Redbooks
ibm.com/redbooks

Getting started with z/OS Container Extensions and Docker

Lydia Parziale

Zach Burns

Marco Egli

Redelf Janßen

Volkmar Langer

Subhajit Maitra

Edward McCarthy

Eric Marins

Jim Newell

IBM Z

IBM

Redbooks

IBM Redbooks

Getting started with z/OS Container Extensions

November 2019

SG24-8457-00

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

First Edition (November 2019)

This edition applies to z/OS Version 2 Release 4, Program Numner 5650-ZOS with APARs OA58008 and at least OA58267.

© 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 ix
Trademarks x

Preface xi
Authors xi
Now you can become a published author, too! xiii
Comments welcome xiii
Stay connected to IBM Redbooks xiii

Chapter 1. Introduction 1
1.1 z/OS Container Extensions overview 2
1.2 Container concepts 2
1.2.1 Why do we have containers 4
1.2.2 Docker overview 6
1.3 zCX architecture 9
1.4 Why use z/OS Container Extensions 13
1.4.1 Qualities of service 14
1.4.2 Colocation of applications and data 14
1.4.3 Security 16
1.4.4 Consolidation 17
1.4.5 Example use cases 18
1.5 Additional considerations 18

Chapter 2. z/OS Container Extensions planning 19
2.1 Prerequisites 20
2.1.1 IBM Z hardware 20
2.1.2 z/OS Version 2, Release 4 24
2.1.3 z/OS Management Facility (z/OSMF) 25
2.1.4 Base DASD requirements 27
2.1.5 Planning for network connectivity 31
2.2 Planning for containers 36
2.3 Private registry considerations 36
2.4 Backup and recovery considerations 36

Chapter 3. Security - Overview 39
3.1 zCX instance security 40
3.1.1 Implications of zCX running as a started task 40
3.1.2 zCX RACF planning 40
3.1.3 RACF groups and user IDs for zCX 41
3.1.4 zFS files and VSAM linear data sets 41
3.1.5 USS directory and files 42
3.1.6 TCPIP Networking 43
3.1.7 z/OSMF 43
3.1.8 Other data sets 43
3.2 Security within the zCX instance 43
3.2.1 Linux in the zCX instance 43
3.2.2 The SSH container 43
3.2.3 zCX administration user IDs 44
3.3 Docker in zCX versus Docker on distributed 45

© Copyright IBM Corp. 2019. All rights reserved.

3.3.1 Docker capabilities of the SSH container 45
3.3.2 Considerations regarding the mounting of file systems 45

Chapter 4. Provisioning and managing your first z/OS Container 47
4.1 Overview of zCX provisioning process 48
4.2 RACF planning 48
4.2.1 Overview 48
4.2.2 The zCX Admin user ID 49
4.2.3 Planning the first logon to a zCX instance 49
4.2.4 Create RACF groups 50
4.2.5 Create RACF user IDs 50
4.2.6 Create SSH Keys 51
4.3 USS planning 52
4.3.1 zCX directories 52
4.4 z/OSMF 53
4.4.1 z/OSMF configuration 53
4.5 Configure TCPIP addresses 55
4.6 SMS planning 57
4.7 zCX property file 58
4.7.1 Customizing the properties file 58
4.7.2 Setting the SSH key in property file 59
4.8 Run zCX provisioning workflow 60
4.8.1 Log on to z/OSMF 60
4.8.2 Select the zCX provisioning workflow 61
4.8.3 Execute the first step of the workflow 64
4.8.4 Execute remaining workflow steps 68
4.8.5 Viewing created instance directory 70
4.8.6 Obtain the command to start the zCX instance 71
4.9 Starting the zCX instance 71
4.9.1 Assigning user ID that zCX instance runs under 71
4.9.2 Access to zCX instance directory by the zCX started task 72
4.9.3 Start the zCX instance 74
4.9.4 Check zCX startup messages 74
4.9.5 Step failure in the workflow 76
4.10 Log on to the zCX instance 77
4.11 Reconfiguring a zCX instance 77
4.11.1 What is reconfigured? 78
4.11.2 Checking current settings 78
4.11.3 Create reconfiguration zCX workflow 78
4.11.4 Execute the reconfiguration workflow 79
4.11.5 Stop and start the zCX instance 82
4.11.6 Verifying reconfiguration changes 82
4.12 Adding a volume to a zCX instance 82
4.12.1 Overview 82
4.12.2 Checking default disk space allocation 83
4.12.3 Create add data disks zCX workflow 83
4.12.4 Run the Add Data Disks workflow 84
4.12.5 Stop and start the zCX instance 88
4.12.6 Verifying Add Data Disk changes 88
4.13 Deprovisioning a zCX instance 89
4.13.1 Create deprovisioning zCX workflow 90
4.13.2 Execute the deprovision workflow 90
4.13.3 Verify that the zCX instance has been deprovisioned 92

4.14 Other provisioning workflows . . . 92
4.15 How to rerun a workflow . . . 92

Chapter 5. Your first running Docker container in zCX . . . 95
5.1 Overview of the process . . . 96
5.1.1 Provision your instance . . . 96
5.1.2 Get a Docker image to run . . . 96
5.1.3 Start your image . . . 96
5.1.4 Access the provided service by the image . . . 96
5.2 Get an image from Docker Hub . . . 96
5.2.1 Download from Docker Hub . . . 96
5.2.2 Download via a local Repository . . . 97
5.2.3 Load the image from a .tar file . . . 97
5.3 Run your Docker image on z/OS . . . 97
5.3.1 hello-world . . . 97
5.3.2 HTTP Server (nginx) . . . 98
5.3.3 Enhanced installation . . . 102
5.4 Managing your Docker image . . . 104
5.4.1 Start . . . 104
5.4.2 Display . . . 104
5.4.3 Stop . . . 105
5.4.4 Terminate . . . 105

Chapter 6. Private registry implementation . . . 107
6.1 Private registry overview . . . 108
6.2 Tag discussion for images . . . 109
6.3 Building a private registry image . . . 112
6.4 Running a local private registry . . . 117
6.4.1 Stopping a private registry . . . 118
6.4.2 Removing a private registry . . . 118
6.5 Deploying a secure private registry . . . 118
6.5.1 When zCX appliance is behind a firewall . . . 121
6.6 Creating TLS certificates on Windows or Macintosh . . . 130
6.7 Working with tags . . . 134
6.8 Deleting an image from a private Docker registry . . . 137
6.9 Using the private registry to create containers . . . 139

Chapter 7. Operation . . . 141
7.1 Software maintenance . . . 142
7.1.1 Maintenance for zCX . . . 142
7.1.2 Maintenance for containers . . . 144
7.1.3 Building and maintaining your own image . . . 146
7.2 Automation . . . 149
7.2.1 Automating zCX instances . . . 149
7.2.2 Automating containers . . . 150
7.3 Backup and recovery . . . 151
7.3.1 Backup and recovery on zCX instance level . . . 152
7.3.2 Backup and recovery on container level . . . 155
7.4 Diagnosis . . . 156
7.4.1 Ways to check healthiness of zCX . . . 156
7.4.2 Gathering problem data for zCX . . . 158
7.5 Monitoring with RMF on zCX instance level . . . 160
7.5.1 RMF overview display . . . 162
7.5.2 RMF CPC capacity . . . 164

7.5.3 RMF job information . . . 165
7.6 Configuring Grafana to monitor zCX containers . . . 166
7.6.1 Install Node-Exporter . . . 167
7.6.2 Install cAdvisor . . . 168
7.6.3 Install Prometheus . . . 169
7.6.4 Installation of Grafana . . . 171
7.6.5 Adding Prometheus as data source to Grafana . . . 172
7.6.6 Creating a first dashboard . . . 174
7.7 Monitoring with Grafana on container level . . . 177
7.7.1 Adjusting the dashboard . . . 177
7.7.2 Instance level data . . . 179
7.7.3 Container level data . . . 181

Chapter 8. Integrating container applications with other processes on z/OS . . . 187
8.1 Interconnecting IBM MQ on z/OS with IBM MQ in zCX . . . 188
8.1.1 Objectives . . . 188
8.1.2 Architecture . . . 188
8.1.3 Scenario . . . 189
8.1.4 Implementation . . . 190
8.2 Accessing Db2 from within Docker container when you are using Jupyter Notebook 199
8.2.1 What is a Jupyter Notebook . . . 199
8.2.2 Scenario . . . 199
8.2.3 Creating a Docker Image with Jupyter installed . . . 200
8.2.4 Using a Jupyter Notebook to access Db2 . . . 206
8.3 Accessing application in a zCX container from z/OS . . . 209
8.3.1 Target application in Docker container . . . 209
8.3.2 Setting up the etcd container . . . 210
8.3.3 Build the etcd Docker image . . . 211
8.3.4 Run the etcd Docker image . . . 212
8.3.5 The z/OS application to call etcd . . . 212
8.3.6 Detail on setting up an etcd container . . . 214
8.3.7 Summary . . . 217

Chapter 9. zCX user administration . . . 219
9.1 Local user management . . . 220
9.1.1 Adding Docker users . . . 220
9.2 Configuring zCX to use an LDAP server . . . 222
9.2.1 LDAP server configuration . . . 222
9.2.2 Creating the LDAP configuration file . . . 225
9.2.3 Enabling LDAP server authentication through z/OSMF Workflows . . . 226
9.3 Resources on the provisioning server and verifying that LDAP is enabled . . . 228

Chapter 10. Persistent data . . . 231
10.1 Overview . . . 232
10.2 Using Docker volumes for persistent data . . . 235

Chapter 11. Swarm on zCX . . . 239
11.1 Swarm introduction . . . 240
11.2 zCX in swarm mode . . . 241
11.2.1 Removing node from swarm . . . 242
11.2.2 Initializing swarm mode . . . 243
11.2.3 Adding node manager to the cluster . . . 243
11.2.4 Adding worker nodes to the manager node . . . 244
11.2.5 Creating a Docker service . . . 245

11.2.6 Scaling up a container 246
11.2.7 Changing node availability to simulate a scheduled maintenance 247
11.2.8 Promoting a worker node 248
11.2.9 Demoting a manager node 248
11.2.10 Scaling down a service 248
11.2.11 Considerations regarding the number of manager nodes. 249

Appendix A. Obtaining the additional material 251

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.

© Copyright IBM Corp. 2019. All rights reserved.

Trademarks

IBM, the IBM logo, and 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.

AIX®
CICS®
Cloudant®
Cognos®
Db2®
DB2®
GDPS®
HyperSwap®
IBM®
IBM Cloud™
IBM Z®
IBM z14®
IBM z15™
MQSeries®
MVS™
OS/390®
Parallel Sysplex®
RACF®
Redbooks®
Redbooks (logo) ®
Tivoli®
UrbanCode®
WebSphere®
z/OS®
z/VM®
z15™

The following terms are trademarks of other companies:

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Preface

IBM® z/OS® Container Extensions (IBM zCX) is a new feature of the next version of the IBM z/OS Operating System (z/OS V2.4). It makes it possible to run Linux on IBM Z® applications that are packaged as Docker container images on z/OS. Application developers can develop, and data centers can operate, popular open source packages, Linux applications, IBM software, and third-party software together with z/OS applications and data.

This IBM Redbooks® publication helps you to understand the concepts, business perspectives, and reference architecture for installing, tailoring, and configuring zCX in your own environment.

Authors

This book was produced by a team of specialists who gathered from around the world to work at IBM Redbooks Center in Poughkeepsie.

Lydia Parziale is a Project Leader for the IBM Redbooks 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 PMI 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.

Zach Burns is a Software Engineer from Golden, Colorado. He holds a bachelor degree of Software Engineering from Clarkson University in Potsdam, NY. Zach has ten years of experience in software development and has contributed to many software and firmware deliverables for the IBM Z platform including the Software Inventory Tool and IBM zAware. He currently works as a DevOps architect providing solutions for internal and external customers using zCX and iZODA.

Marco Egli is a Mainframe Engineer at Swiss Re, who is based in Switzerland. He has worked for more than 10 years in the Mainframe area and has been responsible for Software and Hardware installations as well as the overall architecture of the Mainframe environment. His focus shifted during the past months towards exploiting new technologies on the mainframe. He has written extensively on Chapter 5, “Your first running Docker container in zCX” on page 95 and 8.2, “Accessing Db2 from within Docker container when you are using Jupyter Notebook” on page 199 and contributed to other chapters.

Redelf Janssen is a Client Technical Specialist at IBM Systems hardware sales in Bremen, Germany. He holds a degree in Computer Science from the University of Bremen and joined IBM in 1988. He is responsible for supporting IBM Z customers in Germany. His areas of expertise include IBM Z hardware, z/OS, mainframe simplification, storage management, and availability management. He has written IBM Redbooks publications about several IBM OS/390®, z/OS, and z/OSMF releases. He also teaches classes on z/OS and z/OSMF.

Volkmar Langer is an IBM MVS™ Systems Programmer at Dataport in Germany. He has more than 30 years of experience in z/OS and IBM Z hardware. For over 20 years, he has been responsible for the concepts and design of Dataport’s z/OS Sysplex environments, including LPAR design and disaster recovery concepts.

© Copyright IBM Corp. 2019. All rights reserved.

Subhajit Maitra is a Consulting IT Specialist and member of the IBM North America System Z Hybrid Cloud Technical Sales team based in Hartford, CT. In addition to zCX and IBM Cloud™, his expertise includes IBM Operational Decision Manager, IBM Integration Bus, IBM MQ, IBM Event Streams, IBM Db2®, and CICS® on IBM Z. Subhajit, he has worked in IT for 26 years, and is also an IBM Global Technical Ambassador for Central and Eastern Europe, where he helps IBM clients to implement business-critical solutions on IBM Z servers. He has written IBM Redbooks publications about several IBM Middleware, CICS, and DevOps topics. He holds a Master's degree in computer science from Jadavpur University, in Kolkata, India, and is an IBM zChampion.

