15:34:58.191+0000 I CONTROL [initandlisten]
2018-10-09T15:34:58.191+0000 I CONTROL [initandlisten] ** WARNING: Access control
is not enabled for the database.
2018-10-09T15:34:58.191+0000 I CONTROL [initandlisten] ** Read and write
access to data and configuration is unrestricted.
2018-10-09T15:34:58.191+0000 I CONTROL [initandlisten] ** WARNING: You are
running this process as the root user, which is not recommended.
2018-10-09T15:34:58.191+0000 I CONTROL [initandlisten]
2018-10-09T15:34:58.191+0000 I CONTROL [initandlisten]
2018-10-09T15:34:58.191+0000 I CONTROL [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2018-10-09T15:34:58.191+0000 I CONTROL [initandlisten] ** We suggest
setting it to 'never'
2018-10-09T15:34:58.191+0000 I CONTROL [initandlisten]
2018-10-09T15:34:58.193+0000 I FTDC [initandlisten] Initializing full-time
diagnostic data capture with directory '/data/db/diagnostic.data'
2018-10-09T15:34:58.193+0000 I NETWORK [thread1] waiting for connections on port
12345
2018-10-09T15:34:59.012+0000 I FTDC [ftdc] Unclean full-time diagnostic data
capture shutdown detected, found interim file, some metrics may have been lost. OK

```

After the MongoDB process is up and running, you can start the Mongo shell and create a database and insert new data, as shown in Example 5-6.

Example 5-6 Starting Mongo shell and interact with the database

```

bash-4.2# mongo
MongoDB shell version v3.4.8
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.8
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  http://docs.mongodb.org/
Questions? Try the support group
  http://groups.google.com/group/mongodb-user
Server has startup warnings:
2018-10-09T15:08:51.923+0000 I CONTROL [initandlisten]
2018-10-09T15:08:51.923+0000 I CONTROL [initandlisten] ** WARNING: Access control
is not enabled for the database.
2018-10-09T15:08:51.923+0000 I CONTROL [initandlisten] ** Read and write
access to data and configuration is unrestricted.
2018-10-09T15:08:51.923+0000 I CONTROL [initandlisten] ** WARNING: You are
running this process as the root user, which is not recommended.
2018-10-09T15:08:51.923+0000 I CONTROL [initandlisten]
2018-10-09T15:08:51.923+0000 I CONTROL [initandlisten]

```



```

2018-10-09T15:08:51.923+0000 I CONTROL [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2018-10-09T15:08:51.923+0000 I CONTROL [initandlisten] **          We suggest
setting it to 'never'
2018-10-09T15:08:51.923+0000 I CONTROL [initandlisten]
MongoDB Enterprise >
MongoDB Enterprise > show dbs
admin 0.000GB
local 0.000GB
MongoDB Enterprise > use dockerDB
switched to db dockerDB
MongoDB Enterprise > db.dockerDB.insert({
... name: "Rachel",
... age: 30,
... position: "seller"
... })
WriteResult({ "nInserted" : 1 })
MongoDB Enterprise > db.dockerDB.insert({
... name: "Ross",
... age: 35,
... position: "seller"
... })
MongoDB Enterprise > db.dockerDB.find()
{ "_id" : ObjectId("5bbccc4dba53016cba1d4c26"), "name" : "Rachel", "age" : 30,
"position" : "seller" }
{ "_id" : ObjectId("5bbccc5fba53016cba1d4c27"), "name" : "Ross", "age" : 35,
"position" : "seller" }

```

After you customize your container, you can save it into a new image (including the modifications) in your Docker container that is named `mongodb`, as shown in Example 5-7.

Example 5-7 Performing docker commit to save a new image

```

itsoslec:~ # docker commit mongodb mongodb_v1
sha256:0c32d8afc4d80ba017ce5c49f5ceeeb3f0bd55c1def01ee0fa6773f5e0616eb6
itsoslec:~ # docker image ls
REPOSITORY          TAG          IMAGE ID
CREATED            SIZE
mongodb_v1        latest      0c32d8afc4d8    6
seconds ago        658MB

```

Your new image `mongodb_v1` is now saved into your image repository on the server and you can run a new container that is based on this newly created image, as shown in Example 5-8.

Example 5-8 Processing a new Docker container based on your new image

```

itsoslec:~ # docker run -d -v /mongodb/data:/data/db -p 21017:21017 -p 23017:23017
--name mongodb_c_v1 mongodb_v1
9045bf758982be3821d2400759dabf773abfb934877e6628b41c8973ccb0cd5a

```

Connecting to the MongoDB database

After the image is ready and the container is running, the database can be accessed at any time and the port on which the database is running can be specified, as shown in Example 5-9.

Example 5-9 Example of a shell opened and running a query on the mongoDB database

```
itsoslec:~ # docker exec -it mongoTestv1 mongo mongodb://127.0.0.1:27017
MongoDB shell version v3.4.8
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.8
Server has startup warnings:
2018-10-22T03:51:25.479+0000 I CONTROL [initandlisten]
2018-10-22T03:51:25.479+0000 I CONTROL [initandlisten] ** WARNING: Access control
is not enabled for the database.
2018-10-22T03:51:25.479+0000 I CONTROL [initandlisten] **          Read and write
access to data and configuration is unrestricted.
2018-10-22T03:51:25.479+0000 I CONTROL [initandlisten] ** WARNING: You are
running this process as the root user, which is not recommended.
2018-10-22T03:51:25.479+0000 I CONTROL [initandlisten]
2018-10-22T03:51:25.480+0000 I CONTROL [initandlisten]
2018-10-22T03:51:25.480+0000 I CONTROL [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2018-10-22T03:51:25.480+0000 I CONTROL [initandlisten] **          We suggest
setting it to 'never'
2018-10-22T03:51:25.480+0000 I CONTROL [initandlisten]
MongoDB Enterprise > show dbs
admin      0.000GB
dockerdb   0.000GB
local      0.000GB
MongoDB Enterprise > use dockerdb
switched to db dockerdb
MongoDB Enterprise > db.dockerdb.find({})
{ "_id" : ObjectId("5bcd4accf3c3b6ec088c64ea"), "name" : "Rachel", "age" : 30,
"position" : "seller" }
{ "_id" : ObjectId("5bcd4acdf3c3b6ec088c64eb"), "name" : "Ross", "age" : 35,
"position" : "seller" }
MongoDB Enterprise > exit
bye
itsoslec:~ #
```

5.1.2 IBM Db2 database

In this section, we describe how to configure Db2 V11 on RedHat Linux 7.5 by using Docker EE. For more information about the pre-requirements that must be met to install Db2, see [this website](#).

Db2 software can be downloaded from [this website](#).

To begin, the Red Hat Docker Image is downloaded and then, a Docker container was started, as shown in Example 5-10.

Example 5-10 Output of Docker image downloaded and then starting a docker

```
[root@itsoredg ]# docker pull registry.access.redhat.com/rhel7.5
Using default tag: latest
latest: Pulling from rhel7.5
f8cc1f42a63a: Pull complete
3d3c2dcd04a1: Pull complete
Digest: sha256:33963577a88086a7cafc5efbed42c21f16dee63d79a9ebb416a231097aa44cf2
Status: Downloaded newer image for registry.access.redhat.com/rhel7.5:latest
[root@itsoredg ]#
[root@itsoredg ]# docker container run -it --name DB2server --ipc=host
--cap-add=IPC_OWNER -v /IBM:/IBM -p 50000:50000 -p 50001:50001 -h
"db2server.itso.ibm.com" registry.access.redhat.com/rhel7.5 /bin/bash
[root@db2server /]#
```

Before installing Db2, it is necessary to create a unique user on the operating system. The default, db2inst1, was used, as shown in Example 5-11.

Example 5-11 Create db2inst1 user on OS

```
[root@db2server /]# useradd db2inst1
[root@db2server /]# passwd db2inst1
Changing password for user db2inst1.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@db2server /]#
```

After the user is created, we set up the Db2 variables for the db2inst1 user and root user, as shown in Example 5-12.

Example 5-12 Set variables

```
[root@db2server /]# DB2INST1_PASSWORD=password
[root@db2server /]#
PATH=$PATH:/SETUP/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
[root@db2server /]#
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/db2inst1/sqllib/lib64:/home/db2inst1/sqllib
/lib64/gskit:/home/db2inst1/sqllib/lib32:/opt/ibm/lib:/opt/ibm/lib64:/opt/ibm/lib/
profiled:/opt/ibm/lib64/profiled:/usr/lib
[root@db2server /]# export DB2INST1_PASSWORD PATH LD_LIBRARY_PATH
```

Several libraries must be installed so that the server is ready to perform a Db2 installation, as shown in Example 5-13.

Example 5-13 Example of the command used to install necessary libraries

```
[root@db2server /]# yum install -y pam.s390 pam.s390x ncurses-libs.s390
ncurses-libs.s390x file.s390x libaio.s390 libaio.s390x libstdc++-devel.s390
libstdc++-devel.s390x compat-libstdc++-33-3.2.3-72.e17.s390x
compat-libstdc++-33-3.2.3-72.e17.s390 net-tools hostname iputils nfs-utils less
Loaded plugins: ovl, product-id, search-disabled-repos, subscription-manager
```

To complete this use case, you must install `libxlc` (see Example 5-14), which is available at [this website](#).

Example 5-14 Example of command to install libxlc library

```
[root@db2server /]# yum localinstall
/IBM/FIXPACK/images/rhel/rpms/libxlc-1.2.0.0-151119a.s390x.rpm -y
Loaded plugins: ovl, product-id, search-disabled-repos, subscription-manager
This system is not registered with an entitlement server. You can use
subscription-manager to register.
Examining /IBM/FIXPACK/images/rhel/rpms/libxlc-1.2.0.0-151119a.s390x.rpm:
libxlc-1.2.0.0-151119a.s390x
Marking /IBM/FIXPACK/images/rhel/rpms/libxlc-1.2.0.0-151119a.s390x.rpm to be
installed
...
```

After completion, update the `ibmlib.conf` and `/etc/ld.so.conf.d/ibmlib.conf` files, as shown in Example 5-15.

Example 5-15 Output how files ibmlib.conf

```
[root@db2server /]# cat ibmlib.conf
/opt/ibm/lib
/opt/ibm/lib64
/opt/ibm/lib/profiled
/opt/ibm/lib64/profiled
/usr/lib
[root@db2server /]#
[root@db2server /]# cat /etc/ld.so.conf.d/ibmlib.conf
ldconfig
[root@db2server /]#
```

Download the Db2 installation file, which is available at [this website](#).

Copy the installation file to the server to proceed with the database installation, as shown in Example 5-16.

Example 5-16 File structure of the Db2 install files

```
[root@db2server /]# cd /IBM/DB2/server_awse_o/
[root@db2server server_awse_o]# ls -ltr
total 56
drwxr-xr-x. 4 root root 84 Oct 25 12:20 nlpack
-r-xr-xr-x. 1 root root 5616 Oct 25 12:23 db2setup
-r-xr-xr-x. 1 root root 5764 Oct 25 12:23 db2_deinstall
-r-xr-xr-x. 1 root root 5652 Oct 25 12:23 installFixPack
-r-xr-xr-x. 1 root root 5608 Oct 25 12:23 db2prereqcheck
-r-xr-xr-x. 1 root root 5849 Oct 25 12:23 db2ckupgrade
-r-xr-xr-x. 1 root root 5634 Oct 25 12:23 db2_install
-r-xr-xr-x. 1 root root 5598 Oct 25 12:23 db2ls
drwxr-xr-x. 6 root root 132 Oct 25 12:28 db2
[root@db2server server_awse_o]#
```

You are now ready to start the Db2 installation process. To start the installation, open a command prompt and run `./db2_install` (see Example 5-17).

Example 5-17 Starting the Db2 installation

```
[root@db2server server_awse_o]# ./db2_install
DBI1324W Support of the db2_install command is deprecated.

Default directory for installation of products - /opt/ibm/db2/V11.1

*****
Install into default directory (/opt/ibm/db2/V11.1) ? [yes/no]
yes

Specify one of the following keywords to install Db2 products.
  SERVER
  CONSV
  CLIENT
  RTCL

Enter "help" to redisplay product names.

Enter "quit" to exit.
*****
SERVER
Db2 installation is being initialized.

Total number of tasks to be performed: 54
Total estimated time for all tasks to be performed: 1989 second(s)

Task #1 start
Description: Checking license agreement acceptance
Estimated time 1 second(s)
Task #1 end

Task #2 start
Description: Base Client Support for installation with root privileges
Estimated time 4 second(s)
Task #2 end
...
Task #52 start
Description: Setting default global profile registry variables
Estimated time 1 second(s)
Task #52 end

Task #53 start
Description: Initializing instance list
Estimated time 5 second(s)
Task #53 end

Task #54 start
Description: Registering Db2 Update Service
Estimated time 30 second(s)
Task #54 end

Task #55 start
```

Description: Updating global profile registry
Estimated time 3 second(s)
Task #55 end

The execution completed with warnings.

For more information, see the Db2 installation log at
"/tmp/db2_install.log.26663".
[root@db2server server_awse_o]#

After the installation process is complete, you can check the logs that are found in the /tmp directory. You are now ready to create an instance (see Example 5-18).

Example 5-18 Creating an instance

```
[root@db2server server_awse_o]# /opt/ibm/db2/V11.1/instance/db2icrt -u db2inst1  
db2inst1  
DBI1446I The db2icrt command is running.
```

Db2 installation is being initialized.

Total number of tasks to be performed: 4
Total estimated time for all tasks to be performed: 309 second(s)

Task #1 start
Description: Setting default global profile registry variables
Estimated time 1 second(s)
Task #1 end

Task #2 start
Description: Initializing instance list
Estimated time 5 second(s)
Task #2 end

Task #3 start
Description: Configuring Db2 instances
Estimated time 300 second(s)
Task #3 end

Task #4 start
Description: Updating global profile registry
Estimated time 3 second(s)
Task #4 end

The execution completed successfully.

For more information, see the Db2 installation log at
"/tmp/db2icrt.log.53192".
DBI1070I Program db2icrt completed successfully.

After the instance was created, it was necessary to create a symbolic link for the library `libibmc++.so.1` on the Db2 user home to avoid errors when Db2 is started (see Example 5-19).

Example 5-19 Creating a symbolic link

```
[root@db2server lib32]# ln -s /opt/ibm/lib64/libibmc++.so.1
/opt/ibm/db2/V11.1/lib64
[root@db2server lib32]# cd /opt/ibm/db2/V11.1/lib64
[root@db2server lib64]# ls -ltr libibmc++.so.1
lrwxrwxrwx. 1 root root 29 Oct 25 15:03 libibmc++.so.1 ->
/opt/ibm/lib64/libibmc++.so.1
```

Now, the server is ready to start Db2. In this step, it is necessary to connect by using the Db2 user, `db2inst1`, and start Db2 (see Example 5-20).

Example 5-20 Connecting with `db2inst1` user and start Db2

```
[root@db2server lib64]# su - db2inst1
Last login: Thu Oct 25 14:58:51 -03 2018 on pts/1
[db2inst1@db2server ~]$ db2start
10/25/2018 15:03:35      0  0  SQL1063N  DB2START processing was successful.
SQL1063N  DB2START processing was successful.
[db2inst1@db2server ~]$
```

For this use case, we used the Db2 sample database as our test database. For this testing, we created the sample database, activated it, and connected to it, as shown in Example 5-21.

Example 5-21 Installing the sample database

```
[db2inst1@db2server ~]$ db2samp1

Creating database "SAMPLE"...
Connecting to database "SAMPLE"...
Creating tables and data in schema "DB2INST1"...
Creating tables with XML columns and XML data in schema "DB2INST1"...

'db2samp1' processing complete.