Eric Marins Senior IT Architect or IBM Global Technology and Services in Brazil, has focused on hybrid cloud solutions, Infrastructure and Platform solutions and competencies, including High Availability, Disaster Recovery, networking, Linux, and Cloud. Eric works with IGA (IBM Global Account), designing and implementing complex hybrid cloud solutions involving Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) capabilities. He has more than 17 years of experience in designing and implementing Linux on IBM Z solutions. He is IBM L3 IT Specialist certified and IBM L3 IT Architect certified. 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, Kubernetes, 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, *Scale up for Linux on IBM Z*, REDP-5461 and *Getting Started with Docker Enterprise Edition on IBM Z*, SG24-8429.

Edward McCarthy is an IBM-certified specialist with over 18 years' experience working with the IBM WebSphere® Application Server on various operating systems such as z/OS, Linux on IBM Z, AIX®, and MS Windows. He has designed and configured numerous WebSphere Application Server environments for a number of customers. Additionally, he has been involved with all aspects of WebSphere Application Servers such as tuning, automating administration, problem solving, and integration. Prior to joining IBM in 2000, he was a CICS and IBM MQ systems programmer with an Australian Government Department for over nine years. Mr. McCarthy has worked on several IBM Redbooks and presented on WebSphere topics at various conferences.

Jim Newell is a Client Technical Architect focusing on IBM Z solutions in the United States. He has over 30 years of experience working on the IBM Z platform. He holds a B.S. in Computer Science from the City University of New York. His areas of expertise include Cloud, Analytics, Mainframe and Distributed Platforms, Software Architecture, Infrastructure Architecture, Systems Management, Data Center Operations, and Technical Sales. He also mentors students on various technology subjects at several high schools and colleges where he lives, as part of the IBM Academic Initiative.

A very special thanks to Gary Puchkoff, STSM z/OS New Technology, IBM Poughkeepsie for all of the time, knowledge, and patience he extended to this team.

Thanks to the following people for their contributions to this project:

Robert Haimowitz
IBM WW Client Experience Center, Poughkeepsie

Gus Kassimis, Ahilan Rajadeva,Yüksel Günal, Joe Bostian, Anthony Giorgio, Gary Puchkoff, Mike Kasper, Kathryn Voss, Peter Spera
IBM Poughkeepsie

Mayur Raja, IBM MQ Development Lab
IBM Hursley, UK

Diego Jose Basso Pigossi, Fernando Aragao Costa
IBM Brazil

Julie Bergh
Rocket Software, USA

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 IBM Redbooks 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 here:

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, IBM Redbooks
  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

1

Introduction

z/OS Container Extensions is an exciting new capability that is delivered as part of IBM z/OS V2R4. It is designed to enable the ability to run almost any Docker container that can run on Linux on IBM Z in a z/OS environment alongside existing z/OS applications and data without a separate provisioned Linux server.

This extends the strategic software stack on z/OS as developers can build new, containerized applications, by using Docker and Linux skills and patterns, and deploy them on z/OS, without requiring any z/OS skills. In turn, data centers can operate popular open source packages, Linux applications, IBM software, and third-party software together with z/OS applications and data, leveraging industry standard skills.

As a new user for z/OS Container Extensions (zCX), it's important that you have an understanding of the underlying constructs that make up zCX. This chapter introduces you to those concepts and includes the following topics:

- 1.1, "z/OS Container Extensions overview" on page 2
- 1.2, "Container concepts" on page 2
- 1.3, "zCX architecture" on page 9
- 1.4, "Why use z/OS Container Extensions" on page 13
- 1.5, "Additional considerations" on page 18

© Copyright IBM Corp. 2019. All rights reserved.

1.1 z/OS Container Extensions overview

Enterprises increasingly embrace the use of hybrid workloads. Accordingly, the z/OS software ecosystem must expand and enable cloud-native workloads on z/OS. IBM z/OS Container Extensions (zCX) is a new feature of z/OS V2R4 that helps to facilitate this expansion.

zCX enables clients to deploy Linux on Z applications as Docker containers in a z/OS system to directly support workloads that have an affinity to z/OS. This is done without the need to provision a separate Linux server. At the same time, operational control is maintained within z/OS and benefits of z/OS Qualities of Service (QoS) are retained.

Linux on Z applications can run on z/OS, so you are able to use existing z/OS operations staff and reuse the existing z/OS environment.

zCX expands and modernizes the software ecosystem for z/OS by including Linux on Z applications, many of which were previously available only on a separately provisioned Linux. Most applications (including Systems Management components and development utilities/tools) that are currently available to run on Linux will be able to run on z/OS as Docker containers only.

Anything with s390x architecture (the IBM Z opcode set) in Docker Hub can be run in z/OS Container Extensions. The code is binary-compatible between Linux on Z and z/OS Container Extensions. Also, multi-platform Docker images will be able to run within z/OS Container Extensions.

In addition to open source packages, IBM plans to have IBM and third-party software available for the GA of z/OS 2.4. It is intended that clients will be able to participate with their own Linux applications, which can easily be packaged in Docker format and deployed in the same as open source, IBM, and vendor packages.

1.2 Container concepts

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

This section introduces you to the concepts and information about Docker containers on IBM Z systems. This technology is transforming the application world and bringing a new vision to improve a developer's workflow and speed up application deployment.

The use of containers has quickly become widespread in early-adopter organizations around the world, and it 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. These qualities make possible the quick deployment of applications even while the demand for complex applications increases.

Docker is lightweight technology that runs multiple containers (group of processes that run in isolation) simultaneously on a single host. As a result, the developer's workflow improves and application deployment happens more quickly.

Docker takes hardware virtualization to the next level. The major advantage of this technology is the ability to encapsulate an application in a container with its own operating environment along with its dependencies (code, run time, system tools, system libraries, and settings). So,

you can deploy new applications without needing to install an operating system in a virtual machine, install software, or any other dependencies. In fact, everything you need to deploy for your application is available as one complete package in the form of a container.

Like a hypervisor, Docker allows sharing of resources, such as network, CPU, and disk to run multiple encapsulated workloads. Also, it provides ways to control how many resources can be used by a container. That way, you ensure that an application runs with all necessary resources, yet should not exhaust all resources on the host.

The Figure 1-1 illustrates the difference between Virtual Machine and Container architectures.

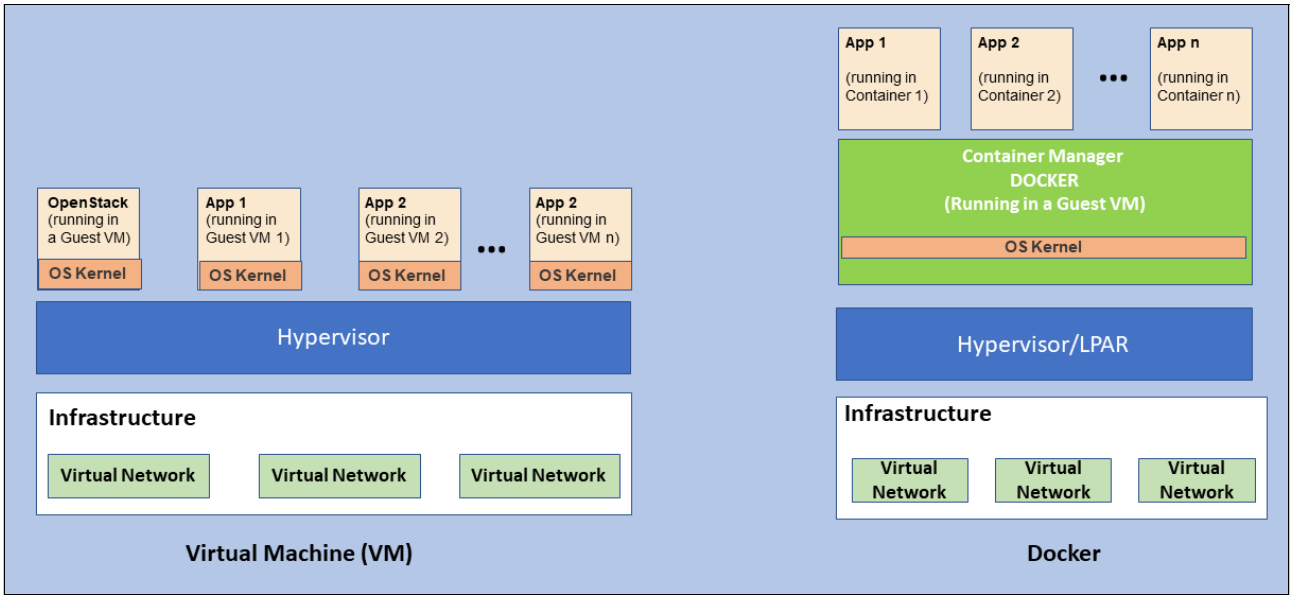


Figure 1-1 Virtual Machine versus Docker

At first, VMs gained 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.

The difference is the layer at which abstraction takes place:

- VMs provide abstraction from the underlying hardware.
- Containers provide abstraction at the application layer (or operating system layer) that packages code and dependencies together.

It is common to deploy each application on its own server (or virtual server), but this approach might result in an inefficient system without financial benefits. The operational costs and the amount of effort that is required to control this large environment tend to exceed the expense and efficiency of a container infrastructure.

With container technology, organizations can reduce costs, increase efficiencies, and increase profits because the capability of applications can be quickly deployed to improve service.

It is easier to spin up a virtual machine to test a new application release, upgrade, or deploy a new operating system. However, it can produce virtual machine sprawl. Therefore, system administrators and developers are adopting container technology to combat this problem with

a better use of server resources. Additionally, containers help organizations to meet growing business needs, without increasing the number of servers or virtual servers. The result is fewer efforts to maintain the environment.

1.2.1 Why do we have containers

This section describes some reasons that application developers and system administrators are adopting container technology to deploy their applications.

No need to run a full copy of an operating system

Unlike VMs, containers run directly on the kernel of hosting OS, which makes them lightweight and fast. Each container is a regular process that is running on the hosting Linux kernel, with its own container configuration and container isolation. This design brings more efficiency than VMs in terms of system resources, and it 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. As a result, developers can 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. Chapter 6., “Private registry implementation” on page 107 provides information on how the related concept of tagging works.

Agility to deploy new applications

Today, developers face challenges when they must deploy new functions into a complex applications. The environments are more critical to operations, bigger (several components), and have many online users that are connected to the system.

For these reasons, developers are transitioning from monolithic applications to distributed microservices. As a result, organizations move more quickly, meet unexpected surges in mobile and social requests, and scale their business consistently.

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

Docker is gaining popularity for the management of the new architecture that is based in microservices. It helps to manage and deploy this new architecture and quickly makes replications and achieves redundancy where the infrastructure requires it. Thanks to Docker, the setup of containerized applications can be 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 run different services such that one container does not affect another container, despite the fact that they run in the same Docker host. In our case, that “same Docker host” is the zCX instance.

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

With *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.

The Figure 1-2 shows isolation that uses Linux *namespaces* and *cgroups*.

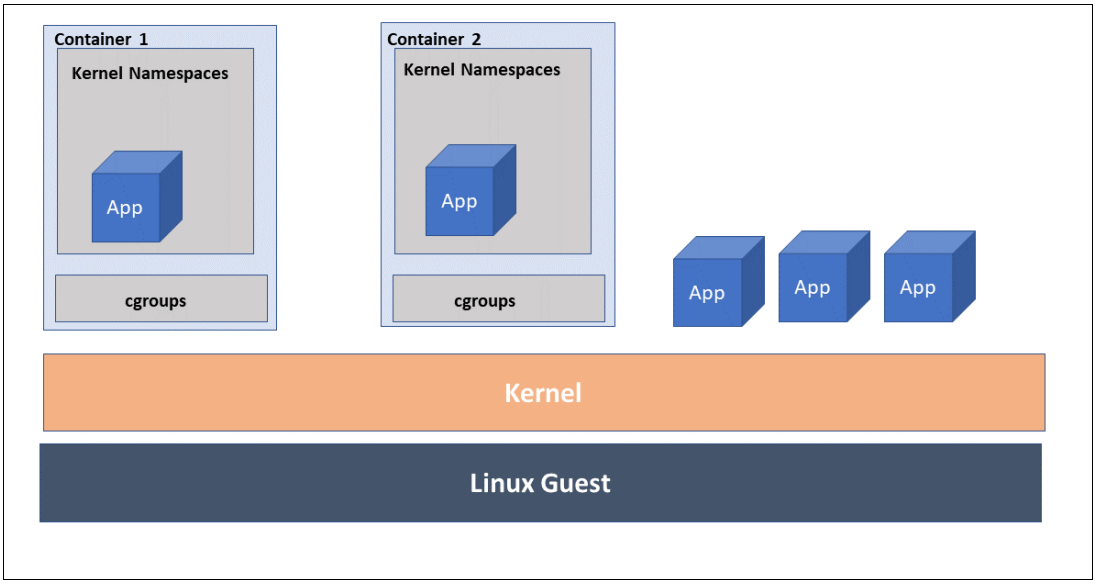


Figure 1-2 Linux namespaces and cgroups

Better resource utilization

Virtualization technology provides 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

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 that a particular platform provides to an organization — rather than which platform a developer prefers.

Application portability provides a great deal of flexibility in deploying applications into production and in choosing which platform to use for application development.

Containers make it easier to move the contained application across systems and run it without compatibility issues.

Colocation of data

For clients that run Linux applications on x86 platforms, it might be necessary to access data that resides on traditional mainframe systems. Colocation alongside the DATA can provide several advantages. Containerizing those applications and running them under zCX allows them to leverage the rich qualities of service that IBM Z and z/OS provide.

1.2.2 Docker overview

In this section, we introduce several basic Docker components, many of which we discuss later 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 1-3.