[db2inst1@db2server ~]$
[db2inst1@db2server ~]$ Db2 activate database sample
DB20000I The ACTIVATE DATABASE command completed successfully.
[db2inst1@db2server ~]$ db2 connect to sample

Database Connection Information

Database server          = DB2/LINUXZ64 11.1.0
SQL authorization ID    = DB2INST1
Local database alias    = SAMPLE

[db2inst1@db2server ~]$
```

After the sample database was up and running, we created a user that can connect to the database, and enabled this new user to be used on any remote connection.

We added user oper01 and granted the user select access to one of the tables that is available on the sample database, as shown in Example 5-22.

Example 5-22 Adding oper01 user and grant select to a table

```
[root@db2server ]# adduser oper01 -g users -s /sbin/nologin
[root@db2server ]# id oper01
uid=1001(oper01) gid=100(users) groups=100(users)
[root@db2server ]#
[root@db2server ]# passwd oper01
Changing password for user oper01.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[root@db2server ]#

[db2inst1@db2server ~]$ db2 "GRANT SELECT ON DB2INST1.DEPARTMENT TO USER oper01";
DB20000I The SQL command completed successfully.
[db2inst1@db2server ~]$
```

After user oper01 was created and granted the correct permissions, it was possible to connect to the database and check the content of the table, as shown in Example 5-23.

Example 5-23 Connecting to sample database as oper01 user

```
[db2inst1@db2server ~]$ db2 connect to sample user oper01
Enter current password for oper01:

      Database Connection Information

Database server          = DB2/LINUXZ64 11.1.0
SQL authorization ID    = OPER01
Local database alias    = SAMPLE

[db2inst1@db2server ~]$ db2 "select * from DB2INST1.DEPARTMENT";
```

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	-
B01	PLANNING	000020	A00	-
C01	INFORMATION CENTER	000030	A00	-
D01	DEVELOPMENT CENTER	-	A00	-
D11	MANUFACTURING SYSTEMS	000060	D01	-
D21	ADMINISTRATION SYSTEMS	000070	D01	-
E01	SUPPORT SERVICES	000050	A00	-
E11	OPERATIONS	000090	E01	-
E21	SOFTWARE SUPPORT	000100	E01	-
F22	BRANCH OFFICE F2	-	E01	-
G22	BRANCH OFFICE G2	-	E01	-
H22	BRANCH OFFICE H2	-	E01	-
I22	BRANCH OFFICE I2	-	E01	-
J22	BRANCH OFFICE J2	-	E01	-

```
14 record(s) selected.
[db2inst1@db2server ~]$
```

After this process was completed, the Db2 v11 database that is running on Docker EE can be used.

Summary

After completing both database management use cases, it is important to complete all prerequisites for the Docker container to ensure that the database functions are started without issues. To complete the prerequisites, follow the steps as described in the cases in this section.

The following use cases were presented in this section:

- ▶ MongoDB: We showed how to start a MongoDB image and then work with the databases inside the container. The database was started with different port and no issues were faced.
- ▶ Db2 database: We installed Db2 on a new container and created an instance and a database that functioned with no issues. A remote user was created that connected and accessed the database data.

After all of the images were configured and customized as needed, it was possible to commit the images and save them as new images that are available for future use. This ability helps to automate testing because it is easy to start a new Docker container that is based on the images that are saved, creates less rework, and is more agile and secure.

The advantage of the use of Docker containers for database management for development and test is that it provides an agile method of having an image ready with no need to configure a new server from scratch.

It also is faster and more practical with no need for many teams to be involved to have a database server ready to be deployed for a new test release, for example.

Docker containers bring many advantages because you continue using the security of the databases as needed, such as creating different users for a remote connection to prevent the use of the system administrator user to be used by application testers.

5.2 IBM Watson Explorer Analytical Components

IBM Watson Explorer Content Analytics (WEX AC) collects and analyzes structured and unstructured content in documents, email, databases, websites, and other enterprise repositories. By providing a platform for crawling and importing content, parsing and analyzing content, and creating a searchable index, WEX AC helps you perform text analytics across all data in your enterprise and make that data available for further study and searching.

Through a provided content mining interface, business analysts can interactively explore data from different facets and discover relationships and anomalies between various facet values. Through an interface that is designed for searches, enterprise users can query the index to quickly find and retrieve relevant documents from a ranked list of results.

In this exercise, we create a Docker image of WEX AC version 12.0.1 and use it to crawler contents from other docker containers we created.

WEX AC can be downloaded from IBM Passport Advantage® at [this website](#).

After we download it, we can then save it in a folder; for example: /IBM/wexfiles.

Later, we connect the WEX AC to a Db2 database. This configuration requires a JDBC library that can be downloaded at [this webpage](#).

If you installed the Db2 container, the files can also be copied from the installed Db2 directory. Copy these files to the `/IBM/wexfiles` directory. A default Db2 installation is shown in Example 5-24.

Example 5-24 Copying the JDBC files from the Db2 default installation directory

```
[root@db2server ~]# cp /opt/ibm/db2/V11.1/java/db2jcc_license_cu.jar
/IBM/wexinstall/
[root@db2server ~]# cp /opt/ibm/db2/V11.1/java/db2cc4.jar /IBM/wexinstall/
[root@db2server ~]#
```

This section demonstrated how we visualize the environments working together.

5.2.1 Preparing the image

As with the previous installations, we start with a base Red Hat Image, as shown in Figure 5-1.

```
[root@itsoredg /]# docker container run -it --name WEXAC1201 -p 8390:8390 -p
8393:8393 -v /IBM:/IBM -v /wexdata:/wexdata -h "wexac12.itso.ibm.com"
registry.access.redhat.com/rhel7.5 /bin/bash
[root@wexac12 /]#
```

Figure 5-1 Creating a basic container for Watson Explorer AC

For this basic container, we map the ports 8390 and 8393 (`-p 8390:8390 -p 8393:8393`), and mounted two temporary volumes (`-v /IBM:/IBM -v /wexdata:/wexdata`). The `/IBM` directory contains the software that was downloaded and `/wexdata` is used to store persistent data for this container. A host name (`-h "wexac12.itso.ibm.com"`) also is set.

This Red Hat image contains a minimum set of installed software; therefore, you must set up a repository to install libraries and other applications that are required. However, you might complete this step by using ISO CDs, which is the easiest way is to configure the Yum repository. The necessary software for WEX AC can be found in the Base OS and Optional packages. The following libraries (32-bit and 64-bit) are required on Linux s390x systems:

- ▶ `libstdc++33`
- ▶ `libstdc++`
- ▶ `zlib`
- ▶ `libXext`
- ▶ `libXft`
- ▶ `libXi`
- ▶ `libXp`
- ▶ `libXtst`

Figure 5-2 shows the installation of the libraries.

```
[root@wexac12 wexfiles]# yum install libstdc++-*.s390x libstdc++-*.s390
zlib.s390x zlib.s390 libXext.s390 libXext.s390x libXft.s390 libXft.s390x
libXi.s390 libXi.s390x libXp.s390 libXp.s390x libXtst.s390 libXtst.s390x
compat-libstdc++-33-3.2.3-72.e17.s390 compat-libstdc++-33-3.2.3-72.e17.s390x
net-tools hostname iputils less file -y
Loaded plugins: ovl, product-id, search-disabled-repos, subscription-manager
This system is not registered with an entitlement server. You can use
subscription-manager to register.
Package libstdc++-4.8.5-28.e17_5.1.s390x already installed and latest version
Package zlib-1.2.7-17.e17.s390x already installed and latest version
Resolving Dependencies
--> Running transaction check
---> Package compat-libstdc++-33.s390 0:3.2.3-72.e17 will be installed
--> Processing Dependency: libm.so.6(GLIBC_2.0) for package:
compat-libstdc++-33-3.2.3-72.e17.s390
.... (long output omitted)
```

Figure 5-2 Performing the installation of the libraries

Figure 5-3 shows the output, post-installation.

```
.... (long output omitted)
Installed:
  compat-libstdc++-33.s390 0:3.2.3-72.e17      compat-libstdc++-33.s390x
0:3.2.3-72.e17      hostname.s390x 0:3.13-3.e17              iputils.s390x
0:20160308-10.e17
libstdc++-devel.s390 0:4.8.5-28.e17_5.1      libstdc++-devel.s390x
0:4.8.5-28.e17_5.1      libstdc++-docs.s390x 0:4.8.5-28.e17_5.1
libstdc++-static.s390 0:4.8.5-28.e17_5.1
  libstdc++-static.s390x 0:4.8.5-28.e17_5.1 net-tools.s390x
0:2.0-0.22.20131004git.e17  zlib.s390 0:1.2.7-17.e17

Dependency Installed:
  expat.s390 0:2.1.0-10.e17_3      fontconfig.s390 0:2.10.95-11.e17
fontconfig.s390x 0:2.10.95-11.e17      fontpackages-filesystem.noarch
0:1.44-8.e17
  freetype.s390 0:2.4.11-15.e17      freetype.s390x 0:2.4.11-15.e17
glibc.s390 0:2.17-222.e17              groff-base.s390x 0:1.22.2-8.e17
  libX11.s390 0:1.6.5-1.e17          libX11.s390x 0:1.6.5-1.e17
libX11-common.noarch 0:1.6.5-1.e17          libXau.s390 0:1.0.8-2.1.e17
  libXau.s390x 0:1.0.8-2.1.e17      libXrender.s390 0:0.9.10-1.e17
libXrender.s390x 0:0.9.10-1.e17      libgcc.s390 0:4.8.5-28.e17_5.1
  libxcb.s390 0:1.12-1.e17          libxcb.s390x 0:1.12-1.e17
nss-softokn-freebl.s390 0:3.36.0-5.e17_5      stix-fonts.noarch 0:1.1.0-5.e17

Complete!
[root@wexac12 wexfiles]#
```

Figure 5-3 Output of libraries installed

Now that the libraries are installed, browse to the folder where we copied the WEX AC files and extract the .tar file by using the `tar -xvf WEX_DAE_AC_V12.0.1_ZLNX_ML.tar` command. We are now ready to start the installation.

Figure 5-4 shows that we confirm that the installation files are available on the server.

```
[root@itsoredg ~]# cd /IBM/wexfiles
[root@itsoredg wexfiles]# ls -l
total 1854392
-rw-r--r--. 1 root root 1898895360 Oct 29 02:41 WEX_DAE_AC_V12.0.1_ZLNX_ML.tar
[root@itsoredg wexfiles]# tar -xvf WEX_DAE_AC_V12.0.1_ZLNX_ML.tar
install.bin
responseFiles/
responseFiles/LinuxAdditionalServer.properties
responseFiles/LinuxMasterAllInOneServer.properties
responseFiles/LinuxMasterDistributedServer.properties
responseFiles/LinuxUpdateInstall.properties
[root@itsoredg wexfiles]# ls -l
total 3708752
-rwxr-xr--. 1 root root 1898864540 Jul 10 11:37 install.bin
drwxr-xr-x. 2 root root          174 Jul 10 11:37 responseFiles
-rw-r--r--. 1 root root 1898895360 Oct 29 02:41 WEX_DAE_AC_V12.0.1_ZLNX_ML.tar
[root@itsoredg wexfiles]#
```

Figure 5-4 Confirming that install files are available on the server

5.2.2 Installing Watson Explorer AC

Because this container has no graphical display, we perform a silent installation. For this type of installation, we need a response file. By using the vi editor, create a file that is named `master.properties` under the `responseFiles` directories. The file must include the contents that is shown in Figure 5-5.

```
[root@wexac12 wexfiles]# cat responseFiles/master.properties
LICENSE_ACCEPTED=true
SERVER_HOSTNAME=wexac12.itso.ibm.com
INSTALL_USER_NAME=esadmin
CREATE_USER=true
INSTALL_USER_PASSWORD=admin
INSTALL_USER_PASSWORD_CONF=admin
NODE_TYPE=ALL_IN_ONE
USER_INSTALL_DIR=/opt/IBM/es/
NODE_ROOT=/home/esadmin/esdata
CCL_PORT=6002
DATA_STORAGE_PORT=1527
ADMIN_HTTP_PORT=8390
SEARCH_SERVER_PORT=8394
APP_SERVER=Embedded
SEARCH_APP_HTTP_PORT=8393
[root@wexac12 wexfiles]#
```

Figure 5-5 Example of response file to be used

The installation can be started by using the following command (see Figure 5-6):

```
/install.bin -i silent -f responseFiles/master.properties command
```

```
[root@wexac12 wexfiles]# ./install.bin -i silent -f
responseFiles/master.properties
[root@wexac12 wexfiles]#
```

Figure 5-6 Starting the silent installation

The installation process take a couple of minutes to complete. If any errors are observed after WEX AC is installed, confirm that you installed all libraries and review the `/tmp/silentInstall.log` file for more information about the error.

Log in as the `esadmin` user that was created during the install process. Check whether all of the services are down (as expected) by using the `esadmin check` command (see Figure 5-7). Start the services by using the `esadmin system startall` command and then, recheck them. They should all be running.

```
[root@wexac12 wexfiles]# su - esadmin
Last login: Mon Oct 29 19:23:20 AEDT 2018 on pts/0
using esdata
[esadmin@wexac12 ~]$ esadmin check
Session ID                               Node ID  PID      State
-----
admin                                     node1    -        -
configmanager                            node1    -        -
controller                                node1    -        -
converter.node1                           node1    -        -
customcommunication                       node1    -        -
database.node1                             node1    -        -
datalistener.node1                        node1    -        -
discovery                                  node1    -        -
dsconfigurator                             node1    -        -
importer                                   node1    -        -
monitor                                    node1    -        -
nlqservice.node1                           node1    -        -
resource.node1                              node1    -        -
resourcedeployer                           node1    -        -
scheduler                                  node1    -        -
searchapp.node1                             node1    -        -
searchmanager.node1                        node1    -        -
searchserver.node1                         node1    -        -
utilities.node1                             node1    -        -
FFQC5324I ----- End of Report. Total: 19 -----
```

Figure 5-7 Checking the status of services

Before starting the system, the following tasks must be completed:

- ▶ Copy the `jdbc` files that we copied previously to the WEX AC libraries folder.
- ▶ Save a fresh copy of the installed folder to use later. This task must be done by root because the `esadmin` user does not have the rights to save to the `/wexdata` folder.
- ▶ Commit the image of this container that was created.

Figure 5-8 shows the commands we used to copy the JDBC libraries and saving the installation folder.

```
[esadmin@wexac12 ~]$
[esadmin@wexac12 ~]$ cd /IBM/wexinstall
[esadmin@wexac12 wexinstall]$
[esadmin@wexac12 wexinstall]$ cp db2jcc4.jar db2jcc_license_cu.jar
/opt/IBM/es/lib/
[esadmin@wexac12 wexinstall]$ exit
[root@wexac12 ~]# cp -prf /home/esadmin/esdata /wexdata/
[root@wexac12 ~]# ls -l /wexdata
total 0
drwxr-xr-x. 8 esadmin esadmin 88 Oct 30 06:03 esdata
[root@wexac12 ~]#
```

Figure 5-8 Copying the JDBC libraries and saving the installation folder

To commit the container image, we must run the commands on the host server (see Figure 5-9). The use of the **docker ps** command lists the running containers and the use of the **docker commit** command creates an image from the running container.

```
[root@itsoredg ~]# docker ps
CONTAINER ID          IMAGE                                COMMAND
CREATED              STATUS                            PORTS
NAMES
5d0ba61384d4         registry.access.redhat.com/rhel7.5  "/bin/bash"
About an hour ago    Up About an hour                  0.0.0.0:9390->8390/tcp,
0.0.0.0:9393->8393/tcp  WEXAC12
91cac1d70466         registry.access.redhat.com/rhel7.5  "/bin/bash"
4 days ago           Up 4 days                          0.0.0.0:50000-50001->50000-50001/tcp
DB2server
[root@itsoredg ~]#
[root@itsoredg ~]# docker commit --author "ITS0" 5d0ba61384d4 wexac1201:ver1
sha256:77f7b50c918fb56a6356faa71727814105b2d3423e47ceb77f6dfd99291429e7
[root@itsoredg ~]#
```

Figure 5-9 Committing an image

We can easily list the images available by using the **docker image ls** command, as shown in Figure 5-10.

```
[root@itsoredg ~]# docker image ls
REPOSITORY              TAG          IMAGE ID
CREATED                SIZE
wexac1201              ver1        77f7b50c918f
hours ago              4.35GB
registry.access.redhat.com/rhel7.5  latest     099889565575
weeks ago              198MB
[root@itsoredg ~]#
```

Figure 5-10 Check images available on Docker

With an image created, we can start multiple copies of WEX AC if we redirect the ports properly; for example, a second container can be easily started, as shown in Figure 5-11.

```
[root@itsoredg ~]# docker container run -dt -p 7390:8390 -p 7393:8393 -h
"wexac12.itso.ibm.com" wexac1201:ver1 /bin/bash
d38dcf053d0cb96fdd97c473bf44081ac04dbb7e38cce10bb129fe2dabf84e1d
[root@itsoredg ~]#
```

Figure 5-11 Start Docker container

Our new container now should be running. You can use the **docker ps** commands to see all of the running containers and the **docker exec** command to attach to it. Without specifying the **-name <name>** option, Docker automatically assigns a composed name to the running container; in our case, it was called `relaxed_clarke` (see Figure 5-12).

```
[root@itsoredg ~]# docker ps
CONTAINER ID          IMAGE                                COMMAND
CREATED              STATUS                            PORTS
NAMES
d38dcf053d0c         wexac1201:ver1                    "/bin/bash"        6
seconds ago         Up 5 seconds                       0.0.0.0:7390->8390/tcp,
0.0.0.0:7393->8393/tcp relaxed_clarke
5d0ba61384d4         registry.access.redhat.com/rhel7.5 "/bin/bash"
About an hour ago   Up About an hour                   0.0.0.0:9390->8390/tcp,
0.0.0.0:9393->8393/tcp WEXAC120
965d78e633fa         registry.access.redhat.com/rhel7.5 "/bin/bash"        25
hours ago           Up 25 hours                         0.0.0.0:8390->8390/tcp,
0.0.0.0:8393->8393/tcp WEXAC1201
91cac1d70466         registry.access.redhat.com/rhel7.5 "/bin/bash"        4
days ago           Up 4 days                           0.0.0.0:50000-50001->50000-50001/tcp
DB2server
[root@itsoredg ~]#
[root@itsoredg ~]# docker exec -it relaxed_clarke /bin/bash
[root@wexac12 /]# su - esadmin
[esadmin@wexac12 ~]$
```

Figure 5-12 Checking Docker container and accessing the container in interactive mode

Now we can start the system by using the **esadmin system startall** command (see Figure 5-13).

```
[esadmin@wexac12 ~]$ esadmin system startall
FFQC5302I Starting the system...
FFQC5359I The Common Communication Layer (CCL) service on wexac12.itso.ibm.com
is running. These servers are available: master,controller,crawler,search
FFQC5368I Starting the system sessions [esadmin system start].
FFQC5365I The required sessions are running.
FFQC5394I All services required to use the system are started.
[esadmin@wexac12 ~]$
```

Figure 5-13 Starting the system

Figure 5-14 shows the status of the services.

```
[esadmin@wexac12 ~]$ esadmin check
Session ID                               Node ID  PID      State
-----
admin                                     node1    39158    Started
configmanager                             node1    31336    Started
controller                                 node1    31336    Started
converter.node1                           node1    39226    Started
customcommunication                       node1    39162    Started
database.node1                             node1    39083    Started
datalistener.node1                        node1    39715    Started
discovery                                  node1    39191    Started
dsconfigurator                             node1    39172    Started
importer                                   node1    39716    Started
monitor                                    node1    31336    Started
nlqservice.node1                          node1    39174    Started
resource.node1                             node1    31336    Started
resourcedeployer                           node1    39193    Started
scheduler                                  node1    31336    Started
searchapp.node1                            node1    39316    Started
searchmanager.node1                       node1    31336    Started
searchserver.node1                         node1    39240    Started
utilities.node1                            node1    31336    Started
FFQC5324I ----- End of Report. Total: 19 -----

[esadmin@wexac12 ~]$
```

Figure 5-14 Starting the services and checking their status

After finishing the installation, we can start using WEX AC by using its web interface. The use of the **docker port <container ID>** command shows if the ports are correctly mapped from the host to the services that are running inside the container; for example, for the container WEXAC1201 (see Figure 5-15).