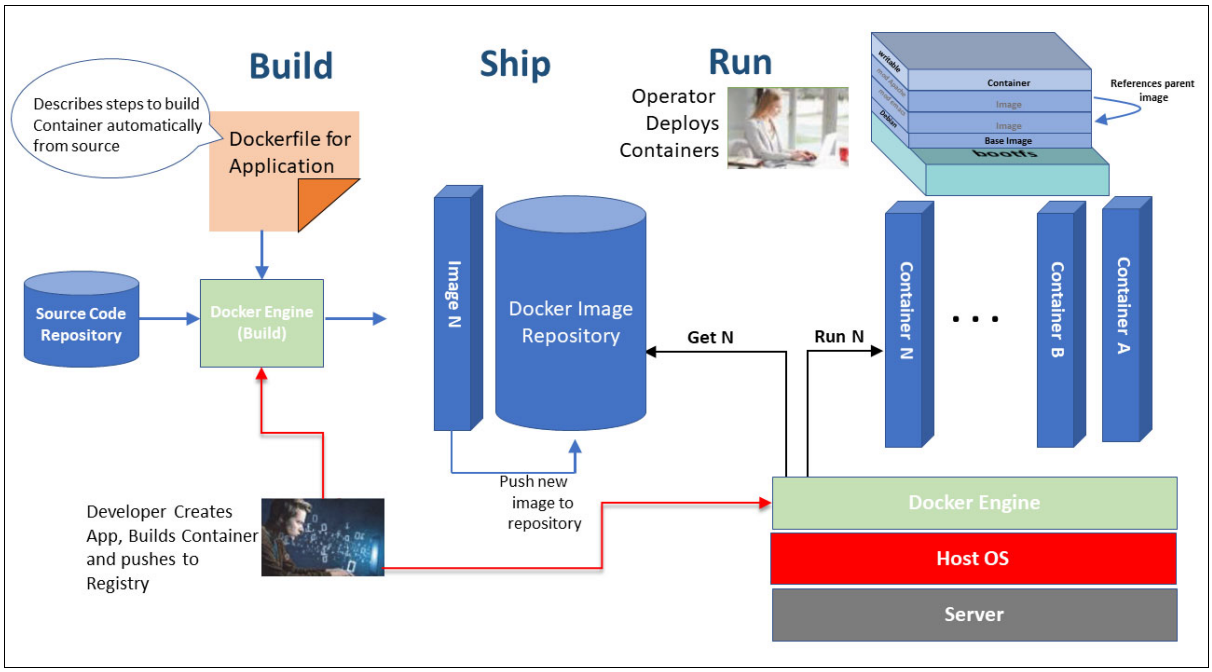


Figure 1-3 Overview of Docker components

Docker image

A Docker image is a read-only snapshot of a container that is stored in Docker Hub or any other repository. As a 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 caching.

See the following resources for Docker images:

- 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.

Consider the following points regarding a Dockerfile:

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

Figure 1-4 on page 7 and Figure 1-5 on page 7 show more information about a Dockerfile.

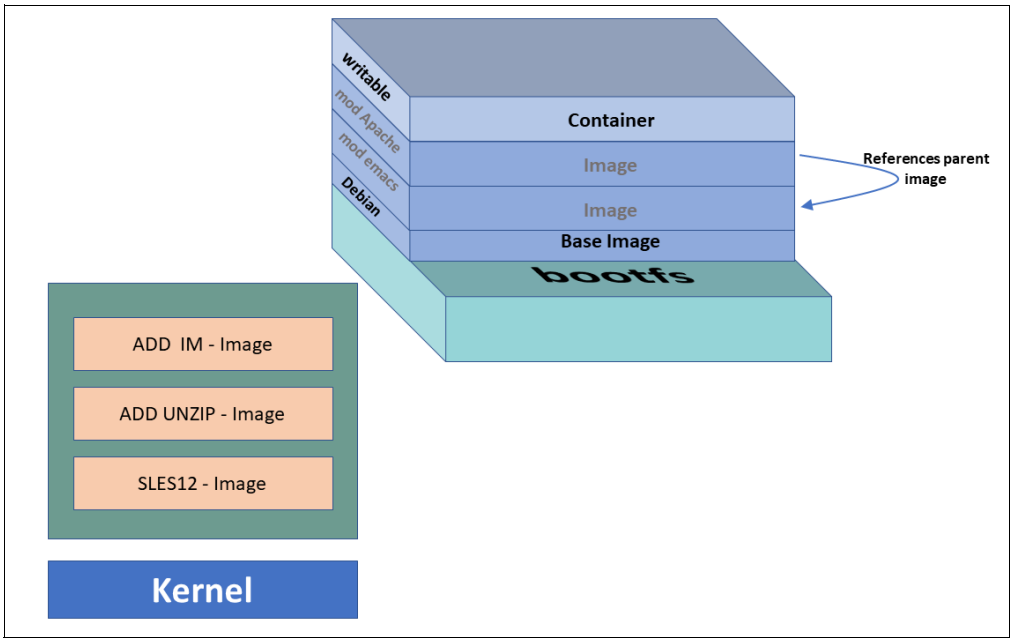


Figure 1-4 Dockerfile Architecture

```
# Dockerfile to build a container with IBM IM
FROM sles12.1:wlp
MAINTAINER Wilhelm Mild "mildw@de.ibm.com"
# Set the Install Reopsitories – 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 1-5 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 reside. 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 1-6.

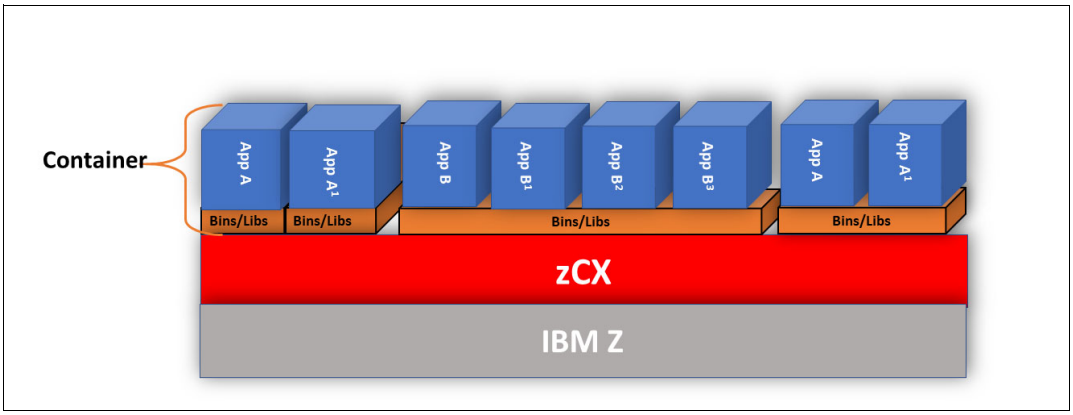


Figure 1-6 Docker containers

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. They use the **`docker push`** and **`docker pull`** commands, respectively. An example of Docker Registry (Hub) is shown in Figure 1-7.

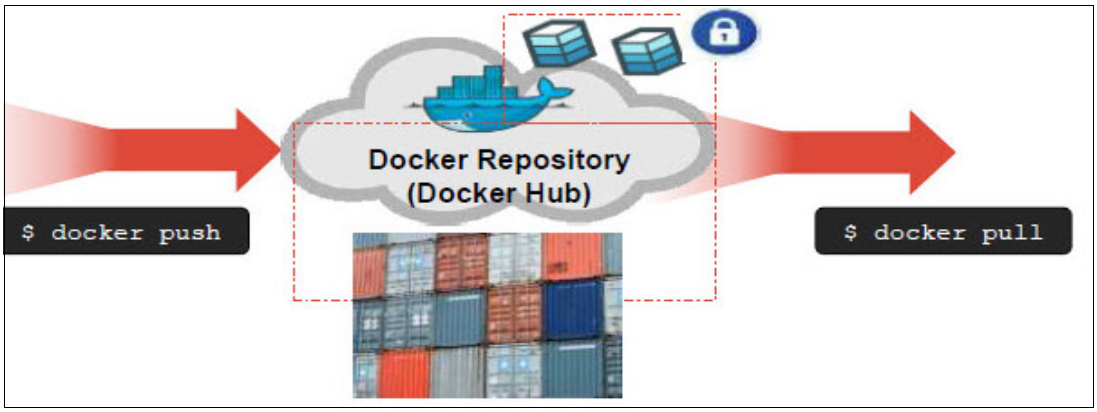


Figure 1-7 Docker registry

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

DockerHub is a popular and public registry that offers to the open community a place to store Docker images.

For more information about Docker Hub, see this website.

Private registries

If your organization needs 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 1-1.

Table 1-1 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.

You can find detail in how to implement a private registry service in chapter 6 of this publication.

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 1-8.

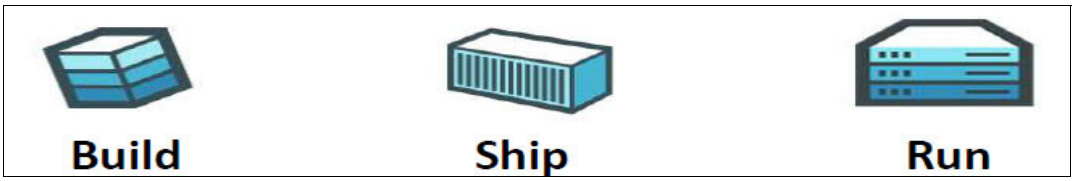


Figure 1-8 Docker Engine functions

Consider the following points regarding the Docker Engine:

- Runs on any zCX instance or virtual Linux server locally.
- Uses Linux kernel cgroups and namespaces for resource management and isolation.

1.3 zCX architecture

zCX is a feature of z/OS V2R4 that provides a pre-packaged turnkey Docker environment that includes Linux and Docker Engine components and is supported directly by IBM. The initial focus is on base Docker capabilities. zCX workloads are zIIP-eligible, so it provides competitive price performance.

There is limited visibility into the Linux environment. Access is as defined by Docker interfaces. Also, there is little Linux administrative overhead.

The basic architecture for zCX is described in Figure 1-9. zCX runs as an address space on z/OS that contains a Linux Kernel, Docker Engine, and the containers that can run within that instance.

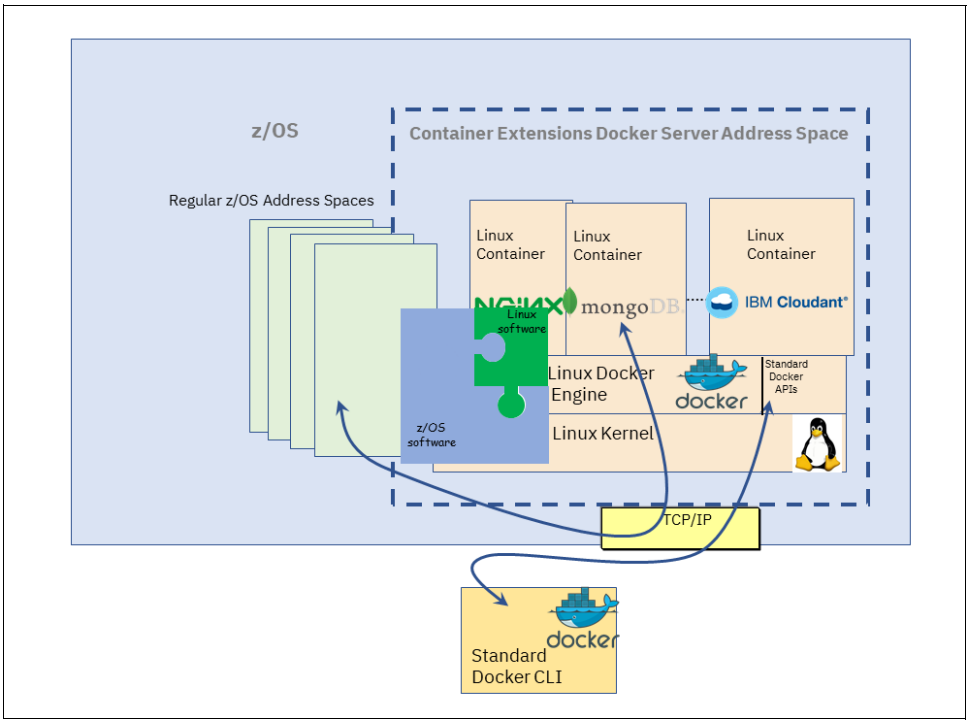


Figure 1-9 zCX basic architecture

Within the zCX address space is the z/OS Linux Virtualization Layer. This layer allows virtual access to z/OS Storage and Networking. It uses the virtual interface to communicate with the Linux kernel, which allows zCX to support unmodified, open source Linux on Z kernels.

Linux storage and disk access is provided through z/OS owned and managed VSAM data sets. This allows zCX to leverage the latest I/O enhancements (for example, zHyperLinks, I/O fabric diagnostics, and so on), built-in host-based encryption, replication technologies, and IBM HyperSwap®. See Figure 1-10 for details.

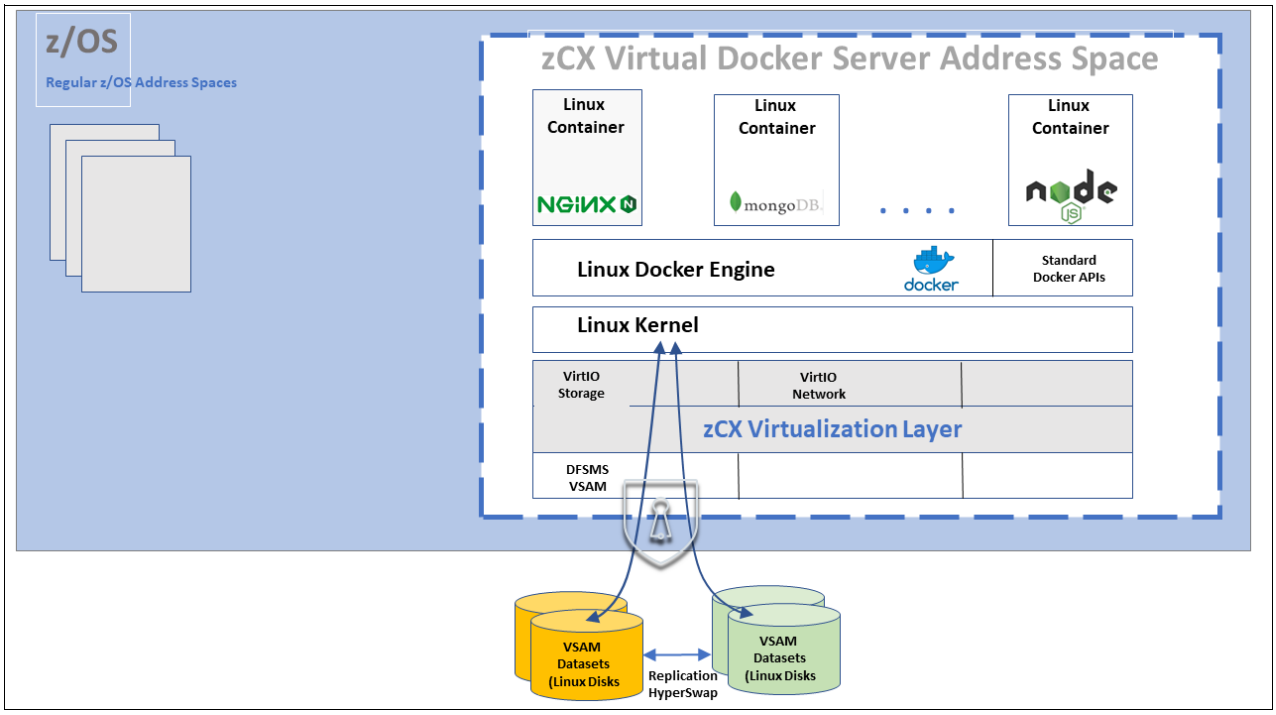


Figure 1-10 Linux storage and disk access

Linux network access is provided through a high-speed virtual SAMEHOST link to the z/OS TCP/IP protocol stack. Each Linux Docker Server is represented by a z/OS owned, managed, and advertised Dynamic VIPA (DVIPA). This allows the restart of the zCX instance in another system in the sysplex. Cross memory services provide high-performance network access across z/OS applications and Linux Docker containers. To restrict external access, z/OS TCP/IP filters can be used. See Figure 1-11 for details.

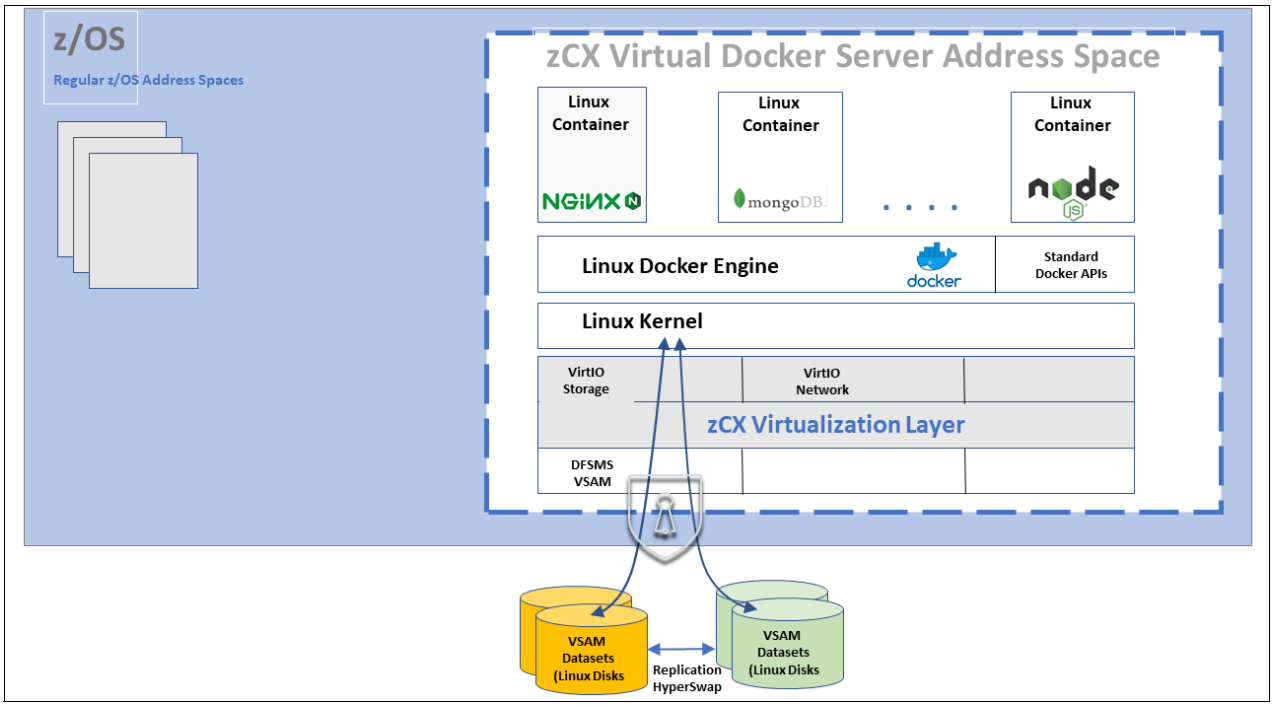


Figure 1-11 Linux network access

CPU, memory, and workload management are all provided by z/OS services. Memory is provisioned through the zCX address space by using private, above-the-2-GB-bar, fixed memory that is managed by VSM (Virtual Storage Manager) and RSM (Real Storage Manager). Virtual CPUs are provisioned to each zCX address space. Each virtual CPU is a dispatchable thread (in other words, MVS TCB) within the address space. zIIP access is provided by the MVS dispatcher. Multiple Docker container instances can be hosted within a single zCX instance.

WLM policy and resource controls including service class association, goals, importance levels, tenant resource groups (including optional caps for CPU and real memory) are extended to zCX.

SMF data is available as well (Type, 30, 72, and so on), which enables z/OS performance management and capacity planning. Figure 1-12 provides more details.

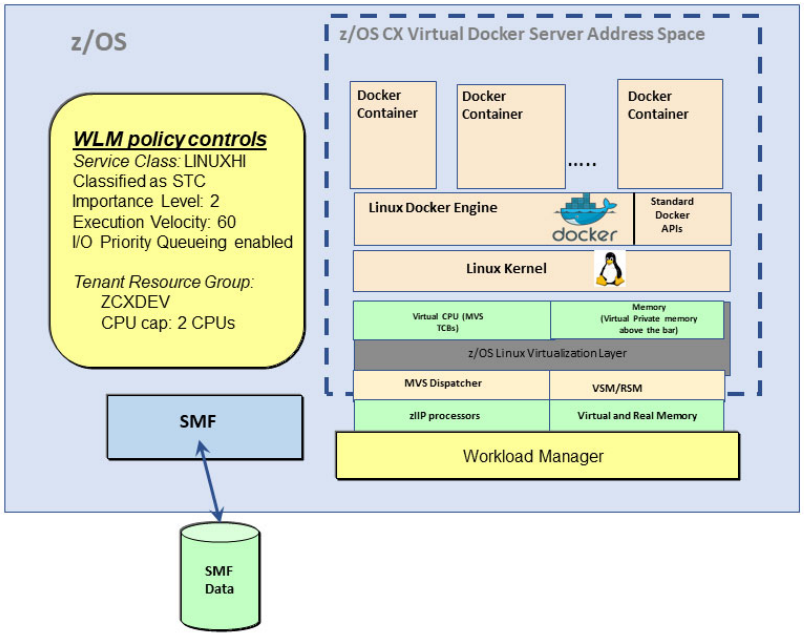


Figure 1-12 Availability of SMF data

Multiple z/CX instances can be deployed within a z/OS System. This allows for isolation of applications, different business/performance priorities (in other words, unique WLM service classes) and the capping of resources that are allocated to workloads (CPU, memory, disk, and so on). Each zCX address space has its own assigned storage, network, and memory resources. CPU resources are shared with other address spaces. Resource access can be influenced by configuration and WLM policy controls.

In summary, zCX provides “hypervisor-like” capabilities by using the z/OS Dispatcher, WLM, and VSM/RSM components to manage access to memory and CPU. The zCX virtualization layer manages the storage, network, and console access by using dedicated resources. There is no communication across zCX virtualization layer instances except through TCP/IP and disk I/O which are mediated by z/OS. Figure 1-13 provides more details.

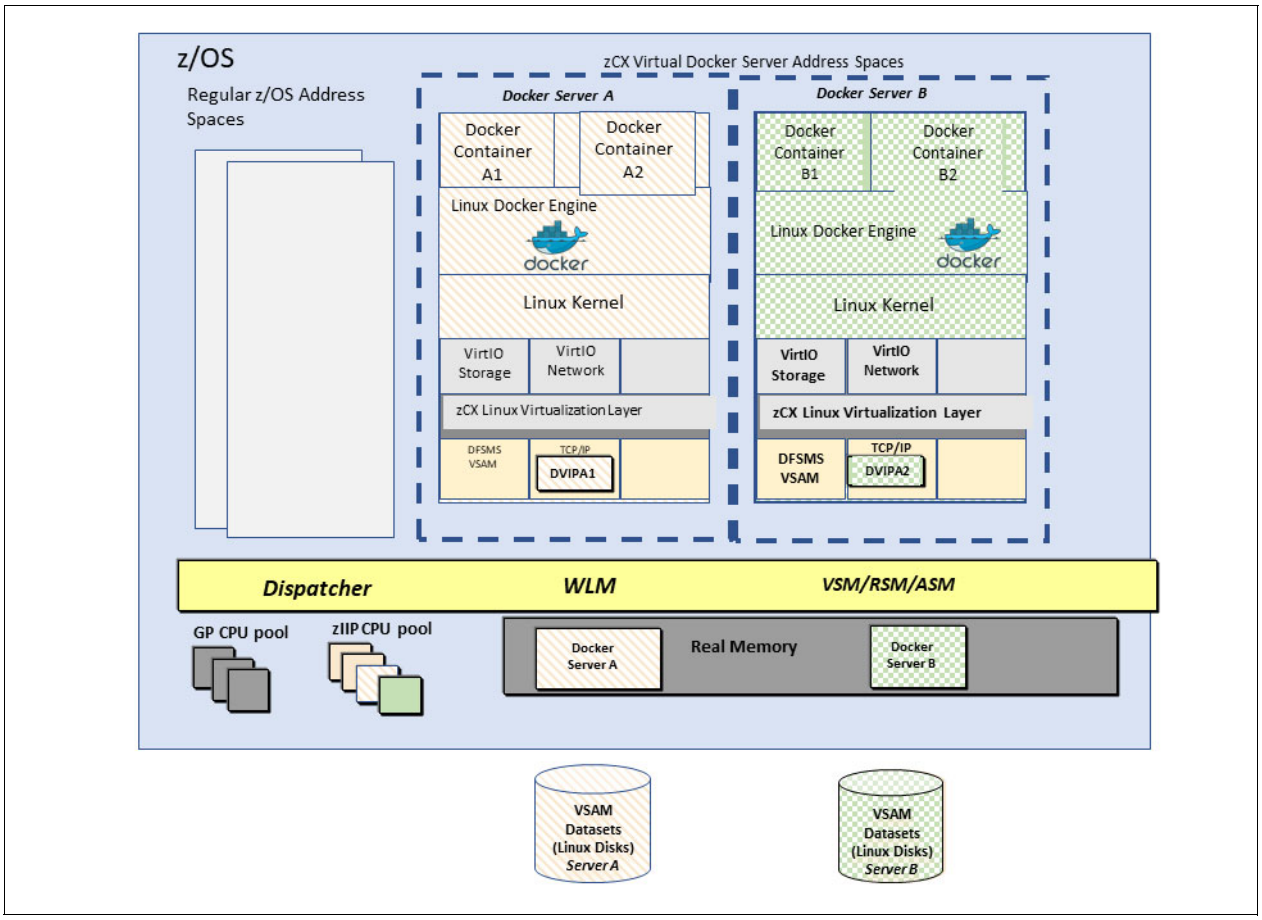


Figure 1-13 zCX "hypervisor-like" capabilities

1.4 Why use z/OS Container Extensions

Today's enterprises increasingly deploy hybrid workloads. In this context, it is critical to embrace agile development practices, leverage open source packages, Linux applications, and IBM and third-party software packages alongside z/OS applications and data. In today's hybrid world, packaging software as container images is a best practice. Today, zCX gives enterprises the best of both worlds, because with zCX you run Linux on Z applications — packaged as Docker containers — on z/OS alongside traditional z/OS workloads.

There are many reasons to consider using zCX, including these:

- You can integrate zCXworkloads into your existing z/OS workload management, infrastructure, and operations strategy. Thus, you take advantage of z/OS strengths such as pervasive encryption, networking, high availability, and disaster recovery.
- You can architect and deploy a hybrid solution that consists of z/OS software and Linux on Z Docker containers on the same z/OS system.
- You enjoy more open access to data analytics on z/OS by providing developers with standard OpenAPI-compliant RESTful services.

In this way, clients take advantage of z/OS Qualities of Service, Colocation of Applications and Data, Integrated Disaster Recovery/Planned Outage Coordination, Improved Resilience, Security, and Availability.

1.4.1 Qualities of service

IBM Z and z/OS are known for having robust qualities of service (reliability, availability, scalability, security, and performance). In today's business environment, as enterprises shift toward hybrid cloud architectures, taking advantage of these qualities helps them gain a competitive advantage.

Too often, hybrid cloud architectures span multiple computing platforms, each with different qualities of service, which leaves enterprises exposed to multiple points of failure. These exposures can lead to lost business, loss of revenue, and even tarnished reputations. The ability to run hybrid workloads within a single platform and OS can help to alleviate these exposures. With zCX, this ability can now become a reality:

- Because zCX runs under z/OS, it inherits many of the qualities of service of the z/OS platform.
- You are no longer required to manage each platform individually and then coordinate activities across those platforms.

1.4.2 Colocation of applications and data

By colocating applications and data, you can simplify many of the operational processes that would be required to manage workloads. Without colocation, events such as maintenance windows and moving workloads between CECs/sites requires great coordination efforts that include non-z/OS administrators. Such efforts add complexity and increase the possibility for errors.

For Tier 1 applications that span both z/OS and Linux, zCX offers the ability to better integrate disaster recovery and planned outage scenarios. For example, with IBM GDPS® you can take advantage of its automatic, integrated restart capabilities to handle site switches or failiures without having to coordinate activities across platforms.

Single points of failure can be eliminated by the use of HyperSwap for the VSAM data sets that zCX uses for storage. Support for Dynamic VIPAs allows for non-disruptive changes, failover, and dynamic movement of workloads.

The following figures depict some example scenarios for integrating Operations and Disaster Recovery.

In Figure 1-14, you would use automated operations facilities within a single LPAR to restart-in-place for both planned and unplanned outages.

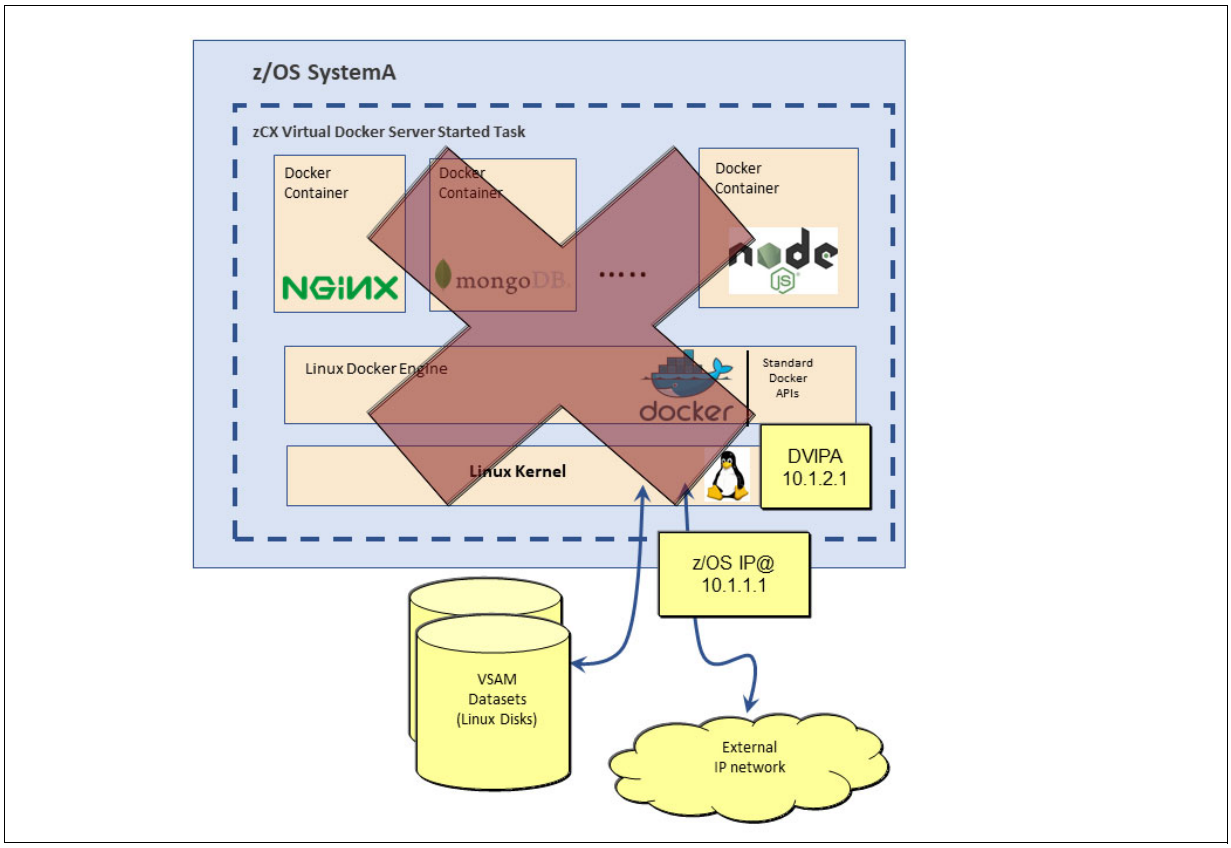


Figure 1-14 Operations disaster recovery scenario #1

Figure 1-15 shows how you could use automated operations facilities within a single LPAR to restart-in-place or restart on another LPAR within the sysplex for both planned and unplanned outages.

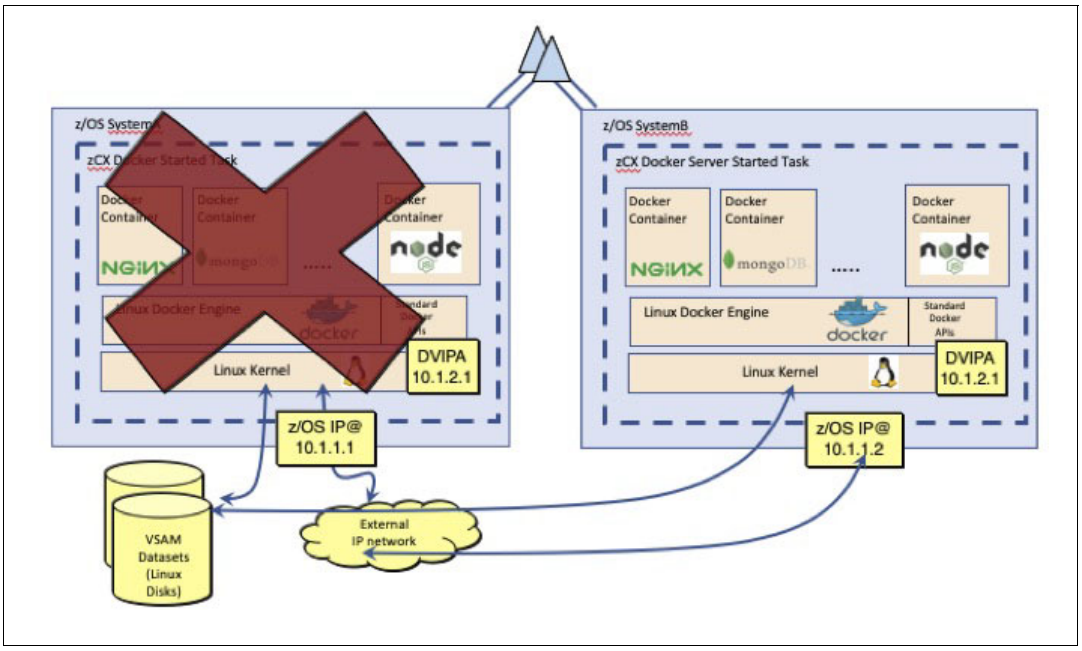


Figure 1-15 Operations disaster recovery scenario #2

In the scenario shown in Figure 1-16, you would use automated operations facilities within a single LPAR to restart-in-place, restart on another LPAR within the sysplex, or restart in an alternate site leveraging GDPS (or another automated DR framework) for both planned and unplanned outages.