```
[root@itsoredg ~]# docker port WEXAC1201
8390/tcp -> 0.0.0.0:8390
8393/tcp -> 0.0.0.0:8393
[root@itsoredg ~]#
[root@itsoredg ~]# docker port relaxed_clarke
8390/tcp -> 0.0.0.0:7390
8393/tcp -> 0.0.0.0:7393
[root@itsoredg ~]#
```

Figure 5-15 Ports 8390 and 8393 mapped to the container.

With the ports correctly mapped, Docker forwards all requisitions that hit the host ITSOREDG to the services that are listening on the container. Open a browser on your desktop and enter the server address URL (the administrative port is 8390 8390).

You can use the host name or the IP address of your host server where you installed Docker; in our case, we use the host name and the following full URL:

<http://itsoredg.cpolab.ibm.com:8390/ESAdmin/>

An example of the default login page for WEX AC is shown in Figure 5-16.

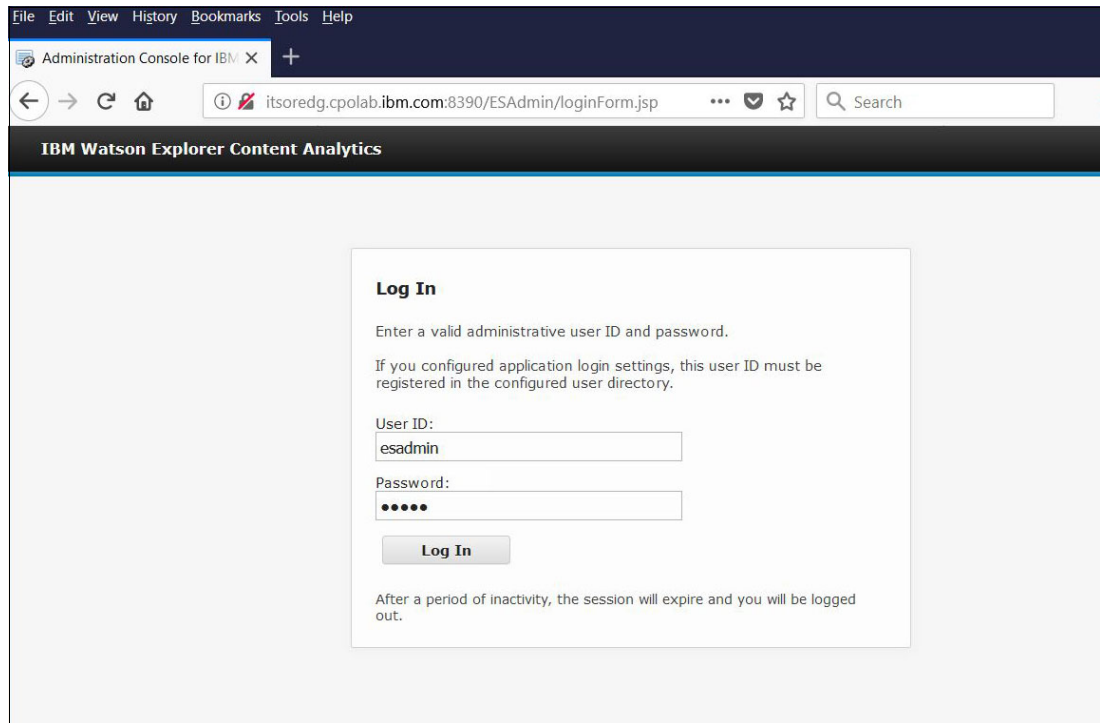


Figure 5-16 Default login page for WEX AC

The user and password were on our master.properties response file (in our example, we used admin for both). After logging in to the web interface, you see the default main page for WEX AC, as shown in Figure 5-17.

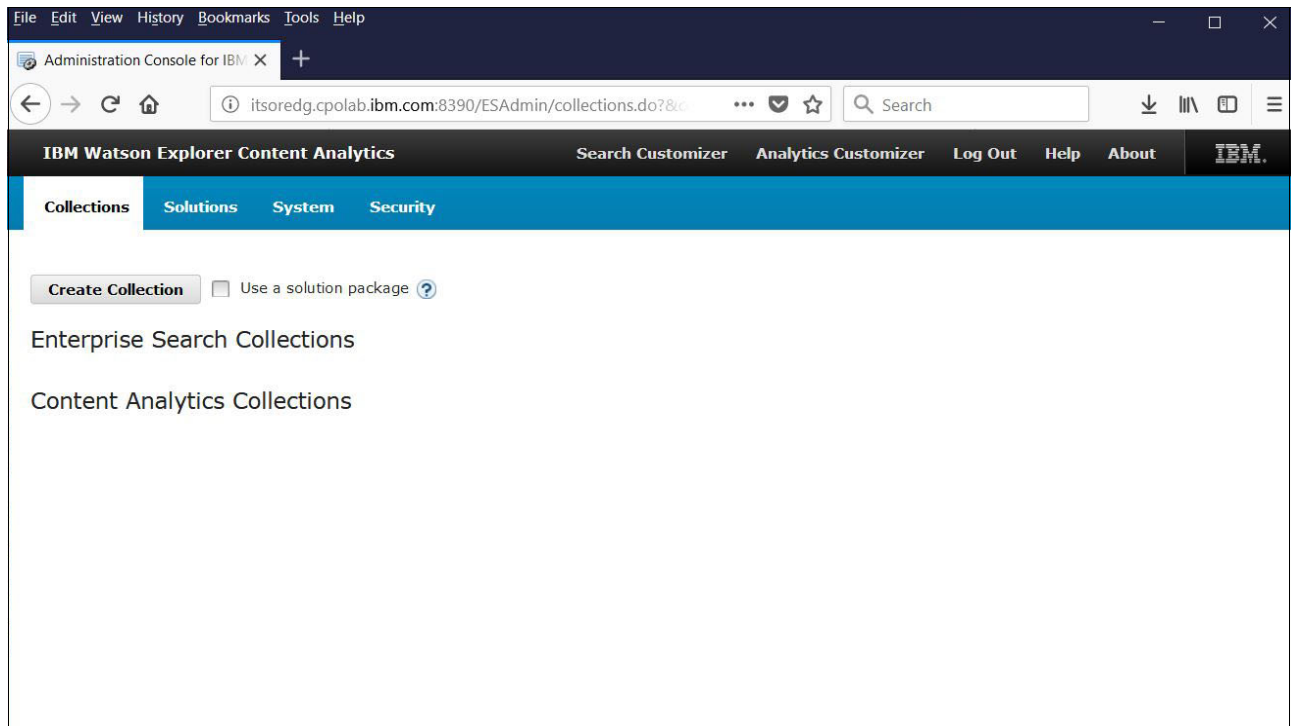


Figure 5-17 Watson Explorer Analytical Content main page

5.3 Docker in swarm mode

In this section, we configure Docker swarm mode on a SuSE Linux Enterprise Server SP3 and Redhat Linux 7.5 server to demonstrate the application portability that is achieved by Docker.

As described in Chapter 3, “Installing and deploying Docker” on page 35, Docker swarm is part of Docker Engine. Therefore, we do not have to install any other packages.

In this example, we complete the following steps:

1. Leave node from swarm.
2. Open firewall ports on manager and work nodes.
3. Update the `/etc/hosts` file.
4. Start the swarm mode.
5. Add node manager to the cluster.
6. Add worker nodes to the Manager node.
7. Create a Docker service.
8. Scale up a container.
9. Change node availability to simulate a schedule maintenance.
10. Promote a worker node.
11. Demote a manager node.
12. Run manager-only nodes.
13. Scale down a container.
14. Determine the number of manager nodes.

Our environment

The specific components of our swarm environment are listed in Table 5-1.

Table 5-1 Our swarm environment

Server name	Server IP	Swarm node role	Operating system version
itsoslea.cpolab.ibm.com	9.76.61.245	Manager	SLES12 SP3
itsosleb.cpolab.ibm.com	9.76.61.246	Manager	SLES12 SP3
itsoreda.cpolab.ibm.com	9.76.61.239	Worker	Redhat 7.5
itsoredb.cpolab.ibm.com	9.76.61.240	Worker	Redhat 7.5

The intention of having different operating system versions is to demonstrate how swarm works independently of the operating system version. However, ensure that you are using the supported version that is listed in the [Docker compatibility matrix website](#).

5.3.1 Removing node from swarm

Use the Docker swarm `leave` command remove a node from swarm because of an issue with swarm or any changes that must be done to your environment, as shown in Example 5-25.

Example 5-25 Swiping swarm configuration

```
itsoslec:~ # docker swarm leave
Node left the swarm.
itsoslec:~ #
```

Note: Optionally, you can append `--force` to force the node to leave the cluster.

5.3.2 Opening firewall ports on manager and worker nodes

To open firewall ports, access the Linux servers by way of SSH, connect as the system administrator (root), and issue the commands that are described in this section.

Environments that are running Redhat Linux

Run the commands that are shown in Example 5-26 on your manager nodes.

Example 5-26 Opening firewall ports on manager nodes

```
[root@dkmanager ~]# firewall-cmd --permanent --add-port=2376/tcp
success
[root@dkmanager ~]# firewall-cmd --permanent --add-port=2377/tcp
success
[root@dkmanager ~]# firewall-cmd --permanent --add-port=7946/tcp
success
[root@dkmanager ~]# firewall-cmd --permanent --add-port=7946/udp
success
[root@dkmanager ~]# firewall-cmd --permanent --add-port=4789/udp
success
[root@workernode2 ~]# firewall-cmd --permanent --add-port=80/tcp
success
[root@dkmanager ~]# firewall-cmd --reload
success
[root@dkmanager ~]#
```

Issue the same commands in all nodes. This list includes 2376 and 2377 tcp ports to promote a worker node to become a manager node (see Example 5-27).

Example 5-27 Opening firewall rules on the worker nodes

```
firewall-cmd --permanent --add-port=2376/tcp
firewall-cmd --permanent --add-port=2377/tcp
firewall-cmd --permanent --add-port=7946/tcp
firewall-cmd --permanent --add-port=7946/udp
firewall-cmd --permanent --add-port=4789/udp
firewall-cmd --permanent --add-port=80/tcp
firewall-cmd --reload
```

Note: If you must remove a firewall entry, run the following command:

```
firewall-cmd --remove-port=port-number/tcp --permanent
```

Environments that are running SuSE Linux

Issue the commands that are shown in Example 5-28 on your manager nodes.

Example 5-28 Opening firewall ports on manager nodes

```
itsosleb:~ # iptables -I INPUT -p tcp --dport 2376 -j ACCEPT
itsosleb:~ # iptables -I INPUT -p tcp --dport 2377 -j ACCEPT
itsosleb:~ # iptables -I INPUT -p tcp --dport 7946 -j ACCEPT
itsosleb:~ # iptables -I INPUT -p udp --dport 7946 -j ACCEPT
```

```

itsosleb:~ # iptables -I INPUT -p udp --dport 4789 -j ACCEPT
itsosleb:~ # iptables -I INPUT -p tcp --dport 80 -j ACCEPT

itsosleb:~ # iptables-save
# Generated by iptables-save v1.4.21 on Sat Oct 13 09:34:49 2018
*mangle
:PREROUTING ACCEPT [4096990:643141056]
:INPUT ACCEPT [4096770:643124588]
:FORWARD ACCEPT [220:16468]
:OUTPUT ACCEPT [3870897:381396461]
:POSTROUTING ACCEPT [3871117:381412929]
COMMIT
# Completed on Sat Oct 13 09:34:49 2018
# Generated by iptables-save v1.4.21 on Sat Oct 13 09:34:49 2018
*nat
:PREROUTING ACCEPT [1594:95855]
:INPUT ACCEPT [1594:95855]
:OUTPUT ACCEPT [68:4080]
:POSTROUTING ACCEPT [68:4080]
:DOCKER - [0:0]
:DOCKER-INGRESS - [0:0]
-A PREROUTING -m addrtype --dst-type LOCAL -j DOCKER-INGRESS
-A PREROUTING -m addrtype --dst-type LOCAL -j DOCKER
-A OUTPUT -m addrtype --dst-type LOCAL -j DOCKER-INGRESS
-A OUTPUT ! -d 127.0.0.0/8 -m addrtype --dst-type LOCAL -j DOCKER
-A POSTROUTING -o docker_gwbridge -m addrtype --src-type LOCAL -j MASQUERADE
-A POSTROUTING -s 172.19.0.0/16 ! -o docker_gwbridge -j MASQUERADE
-A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE
-A POSTROUTING -s 172.18.0.0/16 ! -o docker_gwbridge -j MASQUERADE
-A DOCKER -i docker_gwbridge -j RETURN
-A DOCKER -i docker0 -j RETURN
-A DOCKER-INGRESS -j RETURN
COMMIT
# Completed on Sat Oct 13 09:34:49 2018
# Generated by iptables-save v1.4.21 on Sat Oct 13 09:34:49 2018
*filter
:INPUT ACCEPT [532:48552]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [1905:155211]
:DOCKER - [0:0]
:DOCKER-INGRESS - [0:0]
:DOCKER-ISOLATION - [0:0]
:DOCKER-USER - [0:0]
-A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -p udp -m udp --dport 4789 -j ACCEPT
-A INPUT -p udp -m udp --dport 7946 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 7946 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 2377 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 2376 -j ACCEPT
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-INGRESS
-A FORWARD -j DOCKER-ISOLATION
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT

```

```

-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A FORWARD -o docker_gwbridge -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker_gwbridge -j DOCKER
-A FORWARD -i docker_gwbridge ! -o docker_gwbridge -j ACCEPT
-A FORWARD -i docker_gwbridge -o docker_gwbridge -j DROP
-A DOCKER-INGRESS -j RETURN
-A DOCKER-ISOLATION -i docker0 -o docker_gwbridge -j DROP
-A DOCKER-ISOLATION -i docker_gwbridge -o docker0 -j DROP
-A DOCKER-ISOLATION -j RETURN
-A DOCKER-USER -j RETURN
COMMIT
# Completed on Sat Oct 13 09:34:49 2018
itsosleb:~ #

```

Note: Issue the same commands in your work nodes.

5.3.3 Updating /etc/hosts file

Ensure that your nodes are in the /etc/hosts file (see Example 5-29). This configuration is optional if DNS servers are well-configured in your network.

Example 5-29 /etc/hosts entries

```

9.76.61.245 itsoslea.cpolab.ibm.com itsoslea
9.76.61.246 itsosleb.cpolab.ibm.com itsosleb
9.76.61.239 itsoreda.cpolab.ibm.com itsoreda
9.76.61.240 itsoredb.cpolab.ibm.com itsoredb

```

5.3.4 Initializing swarm mode

The first node in swarm is called Manager leader. This node is responsible for advertising its IP address (see Example 5-30).

Example 5-30 Initializing a swarm cluster

```

itsoslea:~ # docker swarm init --advertise-addr 9.76.61.245
Swarm initialized: current node (ln4ohpyekc43w6xm5263kxmjo) is now a manager.

```

To add a worker to this swarm, run the following command:

```

docker swarm join --token
SWMTKN-1-511hbb16vjgz0tmqlbocdz0p7kvl3kwzjy3rzjo90rd2sxkf9t-cwfk99vu4zxr112h1ts7hr
zxi 9.76.61.245:2377

```

To add a manager to this swarm, run the following command:

```

docker swarm join --token
SWMTKN-1-511hbb16vjgz0tmqlbocdz0p7kvl3kwzjy3rzjo90rd2sxkf9t-dqix0skrafzmvkasdeetpl
8g6 9.76.61.245:2377

```

5.3.5 Adding node manager to the cluster

When you start your cluster, the use of the command prints two commands that you can use to include workers and manager nodes into the cluster (see Example 5-31).

Example 5-31 Joining a manager node

```
itsosleb:~ # docker swarm join --token
SWMTKN-1-511hbb16vjgz0tmq1bocdz0p7kv13kwzjy3rzjo90rd2sxf9t-dqix0skrafzmvkasdeetp1
8g6 9.76.61.245:2377
This node joined a swarm as a manager.
itsosleb:~ #
```

5.3.6 Adding worker nodes to the manager node

To increase capacity, you can add worker nodes to process more workloads (see Example 5-32).

Example 5-32 Joining a worker node

```
[root@itsoreda ~]# docker swarm join --token
SWMTKN-1-511hbb16vjgz0tmq1bocdz0p7kv13kwzjy3rzjo90rd2sxf9t-cwfk99vu4zxr112h1ts7hr
zxi 9.76.61.245:2377
This node joined a swarm as a worker.
[root@itsoreda ~]#
```

We successfully ran the same command on the itsoredb server.

Now, check the status of the cluster by using the command that is shown in Example 5-33.

Example 5-33 Checking status of our swarm cluster

```
itsoslea:~ # docker node ls
```

ID	HOSTNAME	STATUS
9y1912puaf003ca5t194v3q3i	itsosleb	Ready Active
1n4ohpyekc43w6xm5263kxmjo *	itsoslea	Ready Active
osfw8dmvzea9xeuk5x7pr151z	itsoreda.cpolab.ibm.com	Ready Active
xa0r20fz8yt8zwj4yxocj1op	itsoredb.cpolab.ibm.com	Ready Active

5.3.7 Creating a Docker service

In this step, a web server service is started by using a nginx container. Example 5-34 shows the service status before the command is run. Then, we deploy a new service and check the status to confirm that web server service was created.

Example 5-34 Creating a Docker service

```
itsoslea:~ # docker service ls
```

ID	NAME	MODE	REPLICAS
	IMAGE	PORTS	

```
itsoslea:~ #
```

```

itsoslea:~ # docker service create --name webserver -p 80:80 nginx:latest
1iwb1fyykn6g5bwbxgdwbk3uv
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.

```

```

itsoslea:~ # docker service ls
ID                NAME                MODE                REPLICAS
IMAGE            PORTS
1iwb1fyykn6g     webserver           replicated          1/1
nginx:latest     *:80->80/tcp
itsoslea:~ #

```

You can get more information about the web server service by running the command that is shown in Example 5-35 (the nginx container was started in itsosleb server).

Example 5-35 Checking web server service

```

itsoslea:~ # docker service ps webserver
ID                NAME                IMAGE                NODE
DESIRED STATE    CURRENT STATE      ERROR                PORTS
48nt5fpvbvr1    webserver.1        nginx:latest        itsosleb
Running          Running 2 minutes ago
itsoslea:~ #

```

Now, we stopped the container on itsosleb server to see the behavior when someone takes down a container that is managed by swarm (see Example 5-36).

Example 5-36 Stopping nginx container on itsosleb server