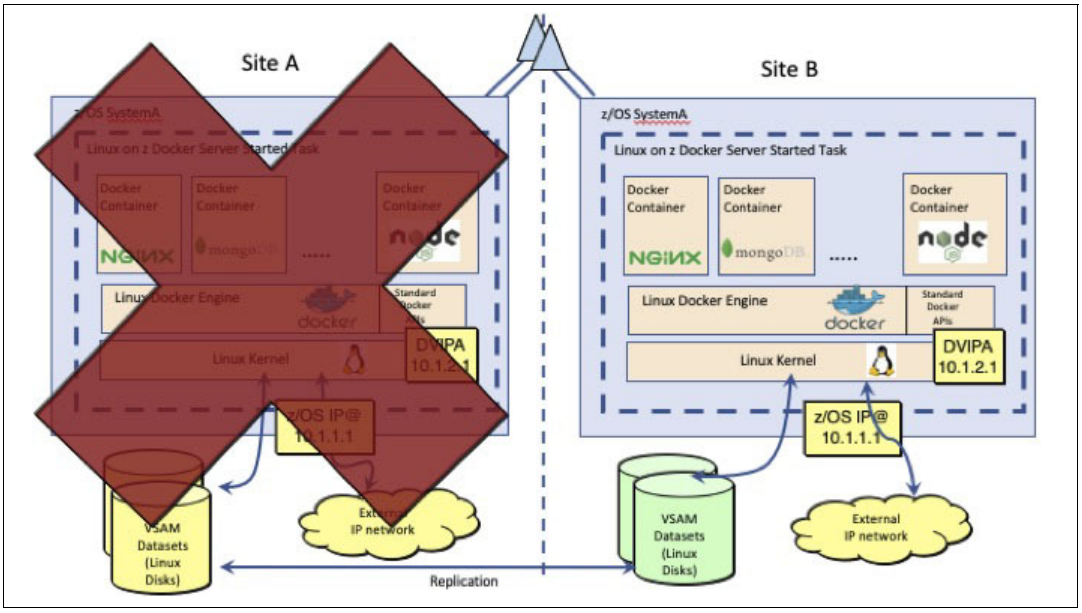


Figure 1-16 Operations disaster recovery scenario #3

1.4.3 Security

Data breaches are being reported at an increasing frequency for organizations large and small. Such breaches lead to a loss of trust for the organization and its customers. Organizations are challenged to secure their data, and also to show auditors and government regulators that customer data is secure (especially sensitive customer data that might be used for fraud or identify theft). As result, security is a major requirement for applications today.

Organizations are adopting container technologies to containerize their workloads and make them easier to deploy and test. Also, these technologies enable the migration of application workloads from one specific platform to another without needing code updates. Additionally, organizations are required to provide a higher level of security for mission-critical and highly sensitive data.

By building your container environment into zCX instances, you 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 blockchain applications and also cloud, mobile, big-data, and analytics applications.

It's well known that IBM Z servers deliver industry-leading security features and reliability, with millisecond response times, hybrid cloud deployment, and performance at scale. The world's top organizations rely on this platform to protect their data with a highly securable configuration to run mission critical workloads..

The zCX contains its own embedded Linux operating system, security mechanisms, and other features which can be configured to build a highly secure infrastructure for running container workloads. Through these benefits, zCX offers:

- Simplified, fast deployment and management of packaged solutions

- The ability to leverage security features without code changes

In the past, the manual way of individually installing applications and their dependencies could satisfy the needs of system administrators and application developers. However, the new business model (mobile, social, big data, and cloud) requires an agile development environment, speed to build, and consistency to quickly deploy complex applications. So, zCX comes with a preinstalled operating system and application codes that provide an environment that can be configured securely, in which to deploy your container workloads.

When correctly configured, a zCX appliance makes your cloud infrastructure more secure and efficient for your Docker deployments. While zCX is a new capability on z/OS, it is still a started task address space, subject to the normal z/OS address space level security constraints.

A zCX instance runs as a standard z/OS address space. Docker containers running inside a zCX instance run in their own virtual machine with its own virtual addresses and have no way to address memory in other address spaces. Access to data in other address spaces is only available through TCP/IP network interfaces or virtual disk I/O.

In a zCX environment, you have access to the Docker CLI and you can deploy new containers, start/stop containers, inspect container logs, delete containers, and so on. However, you are isolated within the environment of your zCX instance and cannot affect other zCX instances, unless your installation explicitly permits such access.

There are several aspects to the security setup for a zCX environment. These range from the typical IBM RACF® requirements for z/OS User ID's, Started Tasks, file access, and USS (UNIX System Services) access to managing access to the Docker CLI.

In Chapter 3, "Security - Overview" on page 39, we discuss this in further detail. In Chapter 4, "Provisioning and managing your first z/OS Container" on page 47 and Chapter 9, "zCX user administration" on page 219, we provide examples of implementing different security approaches.

1.4.4 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 to memory that is assigned to other applications that are running on the same box.

The IBM Z platform offers a high degree of scalability, with the ability to run many container machines on a single zCX instance. The ability to scale up allows you to even further consolidate your x86 workloads onto IBM Z without the need to keep multiple environments.

There are so many reasons to *containerize* an application, but the most important is with regards to business requirements. If you have workloads that are running on z/OS and part of the applications run on an x86, it makes sense to move your x86 workloads into a container under zCX. That way, the workloads are colocated and can take advantage of the qualities of service that IBM Z provides to an infrastructure. You can find examples of colocation in Chapter 8, "Integrating container applications with other processes on z/OS" on page 187.

With Docker, the application developers (or users) can pull the image from a repository and run a few command lines to get the entire application's environment ready and accessible in just a few minutes.

1.4.5 Example use cases

There are many use cases that could be made possible by zCX. The following list gives some examples.

> **Note:** This list does not constitute a commitment or statement of future software availability for zCX.

- Expanding the z/OS software ecosystem for z/OS applications to include support for,
  - The latest Microservices (Logstash, etcd, Wordpress, and so on)
  - Non-SQL databases (MongoDB, IBM Cloudant®, and so on)
  - Analytics frameworks (for example, expanding the z/OS Spark ecosystem, IBM Cognos® components)
  - Mobile application frameworks (example: MobileFirst Platform)
  - Application Server environments (for example, IBM BPM, Portal Server, and so on)
  - Emerging Programming languages and environments
- System Management components
  - System management components in support of z/OS that are not available on z/OS
  - Centralized databases for management
  - Centralized UI portals for management products, as in these examples:
    - IBM Tivoli® Enterprise Portal (TEPS)
    - Service Management Unite (SMU)
- Open Source Application Development Utilities
  - Complement existing z/OS ecosystem, Zowe, and DevOps tools
    Note: Zowe is an open source project within the Open Mainframe Project that is part of The Linux Foundation.
  - IBM Application Discovery server components
  - IBM UrbanCode® Deploy Server
  - Gitlab/Github server
  - Linux based development tools
  - Linux Shell environments
  - Apache Ant, Apache Maven

1.5 Additional considerations

Thanks to z/OS Container Extensions (zCX), z/OS environments can host Linux on Z applications as Docker containers for workloads that have an affinity for z/OS. z/OS Container Extensions does not replace traditional Linux on Z environments.

- If you are a client with Linux on Z installations, you might want to continue to run those installations because the gains from zCX might not be appreciable.
- If you are a z/OS client that used to, but no longer has a Linux on Z installation, you should consider zCX.
- If you are a z/OS client who has never had a Linux on Z installation, then zCX is a low-effort way to try Linux on Z.

2

z/OS Container Extensions planning

This chapter discusses the planning prerequisites that are necessary to enable z/OS Container Extensions. It also discusses planninng activities for containers that run in zCX instances.

This chapter describes the following aspects of planning:

- 2.1, “Prerequisites”
  - 2.1.1, “IBM Z hardware” on page 20
  - 2.1.2, “z/OS Version 2, Release 4” on page 24
  - 2.1.3, “z/OS Management Facility (z/OSMF)” on page 25
  - 2.1.4, “Base DASD requirements” on page 27
  - 2.1.5, “Planning for network connectivity” on page 31
- 2.2, “Planning for containers” on page 36
- 2.3, “Private registry considerations” on page 36
- 2.4, “Backup and recovery considerations” on page 36

© Copyright IBM Corp. 2019. All rights reserved.

2.1 Prerequisites

When considering zCX, you must plan for IBM Z hardware features as well as z/OS, the underlying operating system.

2.1.1 IBM Z hardware

The following IBM Z hardware types support the execution of zCX:

- IBM z14®, type 3906, with the GA2 driver level.
- IBM z14 ZR1, type 3907.
- IBM z15™, type 8561 model T01.

z14 and z15 Feature Code 0104

Ensure that the IBM Z processor, where you plan to run zCX, has hardware feature code 0104 (Container Hosting Foundation) included. If feature code 0104 is not installed on one of the previously mentioned machine types, it can be configured using the IBM e-config hardware configurator, and ordered through a fee-based MES. Figure 2-1 on page 20 shows you the panel from e-config, where you would select feature code 0104 for an IBM z14 system. For IBM z15 configurations, you would select the same feature code.

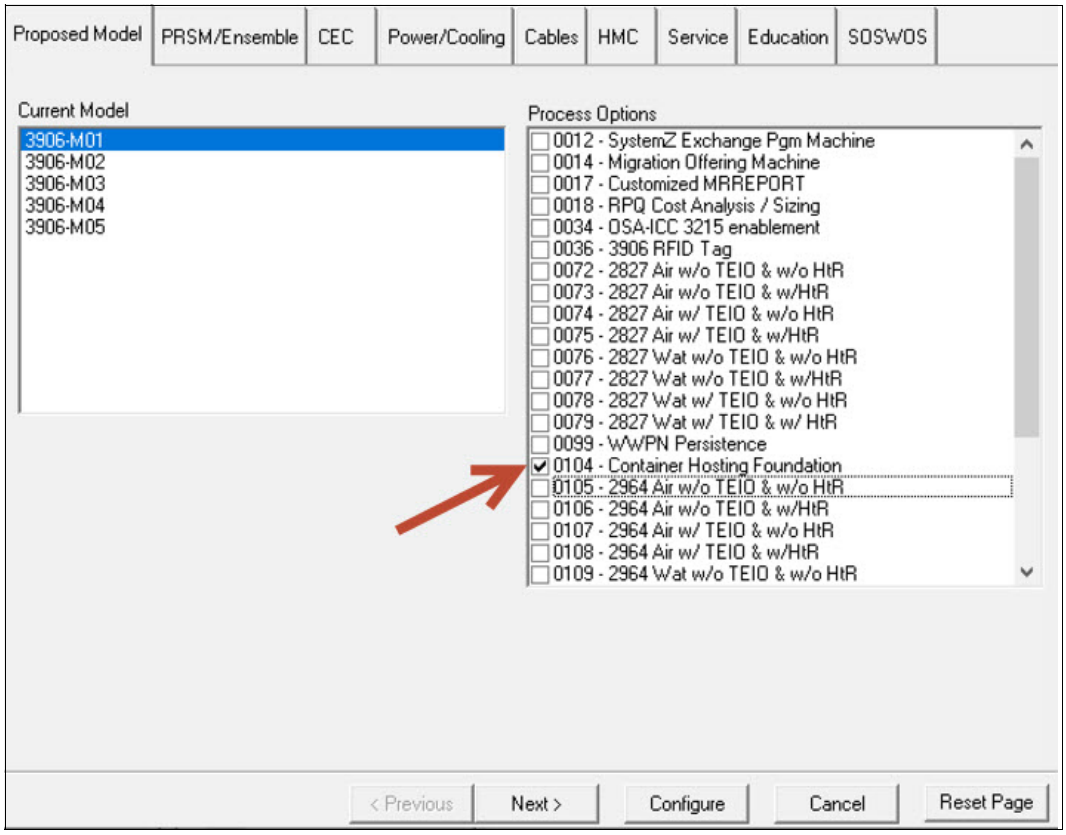


Figure 2-1 Configuring feature code 0104 for a z14 system on IBM e-config tool

CPU planning

To prepare the setup of your zCX environment, you must plan for processor usage. zCX instances can run on either general purpose or zIIP processors. One advantage of running zCX on zIIP processors is that it does not affect your software bill. The other advantage is,

that you can run zIIP processors in symmetric multithreading mode (SMT). When running in SMT mode, you can run two threads in parallel on a single zIIP. Check the settings of your processors with the operator command **D M=CPU.** You can see the output of this command in Example 2-1 on page 21. The core status tells you, that this LPAR is running in multithreading (MT) mode 1 for CPs (1 is the only option for CPs), and running in MT mode 2 for zIIPs.

Example 2-1 Output of D M=CPU operator command

```
D M=CPU
IEE174I 09.38.33 DISPLAY M 231
CORE STATUS: HD=Y   MT=2  MT_MODE: CP=1  zIIP=2
ID    ST   ID RANGE   VP  ISCM  CPU THREAD STATUS
0000   +   0000-0001  H   FC00  +N
0001   +   0002-0003  M   0000  +N
0002   +   0004-0005  LP  0000  +N
0003   +   0006-0007  LP  0000  +N
0004   +I  0008-0009  H   0200  ++
0005   +I  000A-000B  H   0200  ++
0006   +I  000C-000D  M   0200  ++
0007   +I  000E-000F  M   0200  ++
0008   -   0010-0011
0009   -   0012-0013
000A   -   0014-0015
000B   -   0016-0017
000C   -I  0018-0019
000D   -I  001A-001B
000E   -I  001C-001D
000F   -I  001E-001F

CPC ND = 008561.T01.IBM.02.00000002B7F8
CPC SI = 8561.716.IBM.02.000000000002B7F8
         Model: T01
CPC ID = 00
CPC NAME = ARIES
LP NAME = ARIES22    LP ID = 22
CSS ID  = 2
MIF ID  = 2

+ ONLINE    - OFFLINE    N NOT AVAILABLE    / MIXED STATE
W WLM-MANAGED

I        INTEGRATED INFORMATION PROCESSOR (zIIP)
CPC ND  CENTRAL PROCESSING COMPLEX NODE DESCRIPTOR
CPC SI  SYSTEM INFORMATION FROM STSI INSTRUCTION
CPC ID  CENTRAL PROCESSING COMPLEX IDENTIFIER
CPC NAME CENTRAL PROCESSING COMPLEX NAME
LP NAME  LOGICAL PARTITION NAME
LP ID    LOGICAL PARTITION IDENTIFIER
CSS ID   CHANNEL SUBSYSTEM IDENTIFIER
MIF ID   MULTIPLE IMAGE FACILITY IMAGE IDENTIFIER
```

For core mode, which is specified in the LOADxx member of SYSn.IPLPARM (see extract from LOADxx member in Example 2-2 on page 22), the `D M=CPU` operator command is internally changed to `D M=CORE`. If a system is running in core mode, Hiperdispatch is also required (HIPERDISPATCH=YES in IEAOPTxx parmlib member).

Example 2-2 Extract of SYS0.IPLPAM(LOAD01)

```
NUCLEUS  1
NUCLST   XX
IEASYM   XX
*-----------DEFINITION FOR SC74-------------*
HWNAME   CETUS
LPARNAME CETUS22
SYSPLEX  PLEX75   Y
IODF     ** SYS6     ITSO     01 Y
SYSCAT   BH5CAT123CMCAT.BH5CAT
PARMLIB  SYS1.PARMLIB
PARMLIB  SYS1.IBM.PARMLIB
PROCVIEW CORE,CPU_OK
HWNAME   ARIES
LPARNAME ARIES22
SYSPLEX  PLEX75   Y
IODF     ** SYS6     ITSO     01 Y
SYSCAT   BH5CAT123CMCAT.BH5CAT
PARMLIB  SYS1.PARMLIB
PARMLIB  SYS1.IBM.PARMLIB
PROCVIEW CORE,CPU_OK
```

In our installation, the z/OS LPARs are defined in core mode, where you can switch between one and two cores for your zIIP processors. If your core status for zIIP processors is zIIP=1, modify your IEAOPTxx parmlib member to MT_ZIIP_MODE=2 (see extract from our IEAOPT00 parmlib member in Example 2-3).

Example 2-3 Extract from our SYS1.PARMLIB(IEAOPT00)