```

itsosleb:~ # docker ps
CONTAINER ID    IMAGE                COMMAND                CREATED
STATUS          PORTS                NAMES
4f5a955f2c8e   nginx:latest        "nginx -g 'daemon ..." 2 minutes ago
Up 2 minutes    80/tcp              webserver.1.48nt5fpvbvr14ax8xmp5yao64

itsosleb:~ # docker stop 4f5a955f2c8e
4f5a955f2c8e

itsosleb:~ # docker ps
CONTAINER ID    IMAGE                COMMAND                CREATED
STATUS          PORTS                NAMES
itsosleb:~ #

```

Swarm identified that container nginx was down on itsosleb server and automatically started a new container on another cluster. In our example, it started the new container in itsoslea server, as shown in Example 5-37.

Example 5-37 Checking status of web server service after container stops

```

itsoslea:~ # docker service ps webserver
ID                NAME                IMAGE                NODE                ERROR                PORTS
DESIRED STATE    CURRENT STATE      ERROR                PORTS
hcbgnselmc15     webserver.1        nginx:latest        itsoslea
Running          Starting less than a second ago
48nt5fpvbvr1    \_ webserver.1    nginx:latest        itsosleb
Shutdown        Complete 5 seconds ago
itsoslea:~ #

```

5.3.8 Scaling a container up

We scaled up the IBM WebSphere® from 1 to 4 instances, as shown in Example 5-38.

Example 5-38 Scaling up your web server service

```

itsoslea:~ # docker service scale webserver=4
webserver scaled to 4

itsoslea:~ # docker service ps webserver
ID                NAME                IMAGE                NODE                ERROR                PORTS
DESIRED STATE    CURRENT STATE      ERROR                PORTS
jk6m1166lm4b     webserver.1        nginx:latest        itsoslea
Running          Running about a minute ago
hcbgnselmc15     \_ webserver.1    nginx:latest        itsoslea
Shutdown        Complete about a minute ago
48nt5fpvbvr1    \_ webserver.1    nginx:latest        itsosleb
Shutdown        Complete 3 minutes ago
mts75cqmq8r      webserver.2        nginx:latest        itsosleb
Running          Running 13 seconds ago
fgfvlp58y5uu     webserver.3        nginx:latest
itsoreda.cpolab.ibm.com Running          Running 7 seconds ago
rldaefeicqke     webserver.4        nginx:latest
itsoredb.cpolab.ibm.com Running          Running 5 seconds ago
itsoslea:~ #

itsoslea:~ # docker service ls
ID                NAME                MODE                REPLICAS
IMAGE            PORTS
liwb1fyykn6g     webserver          replicated          4/4
nginx:latest     *:80->80/tcp
itsoslea:~ #

```

IBM Z platform can support up to 2 million Docker containers on a single machine. This support maintains the highest levels of security and performance and does not require application server farms.

We now scale this web server service from 4 to 100, as shown in Example 5-39.

Example 5-39 Scaling web server service from 4 to 100

```
itsoslea:~ # docker service scale webserver=100
webserver scaled to 100

itsoslea:~ # docker service ls
ID                NAME                MODE                REPLICAS
IMAGE            PORTS
liwb1fyykn6g     webserver           replicated          25/100
nginx:latest     *:80->80/tcp

itsoslea:~ # docker service ls
ID                NAME                MODE                REPLICAS
IMAGE            PORTS
liwb1fyykn6g     webserver           replicated          54/100
nginx:latest     *:80->80/tcp

itsoslea:~ # docker service ls
ID                NAME                MODE                REPLICAS
IMAGE            PORTS
liwb1fyykn6g     webserver           replicated          78/100
nginx:latest     *:80->80/tcp

itsoslea:~ # docker service ls
ID                NAME                MODE                REPLICAS
IMAGE            PORTS
liwb1fyykn6g     webserver           replicated          100/100
nginx:latest     *:80->80/tcp
itsoslea:~ #
```

5.3.9 Changing node availability to simulate a schedule maintenance

Setting node availability from active to drain is often used when you want to maintain your Docker host. When a node has availability equal to drain, all containers are stopped on this node and started in the other nodes that are members of the cluster.

The status of itsoredb server changed to drain (see Example 5-40).

Example 5-40 Putting server in DRAIN mode

```
itsoslea:~ # docker node update --availability drain itsoredb.cpolab.ibm.com
itsoredb.cpolab.ibm.com

itsoslea:~ # docker node inspect --pretty itsoredb.cpolab.ibm.com
ID:          xa0r20fz8yt8zvj4yxoecl1op
Hostname:    itsoredb.cpolab.ibm.com
Joined at:   2018-10-13 13:48:58.453063359 +0000 utc
Status:
  State: Ready
  Availability: Drain
  Address:9.76.61.240
Platform:
  Operating System:linux
```

```

Architecture:s390x
Resources:
  CPUs: 2
  Memory:15.55GiB
Plugins:
  Log: awslogs, fluentd, gcplogs, gelf, journald, json-file, logentries, splunk,
  syslog
  Network:bridge, host, macvlan, null, overlay
  Volume:local
  metricscollector:docker/telemetry:1.0.0.linux-s390x-stable
Engine Version:17.06.2-ee-16
TLS Info:
  TrustRoot:
  -----BEGIN CERTIFICATE-----
  MIIBazCCARCgAwIBAgIUP6za9nAZwL+MOUZWhyq1ksJPP4wCgYIKoZIZj0EAWIw
  EzERMA8GA1UEAxMIc3dhcm0tY2EwHhcNMTgxMDEzMTMxNzAwWhcNMzgxMDA4MTMx
  NzAwWjATMREwDwYDVQQDEWhzd2FybS1jYTBZMBMGByqGSM49AgEGCCqGSM49AwEH
  AOIABL+IWRJR2rsh59F0ftc0Jd1IIRIzSZAhADG70qiD08DNjJbUQ8XSgdBW7HV
  sR+txNjJbJSRrb9gihNEKXZxPqjQjBAMA4GA1UdDwEB/wQAwIBBjAPBgNVHRMB
  Af8EBTADAQH/MBOGA1UdDgQWBBRxhYZjKs9IuYbqWgg9hjJC/p/1KDAKBggqhkJ0
  PQQDAgNjADBGAiEAveCWxhQw01zvVJeLWM664t0pdL6wCZda8P1XwK4kWw8CIQCm
  6C3Z0ayUS9HjKvKXFGF3Q/jqQFB5pU2v+NVEFfJcAQ==
  -----END CERTIFICATE-----

  Issuer Subject:MBMxETAPBgNVBAMTCHN3YXJtLWNh
  Issuer Public Key:
  MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEv4hZFEnauyHn0XR+1w412UghEjNjKCEAMbvSqIPTwM2Mk1
  tRDxdKBOFbsdWxH63E20s1s1JGtv2CKEQpdnE+g==

```

```

itsoslea:~ # docker service ls

```

ID	NAME	MODE	REPLICAS
liwb1fyykn6g	webserver	replicated	100/100
nginx:latest	*:80->80/tcp		

```

itsoslea:~ #

```

The output that is shown in Example 5-41 shows that all containers for the web server service were stopped on itsoredb.

Example 5-41 Show containers available

```

[root@itsoredb ~]# docker container ps

```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
[root@itsoredb ~]#			

To return itsoredb node back to active, run the command that is shown in Example 5-42. We also are scaling the web server service to 105 to see instances that are running on itsoredb server. This process is done because swarm does not automatically rebalance the cluster.

Example 5-42 Returning itsoredb server back to accept workloads

```

itsoslea:~ # docker node update --availability active itsoredb.cpolab.ibm.com
itsoredb.cpolab.ibm.com

itsoslea:~ # docker service scale webserver=105
webserver scaled to 105

itsoslea:~ # docker node inspect --pretty itsoredb.cpolab.ibm.com
ID:          xa0r20fz8yt8zwy4yxoecjlop
Hostname:    itsoredb.cpolab.ibm.com
Joined at:   2018-10-13 13:48:58.453063359 +0000 utc
Status:
  State: Ready
  Availability: Active
  Address:9.76.61.240
Platform:
  Operating System:linux
  Architecture:s390x
Resources:
  CPUs: 2
  Memory:15.55GiB
Plugins:
  Log: awslogs, fluentd, gcplogs, gelf, journald, json-file, logentries, splunk,
  syslog
  Network:bridge, host, macvlan, null, overlay
  Volume:local
  metricscollector:docker/telemetry:1.0.0.linux-s390x-stable
Engine Version:17.06.2-ee-16
TLS Info:
  TrustRoot:
  -----BEGIN CERTIFICATE-----
  MIIIBazCCARCgAwIBAgIUP6za9nAZwkL+MOUZWnyq1ksJPP4wCgYIKoZIZj0EAWIw
  EzERMA8GA1UEAxMIc3dhcm0tY2EwHhcNMTgxMDEzMTMxNzAwHcNMzgxMDA4MTMx
  NzAwWjATMREwDwYDVQQGEwZkd2Fybs1jYTBZMBMGByqGSM49AgEGCCqGSM49AwEH
  AOIABL+IWRRJ2rsh59F0ftc0Jd1IIRIzSZAADG70qiD08DNjJJbUQ8XSgdBW7HV
  sR+txNjrJbJSRrb9gihNEKXZxPqjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMB
  Af8EBTADAQH/MBOGA1UdDgQWBRRxhYZjKs9IuYbqWgg9hjJC/p/1KDAKBggqhkJ0
  PQQDAgNJAADBGAIeAveCWxhQw01zvVJeLWM664t0pdL6wCZda8P1XwK4kww8CIQcm
  6C3Z0ayUS9HjKvkXFGF3Q/jqQFB5pU2v+NVEfJcAQ==
  -----END CERTIFICATE-----

  Issuer Subject:MBMxETAPBgNVBAMTCHN3YXJtLWNh
  Issuer Public Key:
  MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEv4hZFEnauyHn0XR+1w412UgEjNJKCEAMbvSqIPTwM2Mk1
  tRDxdKBOFbsdWxH63E20s1s1JGtv2CKE0QpdnE+g==
itsoslea:~ #

[root@itsoredb ~]# docker container ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS              NAMES

```

```

74d0f162bee1      nginx:latest      "nginx -g 'daemon ..." 13 seconds ago
Up 10 seconds    80/tcp           webserver.105.sxj9ky7rj0cp32epy15r23w14
cf5ae0e94f3f     nginx:latest      "nginx -g 'daemon ..." 13 seconds ago
Up 10 seconds    80/tcp           webserver.102.6cy878ik9pqjfgubhvj43pkq2
026f55dd0c45     nginx:latest      "nginx -g 'daemon ..." 13 seconds ago
Up 10 seconds    80/tcp           webserver.101.tugv1prq09e4tnf24rp91tbok
c32b01a7c818     nginx:latest      "nginx -g 'daemon ..." 13 seconds ago
Up 10 seconds    80/tcp           webserver.104.kh1tf3rku2voogtunrfz6nt6l
8d7ff4ca3f82     nginx:latest      "nginx -g 'daemon ..." 13 seconds ago
Up 10 seconds    80/tcp           webserver.103.abg3sxx7c7a0ezu6ln4nix24n
[root@itsoredb ~]#

```

5.3.10 Promoting a worker node

At times, you might need to promote a worker node to be a manager when a manager node is taken down or when you need to increase the number of manager nodes in your swarm cluster. How a node is promoted is shown in Example 5-43.

Example 5-43 Promoting itsoredb node

```

itsoslea:~ # docker node promote itsoredb.cpolab.ibm.com
Node itsoredb.cpolab.ibm.com promoted to a manager in the swarm.

# Checking Status of the nodes
itsoslea:~ # docker node ls
ID                HOSTNAME                STATUS
AVAILABILITY      MANAGER STATUS
9y1912puaf003ca5t194v3q3i  itsosleb                Ready      Active
Reachable
1n4ohpyekc43w6xm5263kxmjo *  itsoslea                Ready      Active
Leader
osfw8dmvzea9xeuk5x7pr15lz  itsoreda.cpolab.ibm.com Ready      Active
xa0r20fz8yt8zvj4yxoecjlop  itsoredb.cpolab.ibm.com Ready      Active
Reachable
itsoslea:~ #

# Alternatively, you can issue command below to check Manager node status
itsoslea:~ # docker node inspect --pretty itsoredb.cpolab.ibm.com
ID:      xa0r20fz8yt8zvj4yxoecjlop
Hostname: itsoredb.cpolab.ibm.com
Joined at: 2018-10-13 13:48:58.453063359 +0000 utc
Status:
  State: Ready
  Availability: Active
  Address:9.76.61.240
Manager Status:
  Address:9.76.61.240:2377
  Raft Status:Reachable
  Leader:No
Platform:
  Operating System:linux
  Architecture:s390x
Resources:
  CPUs: 2
  Memory:15.55GiB

```

```

Plugins:
  Log: awslogs, fluentd, gcplogs, gelf, journald, json-file, logentries, splunk,
  syslog
  Network: bridge, host, macvlan, null, overlay
  Volume: local
  metricscollector: docker/telemetry:1.0.0.linux-s390x-stable
Engine Version: 17.06.2-ee-16
TLS Info:
  TrustRoot:
  -----BEGIN CERTIFICATE-----
  MIIBazCCARCgAwIBAgIU6za9nAZwL+MOUZWnyq1ksJPP4wCgYIKoZIZj0EAWIw
  EzERMA8GA1UEAxMIc3dhcm0tY2EwHhcNMTgxMDEzMTMxNzAwWhcNMzgxMDA4MTMx
  NzAwWjATMREwDwYDVQQDEwZhd2FyYS1jYTBZMBMGByqGSM49AgEGCCqGSM49AwEH
  AOIABL+IWRrRj2rsh59F0ftc0Jd1IIRIzSZAhADG70qiD08DNjJJbUQ8XSgdBW7HV
  sR+txNjrJbJSRrb9gihNEKXZxPqjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMB
  Af8EBTADAQH/MBOGA1UdDgQWBRRxhYZjKs9IuYbqWgg9hJJC/p/1KDAKBggqhkjO
  PQQDAgNjADBGAiEAveCWxhQw01zvVJeLWM664t0pdL6wCZda8P1XwK4kww8CIQCm
  6C3Z0ayUS9HjKvXFGF3Q/jqQFB5pU2v+NVEFfJcAQ==
  -----END CERTIFICATE-----

  Issuer Subject: MBMxETAPBgNVBAMTCHN3YXJtLWNh
  Issuer Public Key:
  MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEv4hZFEauyHn0XR+1w412UghEjNJKCEAMbvSqIPTwM2Mk1
  tRDxdKBOFbsdWxH63E20s1s1JGtv2CKE0QpdnE+g==

```

5.3.11 Demoting a manager node

We demote itsoredb server, as shown in Example 5-44. Therefore, this server does not process manager nodes operations.

Example 5-44 Demoting a manager node

```

itsoslea:~ # docker node demote itsoredb.cpolab.ibm.com
Manager itsoredb.cpolab.ibm.com demoted in the swarm.
itsoslea:~ #

itsoslea:~ # docker node demote itsoredb.cpolab.ibm.com
Manager itsoredb.cpolab.ibm.com demoted in the swarm.

itsoslea:~ # docker node ls
ID                HOSTNAME                STATUS
AVAILABILITY      MANAGER STATUS
9y1912puafoo3ca5t194v3q3i  itsosleb                Ready      Active
Reachable
1n4ohpyekc43w6xm5263kxmjo *  itsoslea                Ready      Active
Leader
osfw8dmvzea9xeuk5x7pr151z  itsoreda.cpolab.ibm.com Ready      Active
xa0r20fz8yt8zwj4yxoecljop  itsoredb.cpolab.ibm.com Ready      Active

itsoslea:~ # docker node inspect --pretty itsoredb.cpolab.ibm.com
ID:          xa0r20fz8yt8zwj4yxoecljop
Hostname:    itsoredb.cpolab.ibm.com
Joined at:   2018-10-13 13:48:58.453063359 +0000 utc
Status:
  State: Ready

```

```

Availability:      Active
Address:9.76.61.240
Platform:
  Operating System:linux
  Architecture:s390x
Resources:
  CPUs: 2
  Memory:15.55GiB
Plugins:
  Log: awslogs, fluentd, gcplogs, gelf, journald, json-file, logentries, splunk,
  syslog
  Network:bridge, host, macvlan, null, overlay
  Volume:local
  metricscollector:docker/telemetry:1.0.0.linux-s390x-stable
Engine Version:17.06.2-ee-16
TLS Info:
  TrustRoot:
  -----BEGIN CERTIFICATE-----
  MIIBazCCARCgAwIBAgIUP6za9nAZwL+MOUZWnyq1ksJPP4wCgYIKoZIZj0EAwIw
  EzERMA8GA1UEAxMIc3dhcm0tY2EwHhcNMTEzMTMxNzAwWhcNMzgxMDA4MTMx
  NzAwWjATMREwDwYDVQQDEWhzd2FybS1jYTBZMBMGByqGSM49AgEGCCqGSM49AwEH
  AOIABL+IWRJR2rsh59F0ftc0Jd1IIRIzSZAhADG70qiD08DNjJJbUQ8XSgdBW7HV
  sR+txNjrJbJSRrb9gihNEKXZxPqjQjBAMA4GA1UdDwEB/wQEAwIBBjAPBgNVHRMB
  Af8EBTADAQH/MB0GA1UdDgQWBBRxhYZjKs9IuYbqWgg9hjJC/p/1KDAKBggqhkJ0
  PQQDAgNJAADGAiEAveCWxhQw01zvVJeLWM664t0pdL6wCZda8P1XwK4kww8CIQCm
  6C3Z0ayUS9HjKvkXFGF3Q/jqQFB5pU2v+NVEFfJcAQ==
  -----END CERTIFICATE-----