```
CPENABLE=(10,30)
IIPHONORPRIORITY=YES
HIPERDISPATCH=YES
MT_CP_MODE=1
MT_ZIIP_MODE=2
```

Check with your capacity planning colleagues to find what the optimum MT mode is for your environment. The settings affect all zIIP processors that are defined to a specific z/OS LPAR.

We recommend that you do not over-allocate processors (CPs or zIIPs). Do not define more virtual processors to a single zCX instance than you have available to your z/OS system. If you have zIIP processors active and run them in MT 2 mode, you can define two cores per zIIP processor. In Example 2-1 on page 21, you see this on the CPU THREAD STATUS column, when “++” shows up as a status. Use the reconfiguration workflow to adjust the number of guest CPUs if necessary, and then restart your instance.

In the IEAOPTxx parmlib member, you can specify the parameter `IIPHONORPRIORITY`, which controls the flow of workload to a zIIP processor. If it is set to **YES** (default, see also in Example 2-3 on page 22), workload spills over to CP processors when zIIP processors are unavailable. You can set it to **NO,** but then resource contention might occur.

A more granular approach for different zCX address spaces is to define dedicated service classes in WLM. In the service class menu of WLM, the *Honor Priority* field can contain either DEFAULT (YES) or NO. The red arrow in Figure 2-2 on page 23 shows you this field in the panel in the z/OSMF WLM application.

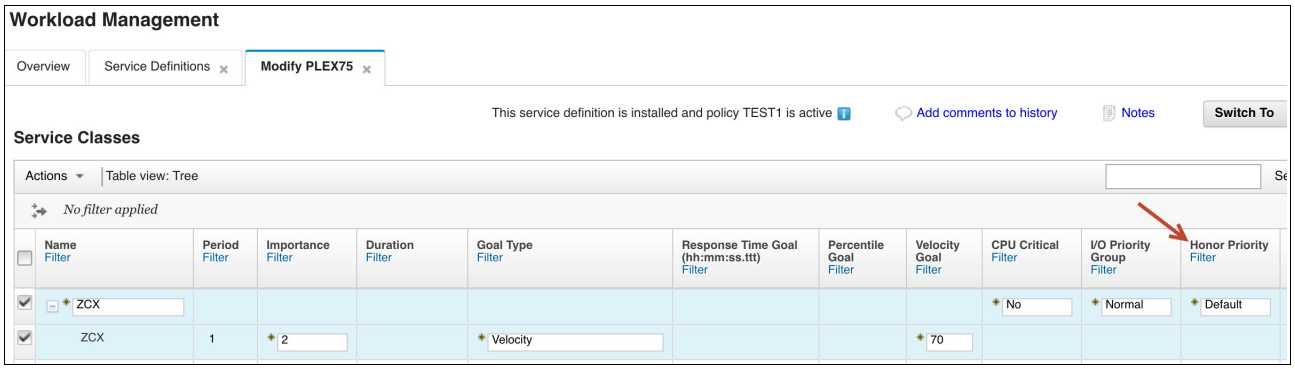


Figure 2-2 WLM service class definition menu with Honor Priority

The Honor Priority setting in a WLM service class overrides the system-wide setting for address spaces that are defined to that specific service class.

Memory planning

When you plan your zCX environment, you must plan for memory allocation, too. Each zCX instance must be assigned a minimum of 2-GB-fixed, private memory to run. If you want to avoid memory constraints, define more memory. You define the amount of memory during execution of the provisioning workflow and allocate it during the start of a zCX instance. The amount of memory that you use in a zCX instance depends on the size and number of containers that run in that instance. If the containers inside a zCX instance need more virtual memory than expected, paging to the defined swap disks takes place on the Linux level, inside the zCX instance. This might lead to performance impacts. Use the reconfiguration workflow to adjust the amount of memory, and then restart your instance.

Use monitoring tools like Grafana to analyze the memory usage of your containers, and adjust the amount of fixed memory, if needed. For details on these aspects of monitoring, see “Configuring Grafana to monitor zCX containers” on page 166 and “Monitoring with Grafana on container level” on page 177.

Resource capping for zCX address spaces

If needed, you can optionally decide to limit the amount of CPU and real memory that zCX address spaces can consume. You do this through the use of WLM Resource groups or Tenant Resource groups. Figure 2-3 on page 23 shows you the menu for WLM resource group definitions and Figure 2-4 on page 24 shows you the menu for WLM Tenant Resource group definitions.

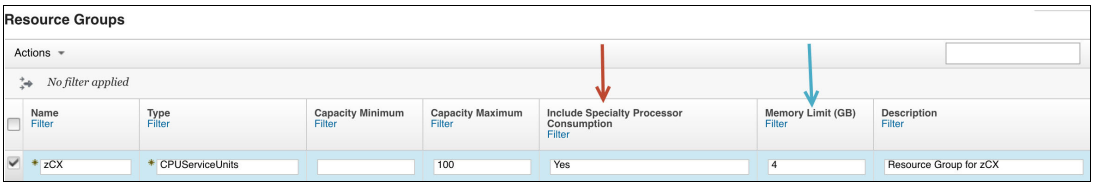


Figure 2-3 WLM menu for Resource group definitions for zCX

The red arrows in Figure 2-3 on page 23 and in Figure 2-4 on page 24 point to the *Include Speciality Processor Consumption* attribute. This attribute ensures that the maximum capacity specified for zCX instances includes CP and zIIP usage.

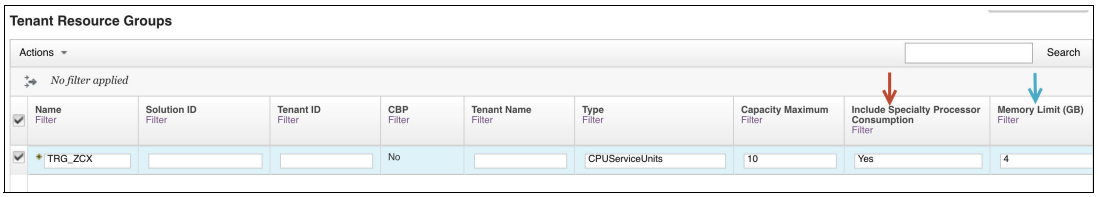


Figure 2-4 WLM menu for Tenant Resource group definitions for zCX

You can also cap real memory, using Resource groups or Tenant Resource groups. You do this by specifying the amount of real storage that service classes associated with a Resource Group or Tenant Resource group can consume. The blue arrows in Figure 2-3 on page 23 and Figure 2-4 on page 24 show you the fields where you would enter the memory limit in GB.

If you use this WLM-based technique, plan for the following:

For Resource groups:	Specify the name of the associated Resource group for the service class associated with the zCX address space.
For Tenant Resource groups:	Define an associated Tenant report class when you specify the Tenant Resource group. WLM classification rules must specify the Tenant report class association for the zCX address space. Figure 2-5 shows the screen in z/OSMF WLM application. The red arrow points to corresponding Tenant resource group.

Figure 2-5 WLM Tenant report class definition for zCX

You can find details on how to define WLM Resource groups and Tenant Resource groups, and what to avoid, in the following IBM Redbooks publication: *z/OS MVS Planning: Workload Management*, SC34-2662.

When you intend to use capping of CPU and memory resources for zCX, we recommend that you perform a detailed capacity planning and performance analysis to avoid any performance impacts on your systems.

2.1.2 z/OS Version 2, Release 4

z/OS V2R4 is the first release where you can run containerized applications by using z/OS Container Extensions (zCX). There are no plans to roll back zCX to a previous release. As a result, each system in a sysplex and any monoplex system where you plan to deploy zCX must be on z/OS version 2, release 4.

zCX is shipped in the z/OS BCP base, and it contains two FMIDs:

HBB77C0	This FMID contains the virtualization layer for zCX. The three character prefix is GLZ. When zCX writes messages to the z/OS SYSLOG or OPERLOG, they start with this prefix.

HZDC7C0	This FMID contains the underlying Linux kernel, the Docker Engine, and the z/OSMF workflows that you need later for provisioning, deprovisioning, and other maintenance or reconfiguration activities of container instances.

2.1.3 z/OS Management Facility (z/OSMF)

When you plan to use zCX, you need to be familiar with z/OSMF. For tasks like the provisioning or deprovisioning of zCX instances, or even maintenance activities for zCX instances, you must use the z/OSMF workflow engine.

Tip: If you have never used z/OSMF before, you must do a setup for the zCX-related activities. Starting with z/OS V2R4, you can perform a fast setup with the z/OSMF Lite configuration. It gives you a very granular way of configuring z/OSMF, meaning that you configure just the z/OSMF nucleus, and then add only the core services you need for zCX.

Besides the configuration of the z/OSMF Workflows task, the following services also must be configured:

- Common Event adapter (CEA). This z/OS component has to be configured in full function mode.
- z/OSMF Notification services.
- z/OSMF Settings services.
- z/OS jobs REST services.
- z/OS data set and file REST services.
- TSO/E address space services.

You can find the details for the z/OSMF setup in the following IBM Redbooks publication: *IBM z/OS Management Facility, SC27-8419.*

z/OSMF workflows for zCX

zCX provides you with a number of workflows that you must run in z/OSMF depending on your activities. You can find the XML-based workflow definitions in your z/OS UNIX System Services environment under `/usr/lpp/zcx_zos/workflows`.

Table 2-1 on page 25 shows you the available workflows and their meaning.

Table 2-1 z/OSMF workflows for zCX

Workflow name	Workflow type	Workflow content
provision.xml	Provisioning workflow	Use this workflow to provision a zCX instance into a running z/OS system.
backup_config.xml	Backup Configuration workflow	Use this workflow to generate a backup of your existing zCX instances. If needed, use the **restore** configuration workflow to reapply the configuration to a specific zCX instance.
reconfigure.xml	Reconfiguration workflow	Use this workflow to update an existing zCX instance. In terms of workflow steps, the reconfiguration workflow is similar to the provisioning workflow. You can modify the variable settings, for example, perform a fine-tuning of your zCX instance. When the workflow finishes, stop the zCX instance, that you have reconfigured, and restart it again, to get the modifications active.
restore_config.xml	Restore Configuration workflow	Use this workflow to restore a zCX appliance configuration that you earlier backed up using the backup configuration workflow.

Workflow name	Workflow type	Workflow content
add_data_disks.xml	Add data disks workflow	Use this workflow when your zCX instances need more disk space for swap data and user data. During execution of this workflow, new VSAM linear data sets are allocated and formatted. You can run this workflow against an active zCX instance, but to use the additional space, you must restart the instance.
upgrade.xml	Upgrade workflow	Use this workflow to apply service and maintenance to an existing zCX instance. During execution of this workflow, an alternate root VSAM linear data set is allocated.
rollback.xml	Rollback workflow	Use this workflow to roll back a maintenance level for an existing zCX instance, in the event that you are suffering any issues with the new maintenance level. During execution, the original root VSAM data set — which has been put aside during execution of the upgrade workflow — is restored. After rollback, stop your zCX instance, and restart it again.
deprovision.xml	Deprovisioning workflow	Use this workflow to deprovision an existing zCX instance. During execution, all VSAM data sets and the instance zFS file system are deleted.

All z/OSMF workflows in Table 2-1 first retrieve and gather the zCX instance properties. On that basis, they perform the subsequent steps either manually or through automation. We suggest that you run the steps through automation, because some of them might take a few minutes to complete.

In z/OSMF, do not make changes to the xml files, because they are managed by SMP/E. Therefore, any modifications that you make to the files are subject to being overwritten.

Besides the xml files with the workflow contents in `/usr/lpp/zcx_zos/properties`, you find a workflow variable file, which you can use as a template. Copy this template to your own home directory, and fill in the variables with values that are appropriate for your zCX instances. Template files can save time after you know the configuration. You supply your customized template file whenever you provision, thus avoiding the need to type all the values in.

z/OSMF workflows for manually starting and stopping zCX instances

In Table 2-2 on page 26, we describe two additional workflows that you can use for manual start and stop actions of your existing zCX containers. You can find the xml file in `/usr/lpp/zcx_zos/workflows/actions`.

Table 2-2 z/OSMF workflows for manual start and stop actions

Workflow name	Workflow type	Workflow description
start_instance.xml	Start instance workflow	Use this workflow to start an already provisioned zCX instance. This workflow uses z/OSMF console REST services in the background to send out the **start** command.
stop_instance.xml	Stop instance workflow	Use this workflow to stop an already provisioned zCX instance. This workflow uses z/OSMF console REST services in the background to send out the **stop** command.

If you plan to use the start and stop workflows, that we describe in Table 2-2 on page 26, review the chapter on z/OS console services in the following IBM Redbooks publications: *IBM z/OSMF Programming Guide*, SC27-8420.

2.1.4 Base DASD requirements

When you plan for zCX instances, you must keep in mind that you need a reasonable amount of z/OS DASD space. This DASD space must reside on ECKD-based 3390 volumes being initialized and accessed by z/OS. The VSAM linear data sets for your zCX instances are automatically allocated when you run the `provision.xml` workflow. This workflow is described in Table 2-1 on page 25. In Table 2-4 on page 27, we describe the different data set types, their minimum sizes, and their use.

Example 2-4 Data set planning requirements for a zCX instance

Data set type	Usage	Size requirements	Addition of space possible?
Root disk	Linux root file system	>=4GB	No; run workflow deprovision.xml and then provision.xml with different size
Configuration disk	This disk holds configuration data for zCX appliance instances	>=2MB	No; run workflow deprovision.xml and then provision.xml with different size
User data disk	This disk holds all Docker images, containers, logs, and volumes	This size is workload dependent, but plan for >=20GB	Yes, run workflow add_data_disks.xml and restart your zCX instance
Swap data disk	These disks are optionally used by the Linux kernel for paging and swapping activities when virtual memory exceeds the real memory	This size is workload dependant, but plan for >=2GB	Yes, run workflow add_data_disks.xml and restart your zCX instance
Diagnostics and log data disk	This disk holds diagnostic data, logs, and first failure data capture (FFDC) information	>-1GB	No; run workflow deprovision.xml and then provision.xml with different size
Instance directory zFS	This disk holds the zCX appliance image, configuration file, and FFDC information	>=4GB	This VSAM data set can be expanded by secondary extents

Based on the information in Table 2-4, you can see that the minimum space that is needed for a single zCX instance is 33 GB. If you plan to set up more instances, multiply the number of instances by a minimum of 33 GB. The greatest amount of DASD space is used for the data disk (>= 20GB). For planning purposes, we recommend that you prepare a number of SMS-managed volumes for your zCX environment. The volumes should be 3390 model 54. In our configuration, this means 54 * 1,113 cylinders, which is 60,102 cylinders per volume. This is approximately 51 GB per volume.

Characteristics for the zCX instance VSAM linear data sets

When you plan for the data sets for a zCX instance, ensure that they meet the following requirements:

- The data sets must be allocated as VSAM linear data sets. This is performed by the `provision.xml` workflow.
- The data sets are allocated with primary extents only. The allocations take place at the time of instance provisioning.
- In ISMF, create a new DATACLAS construct or reuse an existing one. Define as “Extended Format Required”, and “Extended Addressability Enabled” to allow data sets to be allocated that can have a size larger than 4 GB. Figure 2-6 on page 28 shows you these definitions.

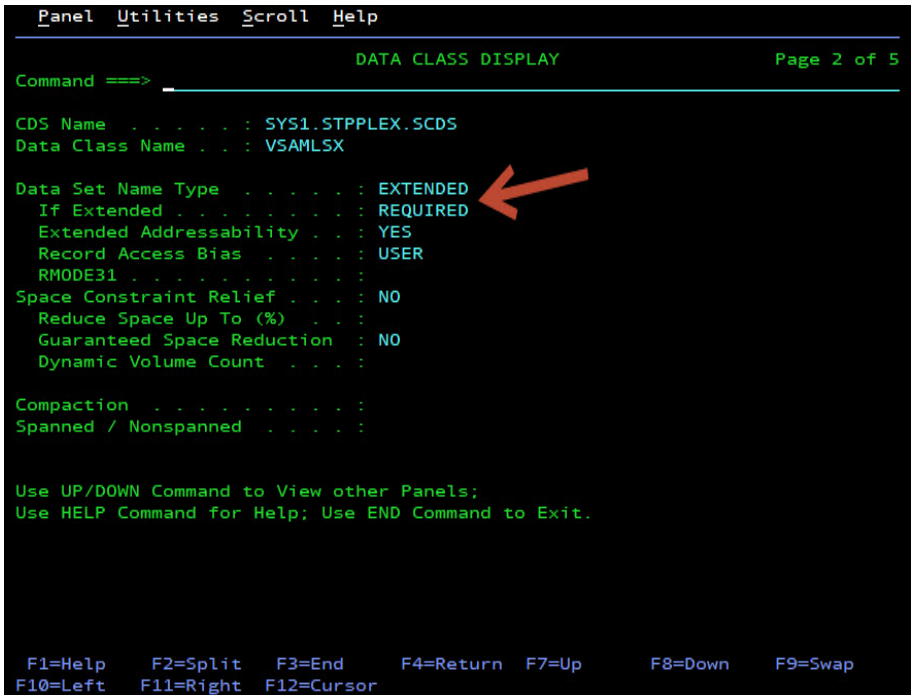


Figure 2-6 SMS Dataclass Construct for extended format and extended addressability

- In ISMF create a new STORCLAS construct or reuse an existing one to ensure that the VSAM linear data sets become SMS managed. Only the storage class needs to be defined. There is no requirement for special parameterization.
- There is no need to use a MGMTCLAS construct for backup and expiration considerations.
- Depending on your SMS rules, create a new STORGRP construct or reuse an existing one where you put all your 3390 volumes for the zCX data sets. We recommend that you create a dedicated volume pool for all of the zCX volumes that you plan to use.
- Define SMS ACS routines to associate your zCX data sets to the correct data class, storage class, and storage group. Example 2-5 shows you how to code the SMS ACS routine for a data class. Your storage management colleagues might want to perform these steps based on your information.

Example 2-5 Example of an SMS data class routine for zCX data sets