  Issuer Subject:MBMxETAPBgNVBAMTCHN3YXJtLWNh
  Issuer Public Key:
  MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEv4hZFEnauyHn0XR+1w412UghEjNJKCEAMbvSqIPTwM2Mk1
  tRDxdKBOFbsdWxH63E20s1s1JGtv2CKEQpdnE+g==

```

5.3.12 Running manager-only nodes

If you are running a production swarm cluster, it is recommended that application workloads are *not* run on manager nodes (it can interfere with manager node operations). Therefore, update the node availability from active to drain. As shown in Example 5-45, we put `itsoslea` and `itsosleb` servers in drain mode.

When you drain a node, the scheduler reassigns any tasks that are running on the node to other available worker nodes in the swarm. It also prevents the scheduler from assigning tasks to the node.

Example 5-45 Docker node output and update from Active to Drain status

```

itsoslea:~ # docker node ls
ID                                HOSTNAME                                STATUS
AVAILABILITY      MANAGER STATUS
9y1912puaf003ca5t194v3q3i      itsosleb                                Ready      Active
Reachable
1n4ohpyekc43w6xm5263kxmjo *    itsoslea                                Ready      Active
Leader
osfw8dmvzea9xeuk5x7pr151z      itsoreda.cpolab.ibm.com                Ready      Active
xa0r20fz8yt8zwj4yxoecj1op      itsoredb.cpolab.ibm.com                Ready      Active

```

```

itsoslea:~ # docker node update --availability drain itsoslea
itsoslea

itsoslea:~ # docker node update --availability drain itsosleb
itsosleb

itsoslea:~ # docker node ls
ID                               HOSTNAME                STATUS
AVAILABILITY                    MANAGER STATUS
9y1912puafoo3ca5t194v3q3i      itsosleb                Ready      Drain
Reachable
1n4ohpyekc43w6xm5263kxmjo *    itsoslea                Ready      Drain
Leader
osfw8dmvzea9xeuk5x7prl5lz      itsoreda.cpolab.ibm.com Ready      Active
xa0r20fz8yt8zwy4yxoecj1op      itsoredb.cpolab.ibm.com Ready      Active
itsoslea:~ #

```

5.3.13 Scaling down a service

Use the command that is shown in Example 5-46 to scale down your service. In this example, we set web server service to have only two instances.

Example 5-46 Example of how to scale down service

```

itsoslea:~ # docker service ls
ID                               NAME                    MODE                REPLICAS
IMAGE                            PORTS
liwb1fyykn6g                    webserver              replicated           200/200
nginx:latest                     *:80->80/tcp

itsoslea:~ # docker service scale webserver=2
webserver scaled to 2

itsoslea:~ # docker service ls
ID                               NAME                    MODE                REPLICAS
IMAGE                            PORTS
liwb1fyykn6g                    webserver              replicated           2/2
nginx:latest                     *:80->80/tcp
itsoslea:~ #

```

5.3.14 Considerations regarding the number of manager nodes

It is vital to understand how swarm mode's fault-tolerance feature works to prevent problems with your cluster service. Although it is possible to run a cluster with only one manager node, it is not ideal for organizations that have high-availability policies. If the single node manager fails, the services continue to process the user requests; however, you must create a cluster to recover the manager node operations.

Docker uses a consensus algorithm that is named raft to achieve internal consistency across the entire swarm cluster and all containers that are running on it. Docker recommends implementing an odd number that is based to your high-availability requirements according to the following rules:

- ▶ A three-manager swarm tolerates a maximum loss of one manager.
- ▶ A five-manager swarm tolerates a maximum simultaneous loss of two manager nodes.
- ▶ An N manager cluster tolerates the loss of at most $(N-1)/2$ managers.

For more information about the cluster algorithm, see [this web page](#).

To ensure a quick recovery, back up the `/var/lib/docker/swarm` folder, as shown in Example 5-47.

Example 5-47 Backing up /var/lib/docker/swarm

```
itsoslea:~ # tar -czvf /tmp/swarm.bkp.tar.gz /var/lib/docker/swarm/
tar: Removing leading `/' from member names
/var/lib/docker/swarm/
/var/lib/docker/swarm/docker-state.json
/var/lib/docker/swarm/certificates/
/var/lib/docker/swarm/certificates/swarm-node.crt
/var/lib/docker/swarm/certificates/swarm-node.key
/var/lib/docker/swarm/certificates/swarm-root-ca.crt
/var/lib/docker/swarm/worker/
/var/lib/docker/swarm/worker/tasks.db
/var/lib/docker/swarm/raft/
/var/lib/docker/swarm/raft/snap-v3-encrypted/
/var/lib/docker/swarm/raft/wal-v3-encrypted/
/var/lib/docker/swarm/raft/wal-v3-encrypted/0000000000000000-0000000000000000.wal
/var/lib/docker/swarm/raft/wal-v3-encrypted/0.tmp
/var/lib/docker/swarm/state.json
itsoslea:~ #

itsoslea:~ # ls -la /tmp/swarm.bkp.tar.gz
-rw-r--r-- 1 root root 3209480 Oct 13 21:09 /tmp/swarm.bkp.tar.gz
itsoslea:~ #
```

5.4 Kubernetes

Kubernetes offers a command-line interface and a GUI (Dashboard) for performing operations on a Kubernetes cluster. Too many commands are available to be included in this book. For more information, see [this web page](#).

In this section, we describe some of the common scenarios that might be encountered.

5.4.1 Adding a worker node to a Kubernetes cluster

For more information about adding a worker node, see “Installing kubernetes on the worker nodes” on page 92.

The components kubeadm, kubectl, and kubelet must be installed on each node in the cluster, regardless if they are the manager.

After the kubeadm, kubectl, and kubelet components are installed and configuration files are in place, the node can join the cluster. A token is created by the master node at installation that is used for joining the worker nodes to the cluster.

Note: Tokens that are created by the master are temporary, and might not still be valid when you are ready to add worker nodes. Likewise, if the output from the cluster `init` command is lost, a new token can be created.

To display the available tokens, run the `kubeadm token list` command. One token is current for another 19 hours, and the original token that was created at the initialization of the cluster expired (see Example 5-48).

Example 5-48 List tokens

```
[lnxadmin@itsoredc ~]$ kubeadm token list
TOKEN          TTL          EXPIRES          USAGES
DESCRIPTION    EXTRA GROUPS
8vpy96.9wuukj4dp15w7hrv 19h          2018-10-25T15:12:21-04:00
authentication,signing <none>
system:bootstrappers:kubeadm:default-node-token
o5voil.hbg4y2eq6nbmpcwt <invalid>    2018-10-16T11:29:03-04:00
authentication,signing The default bootstrap token generated by 'kubeadm init'.
system:bootstrappers:kubeadm:default-node-token
[lnxadmin@itsoredc ~]$
```

To create a token, use the `kubeadm token create` command, as shown in Example 5-49.

Example 5-49 Creating a token

```
[lnxadmin@itsoredc ~]$ kubeadm token create
bwcgzw.8ci8ghy8h581rkqh
[lnxadmin@itsoredc ~]$ kubeadm token list
TOKEN          TTL          EXPIRES          USAGES
DESCRIPTION    EXTRA GROUPS
8vpy96.9wuukj4dp15w7hrv 19h          2018-10-25T15:12:21-04:00
authentication,signing <none>
system:bootstrappers:kubeadm:default-node-token
bwcgzw.8ci8ghy8h581rkqh 23h          2018-10-25T19:20:24-04:00
authentication,signing <none>
system:bootstrappers:kubeadm:default-node-token
o5voil.hbg4y2eq6nbmpcwt <invalid>    2018-10-16T11:29:03-04:00
authentication,signing The default bootstrap token generated by 'kubeadm init'.
system:bootstrappers:kubeadm:default-node-token
[lnxadmin@itsoredc ~]$
```

Our token is created and is valid for 1 full day.

Also, a certificate is needed for authorization. Again, this token can be easily created, as shown in Example 5-50.

Example 5-50 Create key for join command

```
[lnxadmin@itsoredc ~]$ openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa
-pubin -outform der 2>/dev/null | openssl dgst -sha256 -hex | sed 's/^.* //'
834d644caeb828f11ca1040123bfcac9692cacef203fa9735c84afadd98a4a3a
[lnxadmin@itsoredc ~]$
```

With the token and key, we can now join the worker node to the cluster, as shown in Example 5-51.

Example 5-51 Join a worker node

```
[lnxadmin@itsoredc ~]$ kubeadm join 9.76.61.241:6443 --token
bwcgzw.8ci8ghy8h581rkqh --discovery-token-ca-cert-hash
sha256:834d644caeb828f11ca1040123bfcac9692cacef203fa9735c84afadd98a4a3a
[lnxadmin@itsoredc ~]$
```

Confirm the new worker node in the cluster, as shown in Example 5-52.

Example 5-52 get nodes output

```
[lnxadmin@itsoredc ~]$ kubectl get nodes
NAME                                STATUS    ROLES    AGE   VERSION
itsoredb.cpolab.ibm.com             Ready    <none>   11s   v1.12.1
itsoredc.cpolab.ibm.com             Ready    master   9d    v1.12.1
itsoredd.cpolab.ibm.com             Ready    <none>   91m   v1.12.1
[lnxadmin@itsoredc ~]$
```

5.4.2 Pause (cordon off) a worker node

A worker node can be made so that it cannot be scheduled, which stops any new work from being dispatched to the node, but leaves running work intact. Use the **kubectl cordon** command, as shown in Example 5-53.

Example 5-53 Example of how to cordon off a work node

```
[lnxadmin@itsoredc ~]$ kubectl cordon itsoredb.cpolab.ibm.com
node/itsoredb.cpolab.ibm.com cordoned
[lnxadmin@itsoredc ~]$ kubectl get nodes
NAME                                STATUS              ROLES    AGE   VERSION
itsoredb.cpolab.ibm.com             Ready,SchedulingDisabled <none>   12m   v1.12.1
itsoredc.cpolab.ibm.com             Ready               master   9d    v1.12.1
itsoredd.cpolab.ibm.com             Ready               <none>   104m  v1.12.1
[lnxadmin@itsoredc ~]$
```

When you ready to resume dispatching work on the worker node, use the **kubectl uncordon** command, as shown in Example 5-54.

Example 5-54 Uncordon command

```
[lnxadmin@itsoredc ~]$ kubectl uncordon itsoredb.cpolab.ibm.com
node/itsoredb.cpolab.ibm.com uncordoned
[lnxadmin@itsoredc ~]$ kubectl get nodes
NAME                                STATUS    ROLES    AGE   VERSION
itsoredb.cpolab.ibm.com             Ready    <none>   13m   v1.12.1
itsoredc.cpolab.ibm.com             Ready    master   9d    v1.12.1
itsoredd.cpolab.ibm.com             Ready    <none>   104m  v1.12.1
[lnxadmin@itsoredc ~]$
```

5.4.3 Temporarily remove a worker node for maintenance

It is possible to temporarily remove a worker node so that the node can be patched, and so on, and then re-enable the node in the cluster. Use the **kubectl drain** command, as shown in Example 5-55.

Example 5-55 drain command

```
[lnxadmin@itsoredc ~]$ kubectl get nodes
NAME                                STATUS    ROLES    AGE   VERSION
itsoredb.cpolab.ibm.com            Ready    <none>   18h   v1.12.1
itsoredc.cpolab.ibm.com            Ready    master   10d   v1.12.1
itsoredd.cpolab.ibm.com            Ready    <none>   20h   v1.12.1
[lnxadmin@itsoredc ~]$ kubectl drain itsoredb.cpolab.ibm.com --delete-local-data
--ignore-daemonsets=true --force
node/itsoredb.cpolab.ibm.com cordoned
[lnxadmin@itsoredc ~]$ kubectl get nodes
NAME                                STATUS    ROLES    AGE   VERSION
itsoredb.cpolab.ibm.com            Ready,SchedulingDisabled <none>   19h   v1.12.1
itsoredc.cpolab.ibm.com            Ready    master   10d   v1.12.1
itsoredd.cpolab.ibm.com            Ready    <none>   20h   v1.12.1
[lnxadmin@itsoredc ~]$
```

The use of the **drain** command marks the node as unavailable for scheduling new work, and then evicts any running pods that are on that node. DaemonSets cannot be evicted by using the **kubectl drain** command. Specifying **--ignore-daemonsets=true** tells the command it is safe to proceed with the eviction and ignore DaemonSets.

Now, the node is no longer managed by the cluster and any tasks that were planned can be performed.

After the node is made operational again, run the **kubectl uncordon** command to reinstate the node, as shown in Example 5-56.

Example 5-56 uncordon command

```
[lnxadmin@itsoredc ~]$ kubectl uncordon itsoredb.cpolab.ibm.com
node/itsoredb.cpolab.ibm.com uncordoned
[lnxadmin@itsoredc ~]$ kubectl get nodes
NAME                                STATUS    ROLES    AGE   VERSION
itsoredb.cpolab.ibm.com            Ready    <none>   19h   v1.12.1
itsoredc.cpolab.ibm.com            Ready    master   10d   v1.12.1
itsoredd.cpolab.ibm.com            Ready    <none>   20h   v1.12.1
[lnxadmin@itsoredc ~]$
```

5.4.4 Removing a worker node

Situations occur in which you no longer need a particular node in your Kubernetes cluster. The process to remove the worker node is the same as removing a node for maintenance.

On the master node, issue the **kubectl drain** command to gracefully shut down all running pods on that node, as shown in Example 5-57 on page 166.

Example 5-57 kubectl drain command

```
[lnxadmin@itsoredc ~]$ kubectl drain itsoredb.cpolab.ibm.com --delete-local-data
--ignore-daemonsets=true --force
node/itsoredb.cpolab.ibm.com cordoned
[lnxadmin@itsoredc ~]$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
itsoredb.cpolab.ibm.com	Ready,SchedulingDisabled	<none>	19h	v1.12.1
itsoredc.cpolab.ibm.com	Ready	master	10d	v1.12.1
itsoredd.cpolab.ibm.com	Ready	<none>	20h	v1.12.1

```
[lnxadmin@itsoredc ~]$
```

When all of the pods and services on the subject node are gone, the node can be deleted from the cluster by using the **kubectl delete** command, as shown in Example 5-58.

Example 5-58 Delete command

```
[lnxadmin@itsoredc ~]$ kubectl delete node itsoredb.cpolab.ibm.com
node "itsoredb.cpolab.ibm.com" deleted
[lnxadmin@itsoredc ~]$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
itsoredc.cpolab.ibm.com	Ready	master	10d	v1.12.1
itsoredd.cpolab.ibm.com	Ready	<none>	20h	v1.12.1

```
[lnxadmin@itsoredc ~]$
```

To clean the kubernetes information of the worker node that was removed, use the **kubeadm reset** command.

Note: A node cannot be rejoined into the cluster. Use caution when issuing the **kubectl delete node** command. To add a node back to the cluster, the previous installation on the node must be cleaned up by using the **reset** command (see Example 5-59).

The **kubeadm reset** command requires root authority.

Example 5-59 reset command

```
[lnxadmin@itsoredb ~]$ sudo kubeadm reset
[sudo] password for lnxadmin:
[reset] WARNING: changes made to this host by 'kubeadm init' or 'kubeadm join'
will be reverted.
[reset] are you sure you want to proceed? [y/N]: y
[preflight] running pre-flight checks
[reset] stopping the kubelet service
[reset] unmounting mounted directories in "/var/lib/kubelet"
[reset] no etcd manifest found in "/etc/kubernetes/manifests/etcd.yaml". Assuming
external etcd
[reset] please manually reset etcd to prevent further issues
[reset] deleting contents of stateful directories: [/var/lib/kubelet
/etc/cni/net.d /var/lib/docker/shim /var/run/kubernetes]
[reset] deleting contents of config directories: [/etc/kubernetes/manifests
/etc/kubernetes/pki]
[reset] deleting files: [/etc/kubernetes/admin.conf /etc/kubernetes/kubelet.conf
/etc/kubernetes/bootstrap-kubelet.conf /etc/kubernetes/controller-manager.conf
/etc/kubernetes/scheduler.conf]
[lnxadmin@itsoredb ~]$
```

5.5 Web services

During this lab, we learn to load a popular web service container that is named nginx on Ubuntu host.

Start by checking whether the nginx container does not exist in the Docker images. The nginx container does not appear when the `docker images` command is run.

Run the nginx container by issuing the `docker run -it nginx /bin/bash` command. Wait for it to download automatically from the repository (see Example 5-60).

Example 5-60 Automatically download nginx image and start a container

```
root@itsoubub:~# docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED
STATUS        PORTS                               NAMES

root@itsoubub:~# docker images
REPOSITORY    TAG                IMAGE ID              CREATED
SIZE

root@itsoubub:~#

root@itsoubub:~# docker run -it nginx /bin/bash
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
Digest: sha256:9ad0746d8f2ea6df3a17ba89eca40b48c47066dfab55a75e08e2b70fc80d929e
Status: Downloaded newer image for nginx:latest
```

Now, create a storage volume and edit your `index.html` file in the storage folder. This process permits you to pull files inside the container (see Example 5-61).

Example 5-61 Create a storage volume on Docker

```
root@itsoubub:~# docker volume create wwwpages
wwwpages
root@itsoubub:~# vi /var/lib/docker/volumes/wwwpages/_data/index.html
```

The use of the command that is shown in Example 5-62 starts the container by using port 8088 and mapping the storage volume `wwwpages`.

Example 5-62 Start Docker Container using port 8080 and mapping the storage volume

```
root@itsoubub:~# docker run -it --name nginx_web -v wwwpages:/wwwpages -d -p
8088:80 nginx /bin/bash
67aed73e81a2acebe12f381a1d851b42b13e58928bbc4cac0e8884aba434650d
```

Inside the container, copy the `index.html` file to the `nginx html` folder and start the web service, as shown in Example 5-63.

Example 5-63 Use `attach` to interact with the container and start `nginx` service

```
root@itsoubub:~# docker attach nginx_web  
  
root@67aed73e81a2:/# cp /wwwpages/index.html /usr/share/nginx/html/  
  
root@67aed73e81a2:/# service nginx start
```

You can see that the service is running by opening the webbed in the browser and using the host IP `http://address:port`, as shown in Figure 5-18.

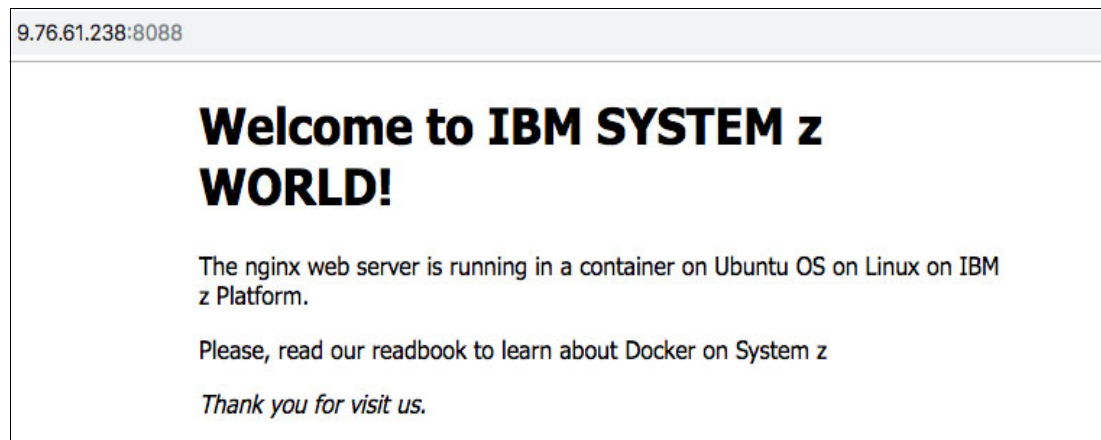


Figure 5-18 Web server is running on host port 8088 and container port 80.

The web services messages access is direct in the container shell, as shown in Example 5-64.

Example 5-64 Output of the web services messages

```
root@67aed73e81a2:/# 9.57.214.228 - - [12/Oct/2018:17:46:18 +0000] "GET /  
HTTP/1.1" 200 478 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6)  
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36" "-"  
2018/10/12 17:46:18 [error] 17#17: *1 open() "/usr/share/nginx/html/favicon.ico"  
failed (2: No such file or directory), client: 9.57.214.228, server: localhost,  
request: "GET /favicon.ico HTTP/1.1", host: "9.76.61.238:8088", referer:  
"http://9.76.61.238:8088/"  
9.57.214.228 - - [12/Oct/2018:17:46:18 +0000] "GET /favicon.ico HTTP/1.1" 404 555  
"http://9.76.61.238:8088/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_6)  
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36" "-"  
9.57.232.251 - - [12/Oct/2018:17:48:12 +0000] "GET / HTTP/1.1" 200 478 "-"  
"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/68.0.3440.106 Safari/537.36" "-"  
2018/10/12 17:48:13 [error] 17#17: *3 open() "/usr/share/nginx/html/favicon.ico"  
failed (2: No such file or directory), client: 9.57.232.251, server: localhost,  
request: "GET /favicon.ico HTTP/1.1", host: "9.76.61.238:8088", referer:  
"http://9.76.61.238:8088/"  
9.57.232.251 - - [12/Oct/2018:17:48:13 +0000] "GET /favicon.ico HTTP/1.1" 404 555  
"http://9.76.61.238:8088/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36  
(KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36" "-"
```

5.6 Tying it all together

After the environment is set up, it is time to connect all the pieces. First, we configure WEX AC to access the sample data on the Db2 container. Then, we create a small website that uses the NGINX container with the Linux manpages that also are crawled with WEX AC.

5.6.1 Connecting to the Db2 container

Start by logging in to the WEX AC interface. We set up the user ID `esadmin` with a password of `admin`. The default login page is shown in Figure 5-19.

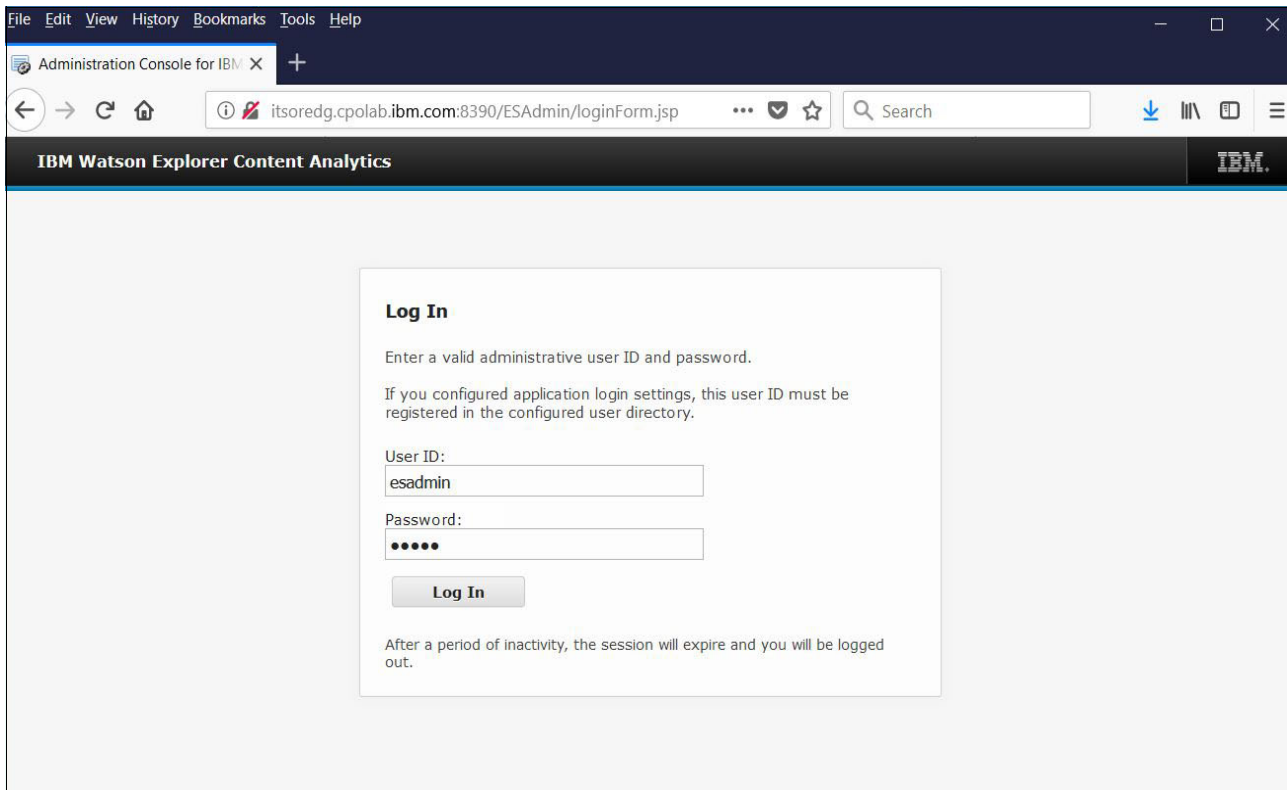


Figure 5-19 Default login page for WEX AC

After logging in to the web interface, you are presented with the default main page for WEX AC, as shown in Figure 5-20.

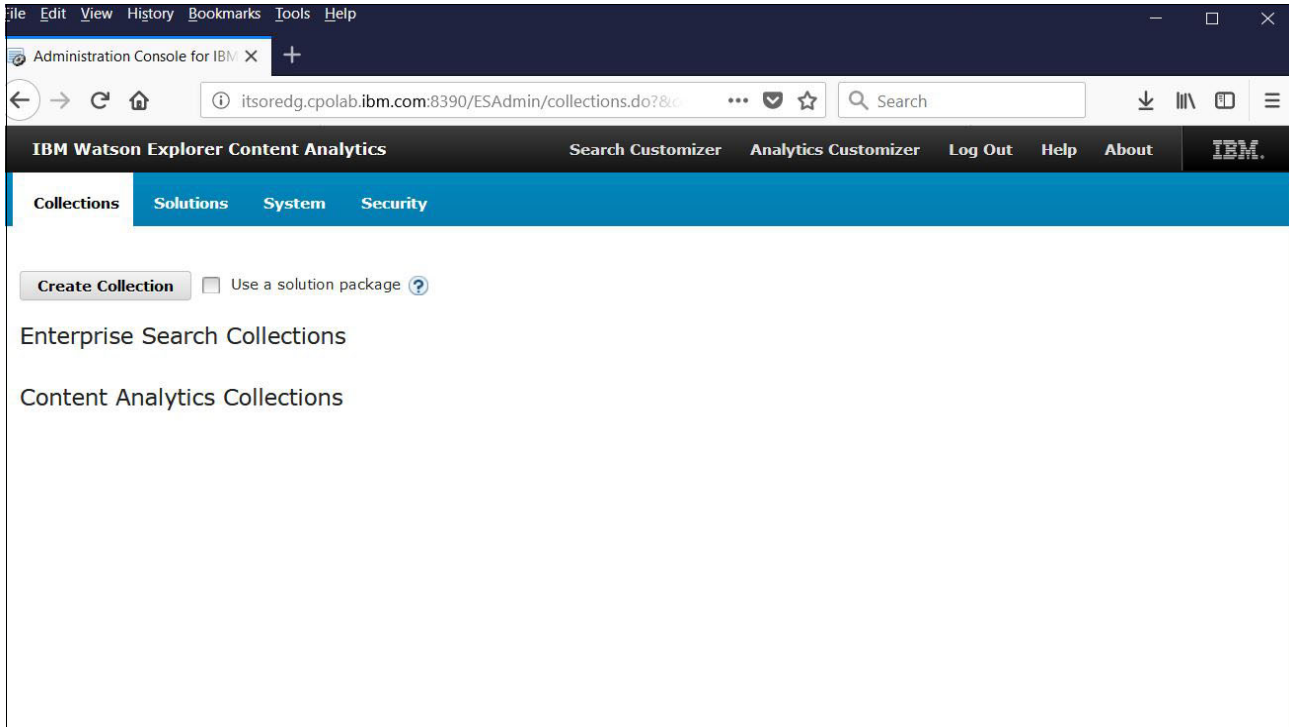


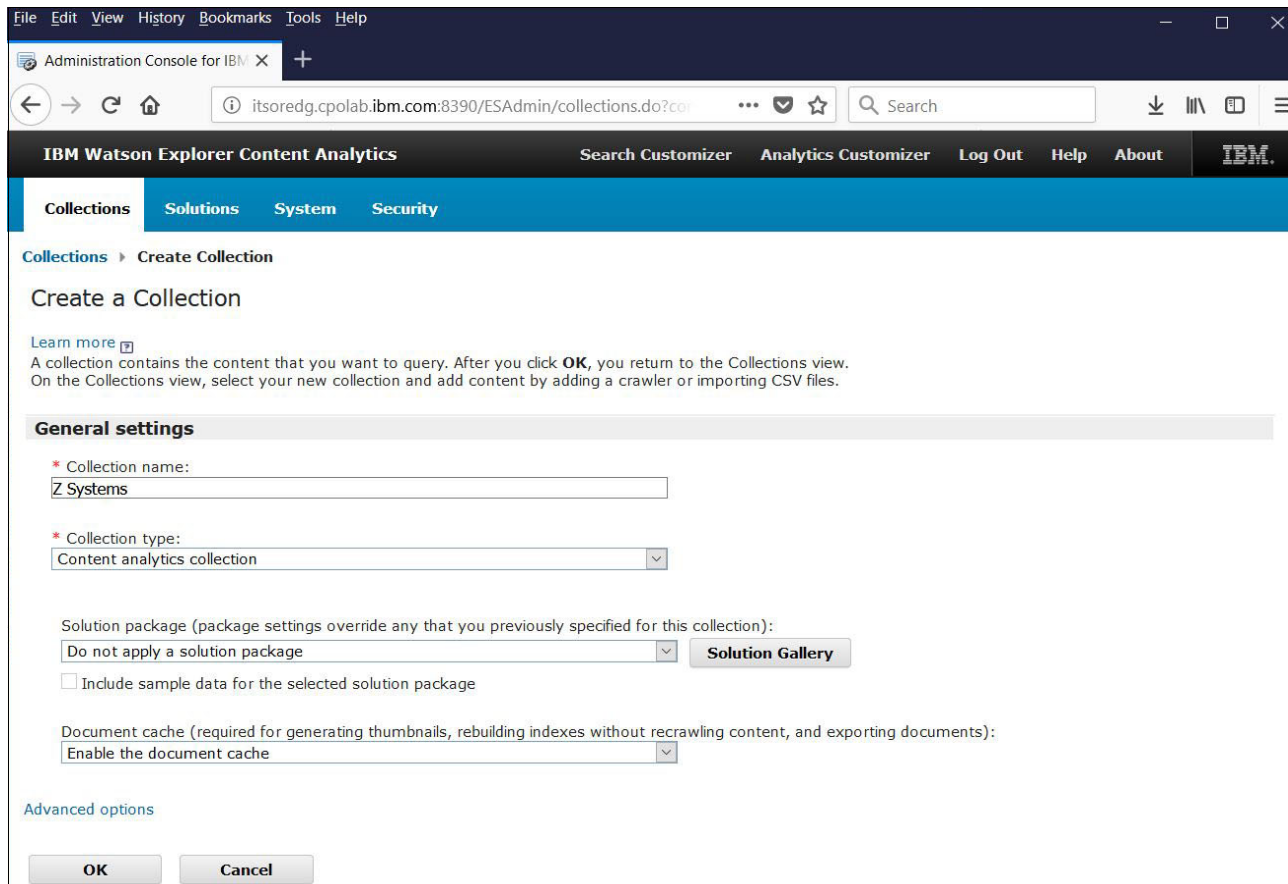
Figure 5-20 Watson Explorer Analytical Content main page

5.6.2 Creating a collection

When you start using WEX, you must first create one or more collections. A *collection* is a set of data sources and options for crawling, parsing, indexing, and searching those data sources.

Complete the following steps to create a collection:

1. On the main page, click **Create a collection**. Choose **Analytical**, and provide a name for this collection. In our example, we used “Z Systems”, as shown in Figure 5-21.



The screenshot shows a web browser window displaying the 'Administration Console for IBM' interface. The main navigation bar includes 'Search Customizer', 'Analytics Customizer', 'Log Out', 'Help', and 'About'. The 'Collections' menu is active, and the 'Create Collection' page is shown. The form is titled 'Create a Collection' and includes a 'Learn more' link and a brief description: 'A collection contains the content that you want to query. After you click OK, you return to the Collections view. On the Collections view, select your new collection and add content by adding a crawler or importing CSV files.' The 'General settings' section contains the following fields: 'Collection name' (text input with 'Z Systems'), 'Collection type' (dropdown menu with 'Content analytics collection'), 'Solution package' (dropdown menu with 'Do not apply a solution package' and a 'Solution Gallery' button), and an unchecked checkbox for 'Include sample data for the selected solution package'. The 'Document cache' section has a dropdown menu set to 'Enable the document cache'. At the bottom, there are 'Advanced options' and 'OK' and 'Cancel' buttons.

Figure 5-21 Creating a collection

2. Select **Content analytics collection** as the collection type. Several configuration options are available; for this example, the name and a collection type is enough information.
3. Click **OK** to finish and return to the main page.

On the main page, you can see that the Z Systems collection was created. A crawler tab is available on the left side of the new collection.

4. Point to the tab and click the plus sign to configure the JDBC crawler (see Figure 5-22).

Note: A crawler is a software program that retrieves documents from data sources and gathers information that can be analyzed, indexed, searched, and mined.

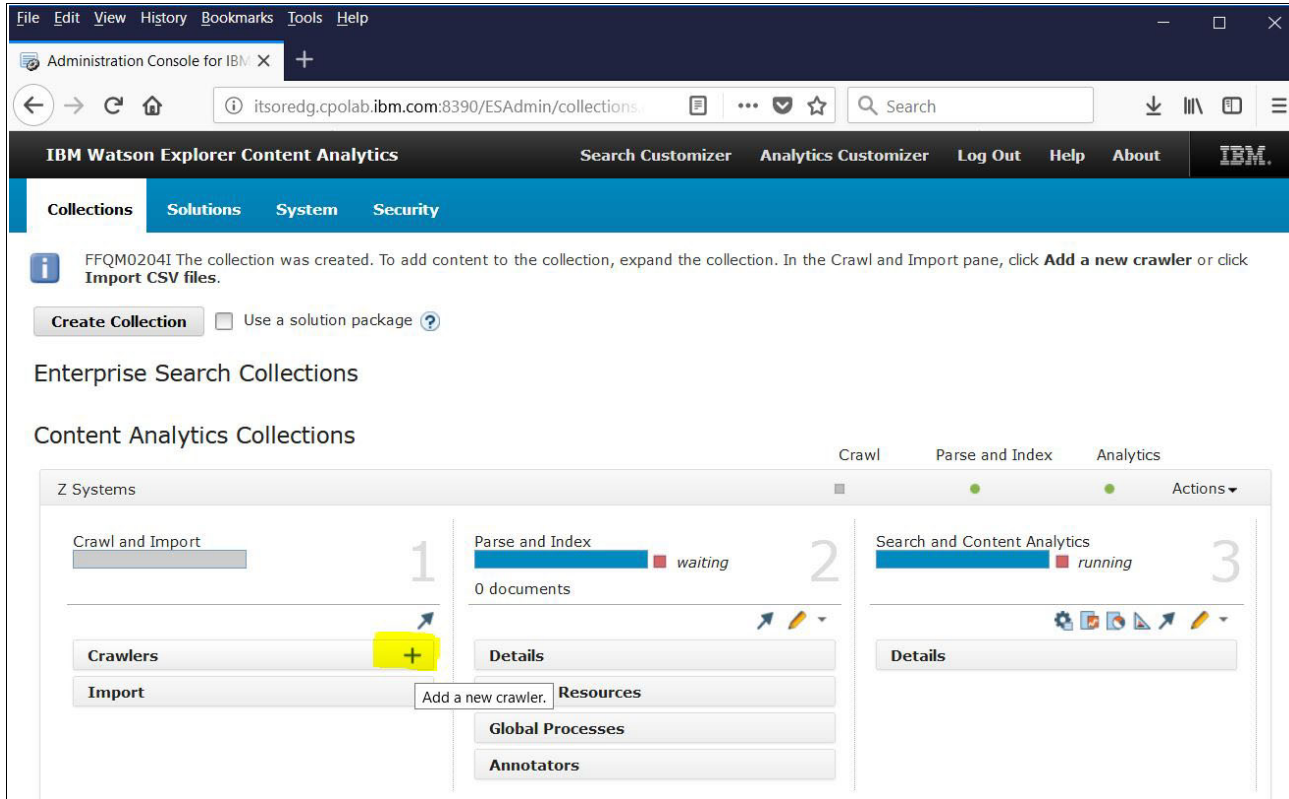


Figure 5-22 Configuring a crawler to the collection

5.6.3 Configuring a Db2 crawler

Complete the following steps to configure a Db2 crawler:

1. On the Create a Crawler page (see Figure 5-23), select **JDBC database** from the Crawler type drop-down menu. Click **Next**.

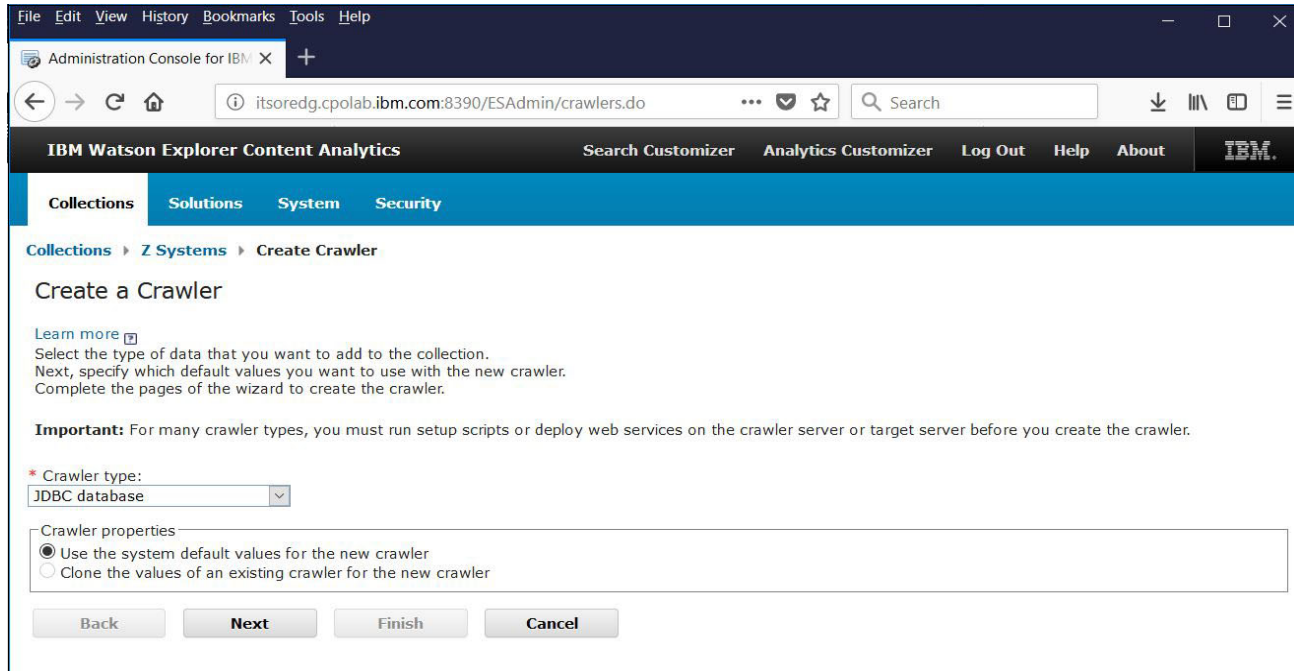


Figure 5-23 Configuring the Crawler type

2. Select the Crawler properties and name the crawler. As shown in Figure 5-24, we entered the name **JDBC docker crawler**. You can also configure other settings for this crawler by clicking **Advanced options**. In our example, the default settings are used. Click **Next**.

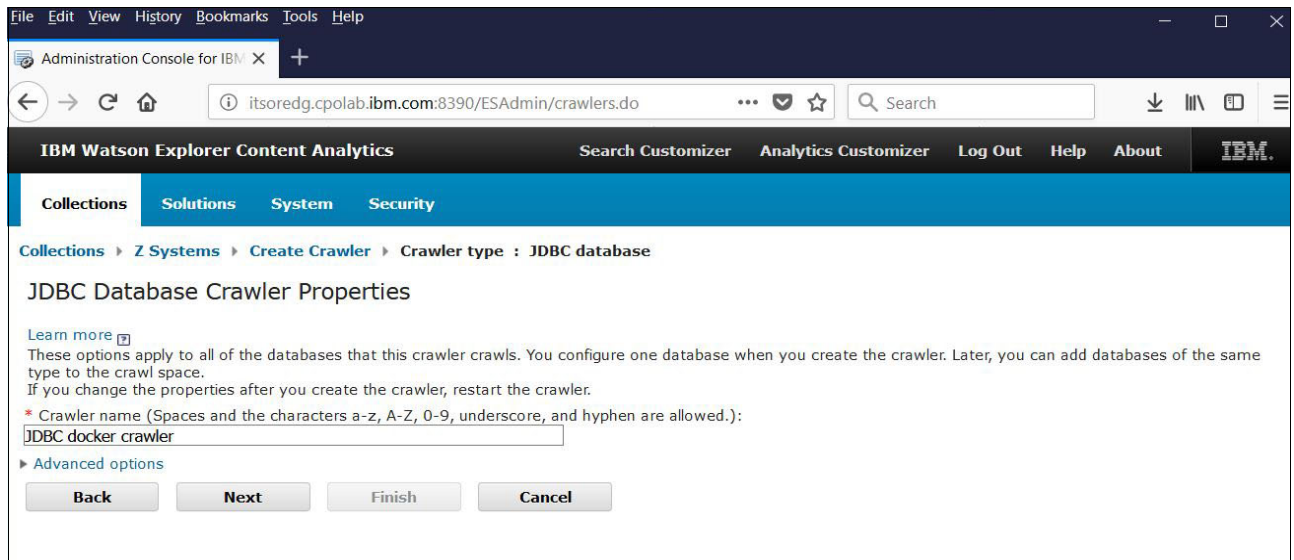


Figure 5-24 Defining the crawler properties

3. On the next page, select the JDBC Database Driver. In the drop-down menu for the JDBC database type, select **Db2**, and the page refreshes. If you copied the JDBC .jar files (as shown in Example 5-24 on page 140), the JDBC driver name and JDBC driver class path fields (as shown in Figure 5-25) are automatically completed; otherwise, you must enter the information. Click **Next**.

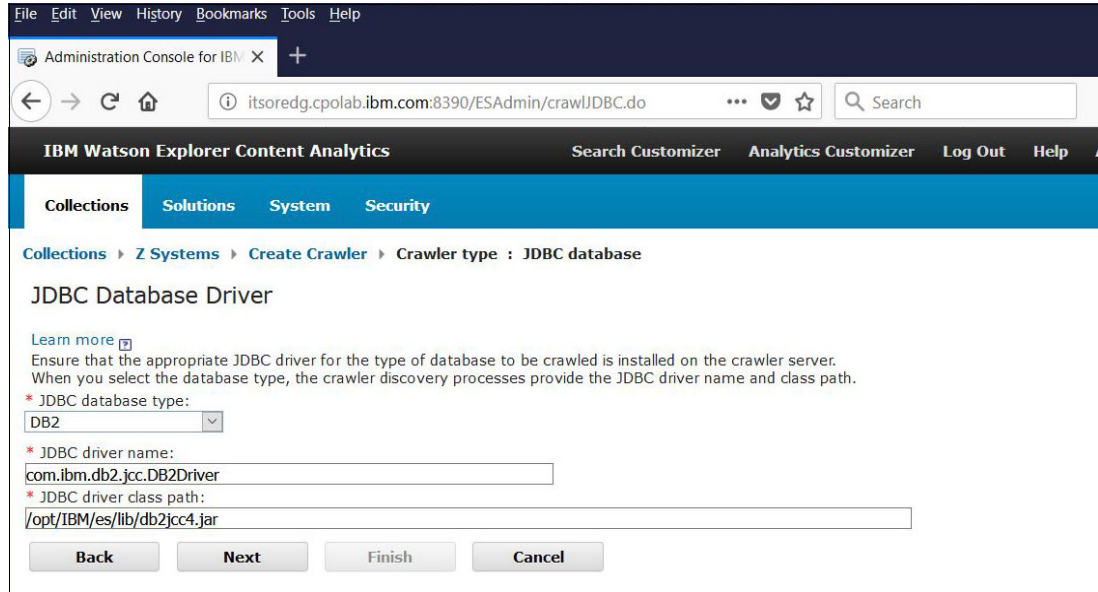


Figure 5-25 Selecting the JDBC drivers

4. As shown in Figure 5-26, enter the authorization details to access the database. The driver name and the driver path are completed. The database URL is `jdbc:db2://<iaddress>:<port>/database`. (We use the internal IP address of the container that includes the Db2 database installed.) Also, enter the user ID and password; then, click **Next**.

The screenshot shows the 'Administration Console for IBM' interface. The breadcrumb trail is 'Collections > Z Systems > Create Crawler > Crawler type : JDBC database'. The main heading is 'JDBC Database to Crawl'. Below this, there is a 'Learn more' link and a paragraph explaining the purpose of the configuration. The form contains the following fields and options:

- JDBC driver name: `com.ibm.db2.jcc.DB2Driver`
- JDBC driver class path: `/opt/IBM/es/lib/db2jcc4.jar`
- * Database URL: `jdbc:db2://itsoredg.cpolab.ibm.com:50000/sample`
- Connection credentials:
 - Reuse configured connection credentials
 - Specify new connection credentials
- User ID: `oper01`
- Password: (masked with dots)

At the bottom, there are four buttons: 'Back', 'Next', 'Finish', and 'Cancel'.

Figure 5-26 Entering the database login information

5. Complete the following steps to configure the tables that you want to crawl:
 - a. Click the magnifying glass **Search for tables**, as shown in Figure 5-27. The tables appear in the Available tables box. Select the tables that you want to crawl and click the arrow icon to move tables from Available tables to Tables to crawl; then, click **Next**.

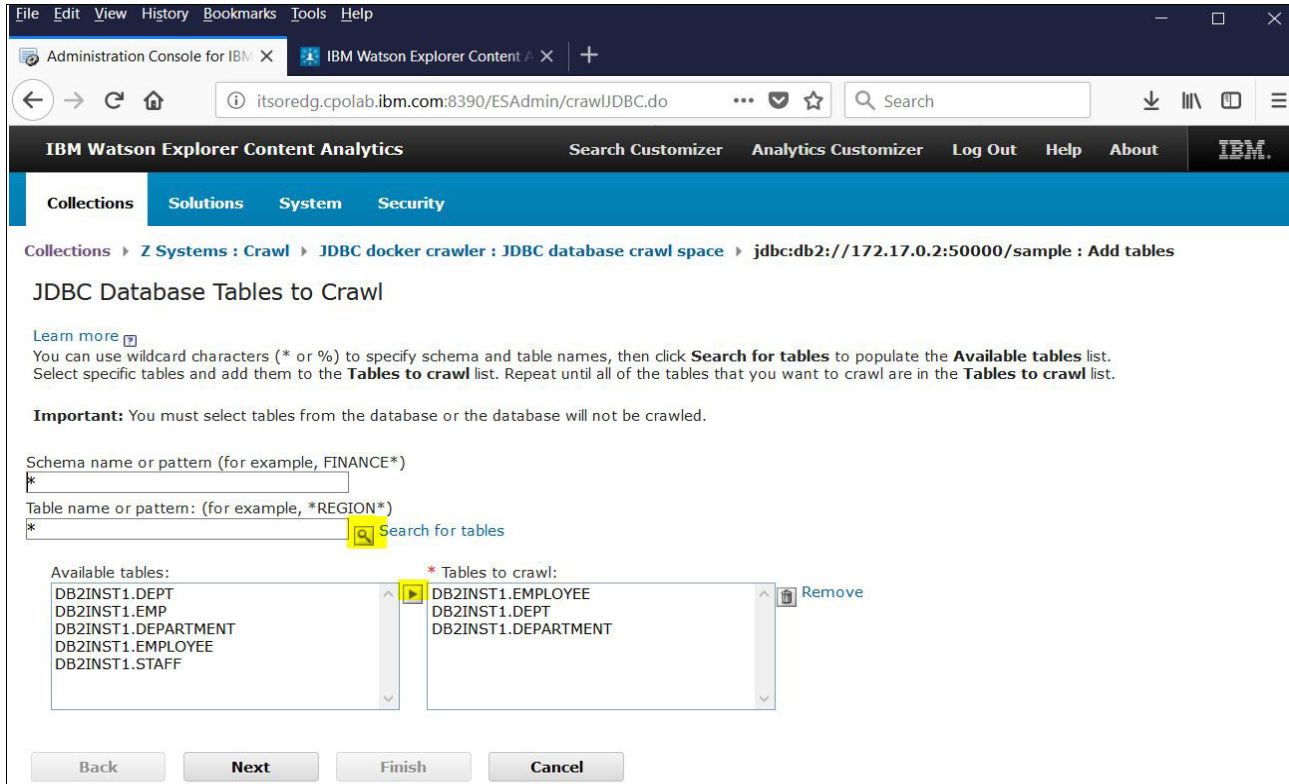


Figure 5-27 Selecting the tables to crawl

- b. For each table that is selected, you must configure how WEX AC crawls the contents. On the next page (see Figure 5-28 on page 177), select the **Generate from the column name** option (number 1 in Figure 5-28 on page 177) in the Options for autogenerated index field names section.
- c. In the Fields to crawl section (number 2 in Figure 5-28 on page 177), select the **All** option to select all column names. Then, select **Create for All, Returnable, Faceted Search, Free Text Search, and In Summary**.
- d. Select a Unique identifier (if it was not selected automatically). Leave all other options with the defaults selections and click **Next**. Repeat this process for the other tables you that you need.

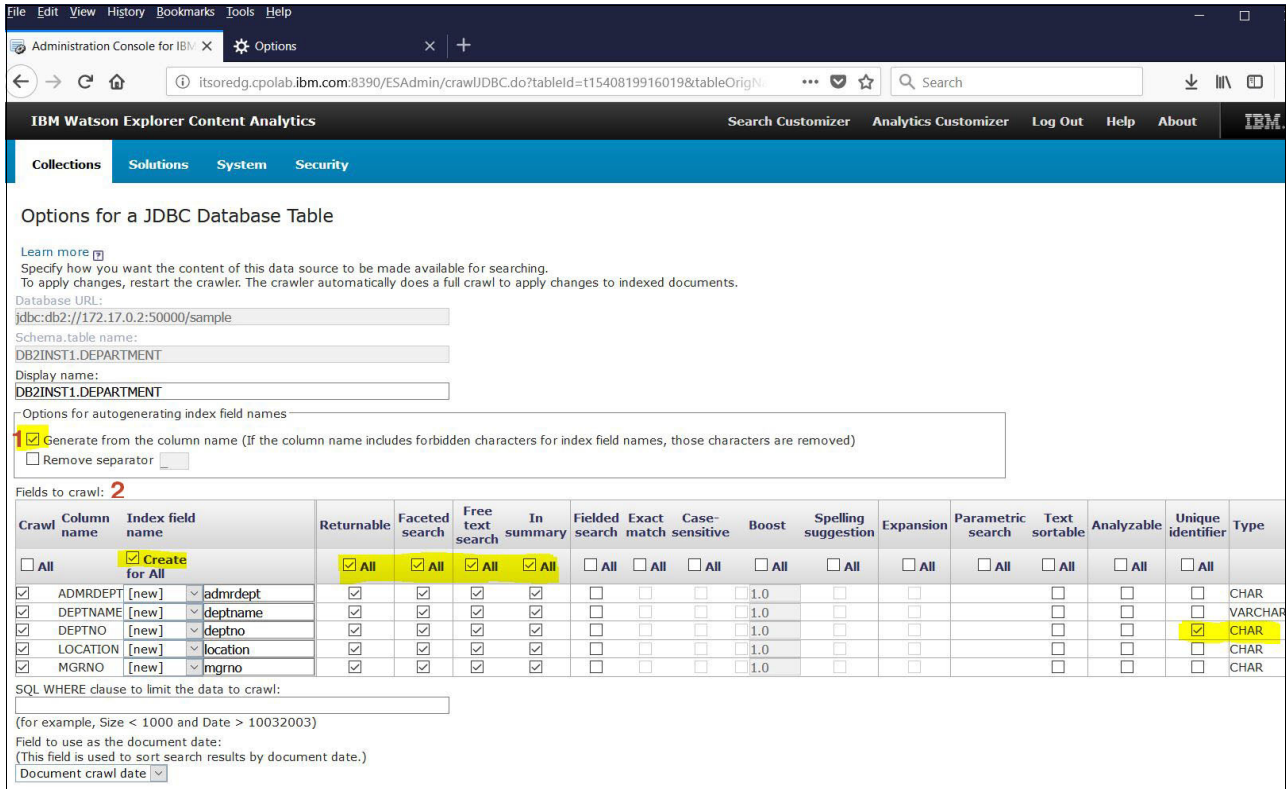


Figure 5-28 Defining the fields that will be crawled for each table

- e. After finishing the tables configuration, a pencil icon appears on the left side of each table (see Figure 5-29). Click **Finish** to conclude the crawler configuration and return to the collection main page.

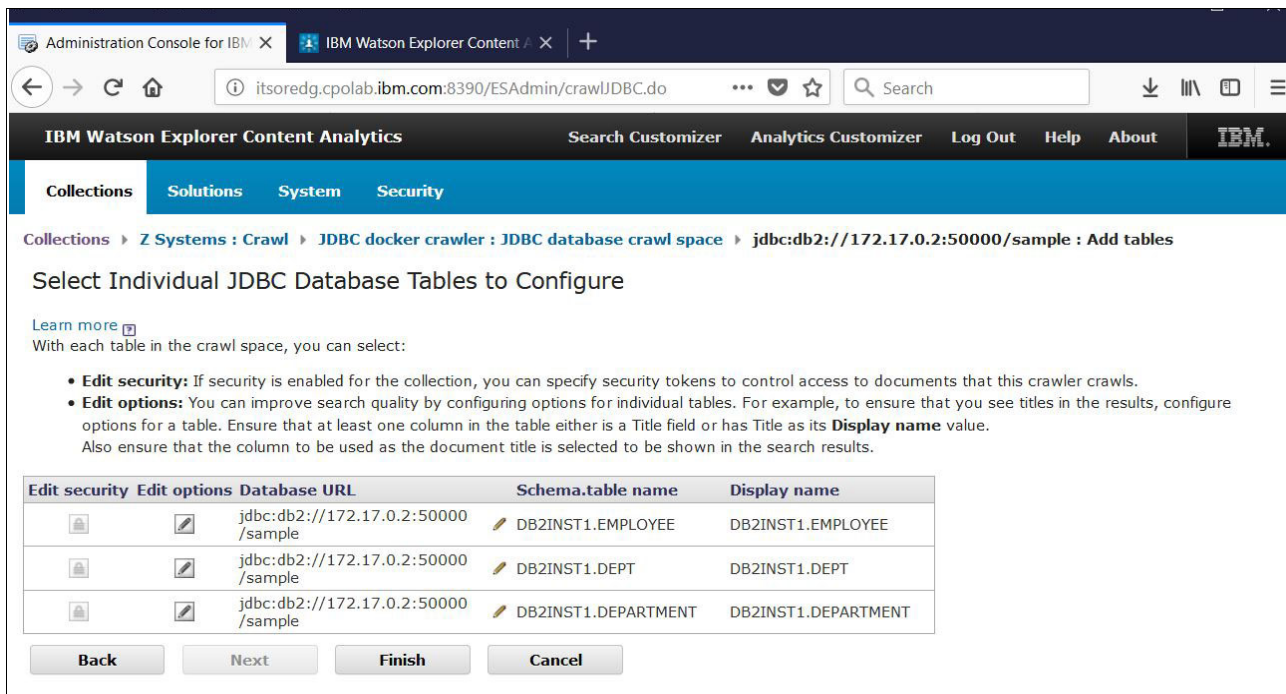


Figure 5-29 Tables configured and ready to crawl

- At the main collection page, you see that the crawler is configured, but its status is stopped (see Figure 5-30). Click the arrow icon that is next to the word “stopped” to start crawling the database.

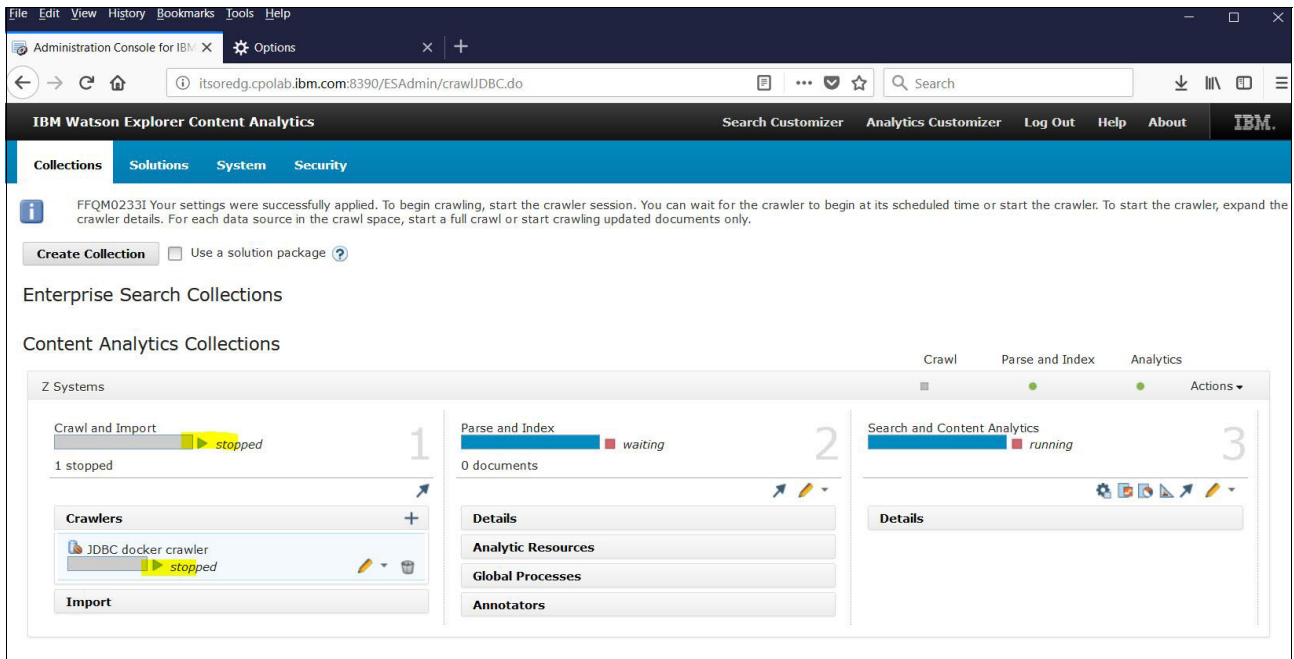


Figure 5-30 Configured crawler

Within minutes, you see the number of documents under the Parse and Index section that were crawled (see Figure 5-31).

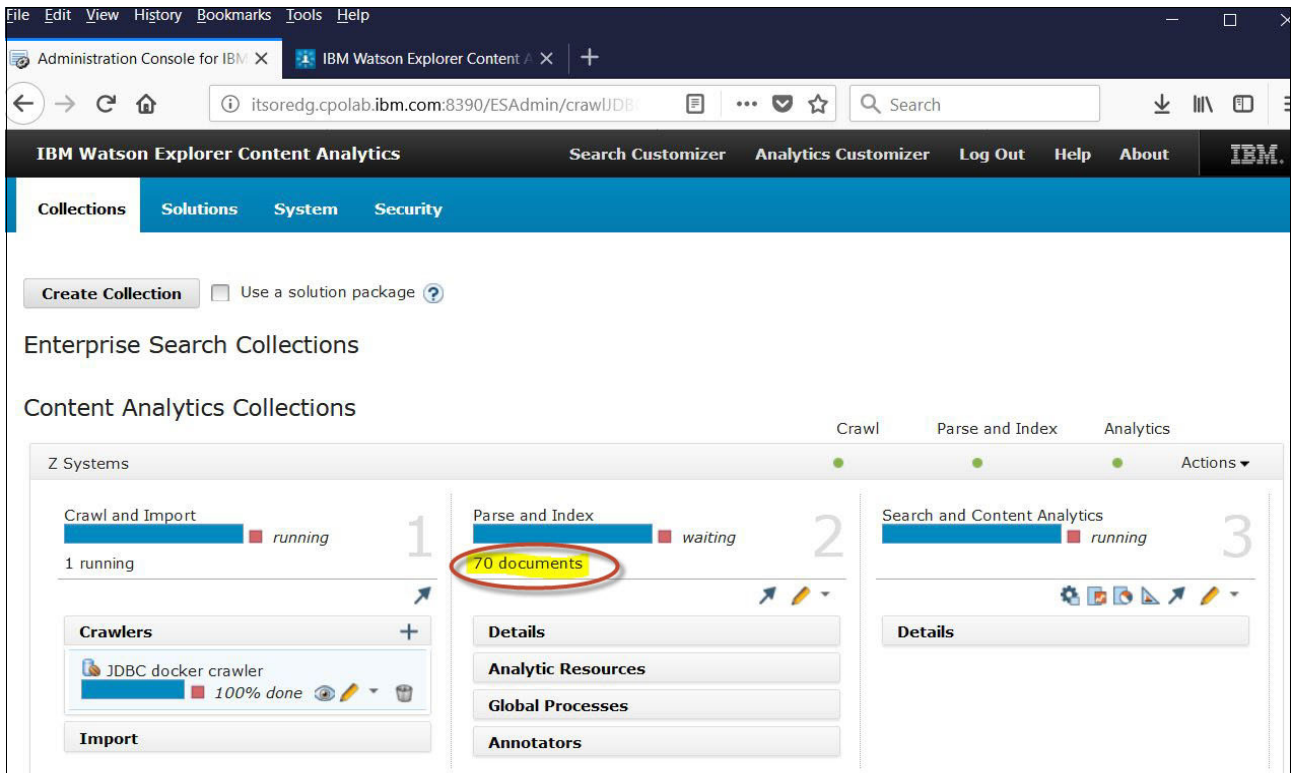


Figure 5-31 Documents parsed and indexed, ready for analyses

Searching database contents using Watson Explorer Content Analytics

After you finish the collection and the crawler configuration, you can start searching for the database contents. Log in to the Content Miner web interface that is listening on port 8393. In our example, the following URL is used:

`http://itsoredg.cpolab.ibm.com:8393/ui/analytics/`

5.6.4 Mining the contents

After you create the collections and configure the crawlers, you can use the content analytics miner to analyze your content and discover trends, correlations, and deviations in data over time. The content analytics miner provides real-time statistical analysis of structured and unstructured data. An example of the Contents Analytics page is shown in Figure 5-32.

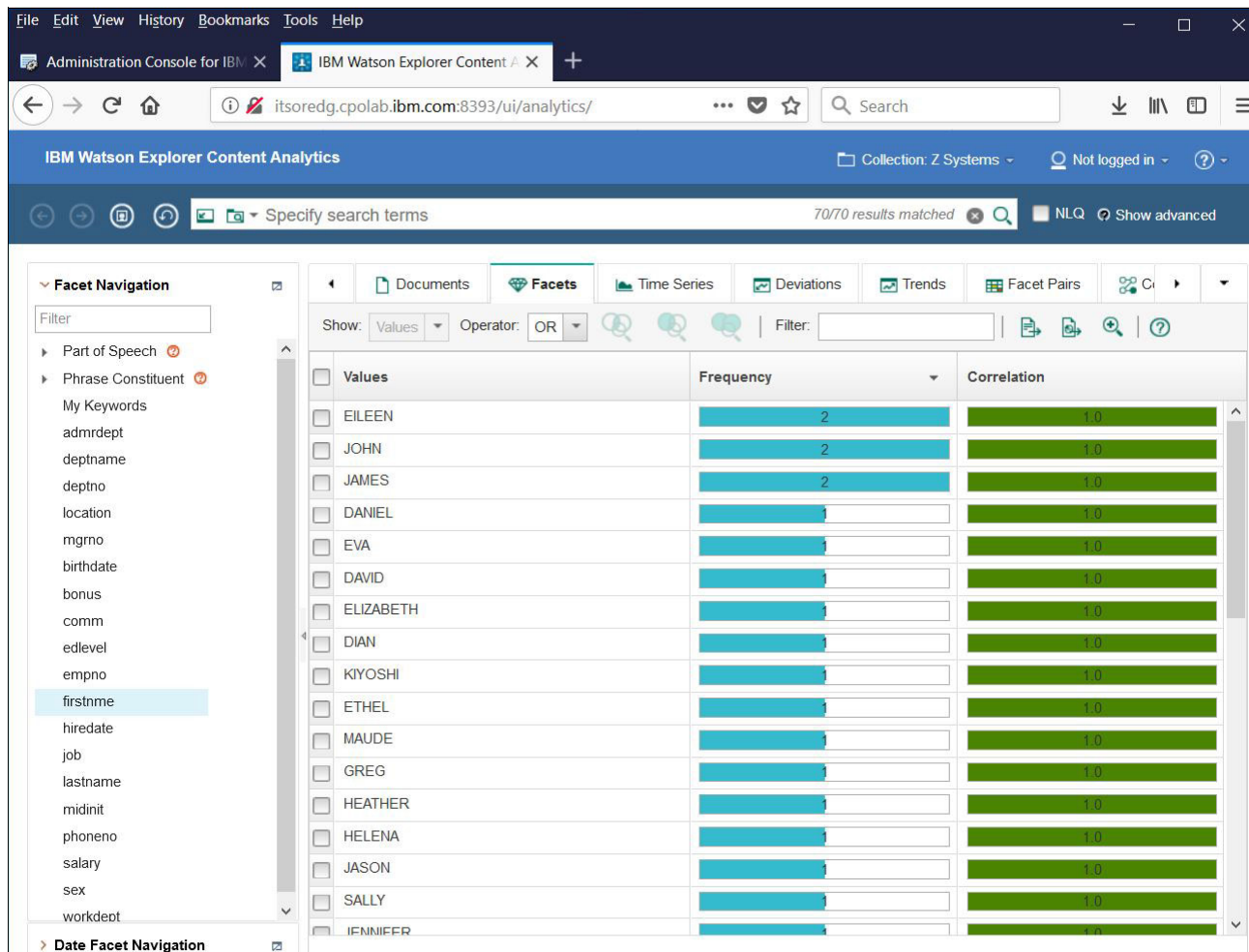


Figure 5-32 Analyzing the contents of the database.

As demonstrated in this use case, complex containers can interact with each other seamlessly. It does not matter if they are on the same host or on different networks.

The nature of containers is meant to be short-lived; therefore, you might want to save the search results information that is gathered with this container onto a different volume.

This information can be saved by creating an image in a different way.

Thus far, we created an image interactively. Now, we use a Dockerfile. A Dockerfile is a simple text file that contains instructions on how to build an image. All of the steps that we performed inside the container up to now can be on a Dockerfile; however, because that process was used, we use the base image instead.

Figure 5-33 shows how to create a directory to host the files and configuration.

```
[root@itsoredg ~]# mkdir wex12docker 1
[root@itsoredg ~]# cd wex12docker/
[root@itsoredg wex12docker]# vi Dockerfile 2
[root@itsoredg wex12docker]# vi entrypoint.sh 3
[root@itsoredg wex12docker]# ls -l
total 8
-rw-r--r--. 1 root root 83 Oct 30 03:43 Dockerfile
-rwxr-xr-x. 1 root root 596 Oct 30 03:54 entrypoint.sh
[root@itsoredg wex12docker]#
```

Figure 5-33 Creating an image using a Dockerfile

As indicated by a 1 in Figure 5-33, create a Dockerfile with a few instructions 2 (shown in Figure 5-34) and then build a new image. We also add a script file 3 that is run when the container starts.

In Figure 5-34, the script is called `entrypoint.sh` and started the WEX AC services for us.

The contents of the Dockerfile are shown in Figure 5-34.

```
[root@itsoredg wex12docker]# cat Dockerfile
FROM wexac1201:ver1
EXPOSE 8390 8393 8394
COPY ./entrypoint.sh /root/entrypoint.sh
ENTRYPOINT ["/root/entrypoint.sh"]
[root@itsoredg wex12docker]#
```

Figure 5-34 Sample Dockerfile

The contents of the `entrypoint.sh` file are shown in Figure 5-35.

```
[root@itsoredg wex12docker]# cat entrypoint.sh
#!/bin/sh
echo `date` >> /tmp/startwex.log 2>&1
if test -d "/wexdata/esdata/config"; then
    echo "Configuration files exist, using files from wexdata" >>
    /tmp/startwex.log 2>&1
    mv /home/esadmin/esdata /home/esadmin/esdata.bak >> /tmp/startwex.log
    2>&1
    ln -s /wexdata/esdata /home/esadmin/esdata >> /tmp/startwex.log 2>&1
    envfiles=("/home/esadmin/.profile" "/home/esadmin/.bash_profile")
    for f in "${envfiles[@]}"; do
        sed -i -e "s/\/home\/esadmin\/esdata\/wexdata\/esdata/g" "$f"
    >> /tmp/startwex.log 2>&1
    done
else
    echo "Using files from default installed location"; >>
    /tmp/startwex.log 2>&1
fi
su - esadmin -c "esadmin system startall" >> /tmp/startwex.log 2>&1
exec "$@"
[root@itsoredg wex12docker]#
```

Figure 5-35 Sample `entrypoint.sh` script

This example is a fairly simple Dockerfile, but it is all we need to build a new image. Building the image by using the Dockerfile is shown in Figure 5-36.

```
root@itsoredg wex12docker]# docker build -t mywex11201:v1 .
Sending build context to Docker daemon 3.584kB
Step 1/3 : FROM wexac1201:ver1
----> 77f7b50c918f
Step 2/3 : EXPOSE 8390 8393 8394
----> Running in 0db9cc4abfb1
----> fd5a77bf55a9
Removing intermediate container 0db9cc4abfb1
Step 3/3 : COPY ./entrypoint.sh /root/entrypoint.sh
----> 9466271f6256
Removing intermediate container 8efdc614176c
Successfully built 9466271f6256
Successfully tagged mywex11201:v1
[root@itsoredg wex12docker]#
```

Figure 5-36 Building an image using a Dockerfile

The command to list the images in our repository of images is shown in Figure 5-37.

```
itsoredg wex12docker]# docker image ls
REPOSITORY          TAG          IMAGE ID          SIZE
mywex11201          v1           9466271f6256     4.35GB
seconds ago
wexac1201           ver1         77f7b50c918f     4.35GB
minutes ago
registry.access.redhat.com/rhel7.5 latest       099889565575     198MB
weeks ago
[root@itsoredg wex12docker]#
```

Figure 5-37 Listing the images created

This new image can be used to start a container in the same way we did before. However, because we might have more than one container that uses the ports, we still need to redirect them to a different number.

The command that uses the `-d` option for detached instead of `-i` is shown in Figure 5-38. This command makes the container start and continue running in detached mode.

```
[root@itsoredg wex12docker]# docker container run -dt -p 7390:8390 -p 7393:8393
-h "wexac1201.itso.ibm.com" mywex11201:v1 /bin/bash
4b90a447ca001b14a71acad8cf1bad20d759885b1ce41e6a9620e34f4055c47f
[root@itsoredg wex12docker]#
```

Figure 5-38 Starting the container in detached mode

If you log in into the container, you should see that all WEX AC services were automatically started by the `entrypoint.sh` script. You can see that the services are running and attempting to access the web pages by using ports 7390 and 7393.

The `entrypoint.sh` script contains another utility. Before starting the WEX AC services, it first checks for the existence of the `/wexdata/esdata/config` directory. If this directory is found, the script creates a symbolic link from `/home/esadmin` to that directory and WEX AC uses the configuration files, which means that the data that is crawled by WEX survives outside of the container's space.

Redbooks

Getting Started with Docker Enterprise Edition on IBM Z

(0.2"spine)
0.17"->0.473"
90->249 pages



SG24-8429-00

ISBN 0738457507

Printed in U.S.A.

Get connected