```
PROC DATACLAS
FILTLIST ZCX         INCLUDE(ZCX.**)
/*                                                                    */
/**********************************************************************/
/*                                                                    */
  SELECT
  WHEN (&DSN EQ &ZCX)
      DO
        WRITE 'DSORG = ' &DSORG
        IF &DSORG = 'VS' THEN
          SET &DATACLAS EQ 'VSAMLSX'
        ELSE
          SET &DATACLAS EQ ''
      END

    END
  END
```

Characteristics for a zCX instance zFS file system

You also must allocate one zFS file system for each of your zCX instances. This is used to store the zCX appliance image and configuration data. In a sysplex environment, we recommend that you mount it under a mount point within the `/global` directory. In this way, you provide access to that file system from all participating systems. The required size for the file system is also >= 4GB.

Planning for data set naming conventions

Naming conventions for the VSAM data sets are allocated during the execution of the `provision.xml` workflow, and these conventions require planning by you. Keep the following requirements in mind:

- The high-level qualifier (HLQ) of those data sets can contain up to 28 characters including the periods between the qualifiers.
- The `provision.xml` workflow allocates those data sets by using that HLQ in the format `HLQ.zcxinstancename.suffix`.
  - `zcxinstancename` is the jobname that you choose in the workflow, which you later use to start the zCX instance.
  - `suffix` is generated by the provisioning workflow and points to the function of the data set. The following data sets are provisioned (data set names are from our test environment):

    ```
    ZCX.REDB.ZCXRJ01.CONF
    ZCX.REDB.ZCXRJ01.DATA1
    ZCX.REDB.ZCXRJ01.DLOG1
    ZCX.REDB.ZCXRJ01.ROOT
    ZCX.REDB.ZCXRJ01.SWAP1
    ZCX.REDB.ZCXRJ01.ZFS
    ```
- If you plan to run different zCX instances (like production, development, or test) in a single z/OS system, you might want to distinguish the data sets by putting PROD, DEV, or TEST into the HLQ part of the data set names.

Figure 2-7 and Figure 2-8 on page 30, show you the structure of the data sets and their naming conventions.

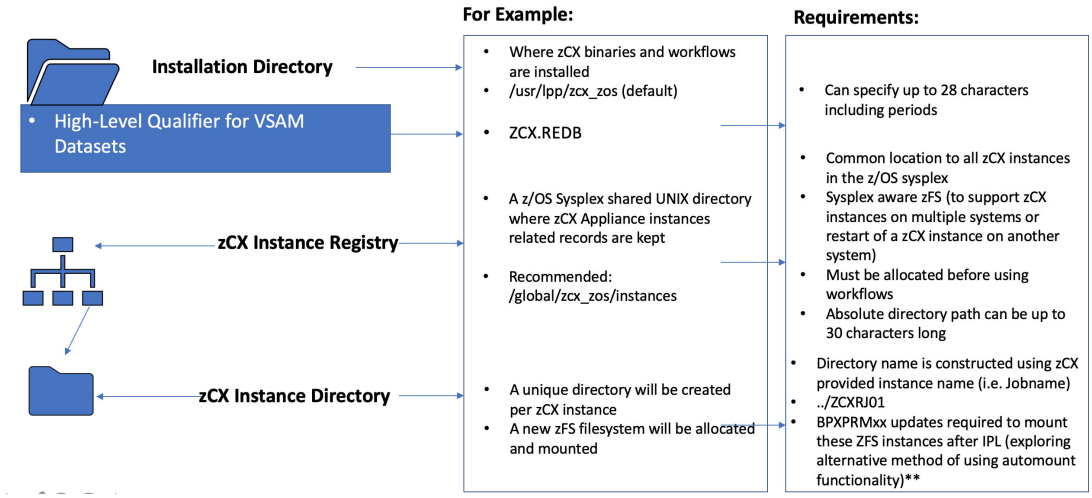


Figure 2-7 zCX instance data set and zFS structure, part 1

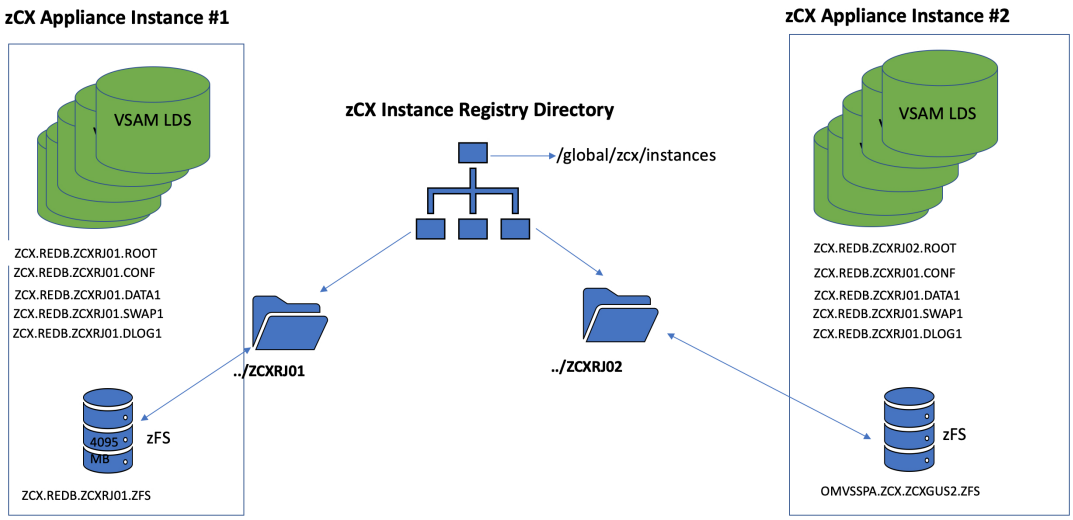


Figure 2-8 zCX instance data set and zFS structure, part 2

The zCX instance has its own directory. This is created during execution of the provisioning workflow. During the execution of this workflow, the zFS file system that belongs to the new instance is also allocated and mounted here. To ensure that the file system also is mounted after subsequent IPLs of your z/OS system, you must update your BPXPRMxx member of SYS.PARMLIB with the name of the zFS file system and its mount point. Example 2-6 shows you how to find out the details of your mounted zFS file systems.

Example 2-6 Display OMVS operator command to check for the zCX instance file system

```
D OMVS,F
BPXO045I 10.30.33 DISPLAY OMVS 033
OMVS     0011 ACTIVE             OMVS=(74,4A)
TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
ZFS           113 ACTIVE                      RDWR  09/05/2019  L=56
  NAME=ZCX.REDB.ZCXRJ01.ZFS                         19.17.48    Q=56
  PATH=/global/zcx/instances/ZCXRJ01
  NOSETUID
  OWNER=SC74     AUTOMOVE=Y CLIENT=N
```

You can take the information from the D OMVS,F shown in Example 2-6, to prepare the **mount** statement that you define in your BPXPRMxx member. Example 2-7 gives you the syntax for such a statement.

Example 2-7 Mount statement in SYS1.PARMLIB(BPXPRMxx) for zCX instance file system

```
MOUNT FILESYSTEM('ZCX.REDB.ZCXRJ01.ZFS') TYPE(ZFS)
      MODE(RDWR) MOUNTPOINT('/global/zcx/instances/ZCXRJ01')
      AUTOMOVE
```

Planning for a zCX started procedure

You must define a started procedure to run your zCX instances, after they have been provisioned. SYS.IBM.PROCLIB provides you with a sample procedure, called GLZ, that you must copy to system's proclib.

2.1.5 Planning for network connectivity

The z/OS Linux virtualization layer gives virtual access to the z/OS TCP/IP network through use of the **`virtio`** Linux interface. Figure 2-9 on page 31 shows you the general overview of the zCX virtualization layer that exists inside the zCX virtual Docker server address space.

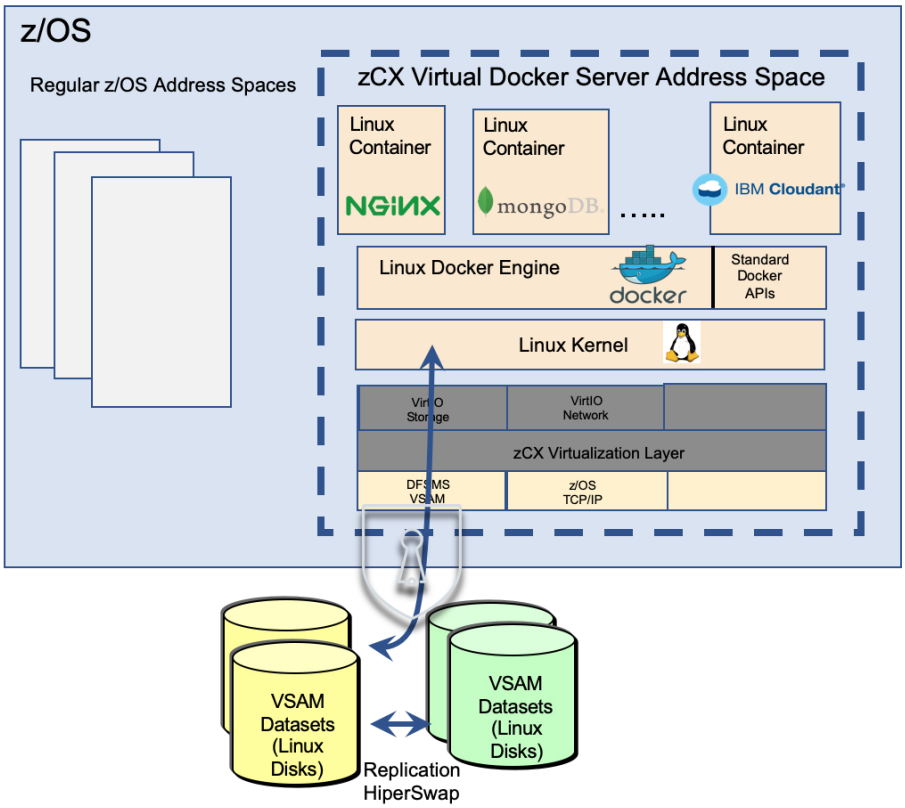


Figure 2-9 z/OS Linux virtualization layer

In Figure 2-10 on page 32 a red arrow points to the dynamic virtual IP address (DVIPA), named 10.1.2.1 for your Docker server inside a single zCX address space. This is a z/OS owned, managed, and advertised DVIPA. In contrast to a static IP address, the DVIPA allows you to restart your zCX instance on another z/OS system in your sysplex. This capability is an advantage during planned or unplanned z/OS system outages.

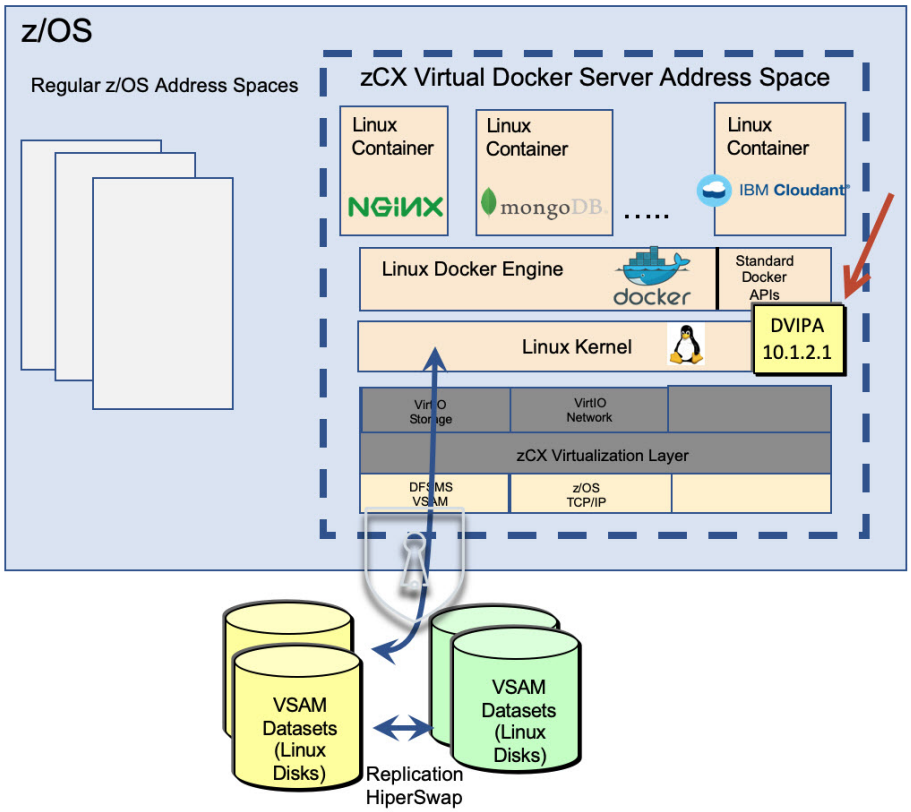


Figure 2-10 z/OS TCP/IP DVIPA for zCX

zCX introduces a new application instance DVIPA type called zCX. You must create this with a VIPARange statement in your TCP/IP configuration. Example 2-8 on page 32 shows you the syntax. In a regular TCP/IP configuration, you see more DVIPA definitions. This example shows only the syntax. At the end of the statement, you must define a new attribute in the VIPARANGE DEFINE statement in z/OS V2R4, which is `ZCX`.

Example 2-8 DVIPA definition for one zCX instance in TCP/IP profile

```
VIPADYNAMIC
VIPARANGE DEFINE 255.255.255.255 129.40.23.13 ZCX ;
ENDVIPADYNAMIC
```

The following rules apply for the VIPARANGE statements:

- You can define multiple DVIPAs on a single VIPARANGE statement, as long as they belong to the same subnet.
- Prepare a DVIPA in your provisioning workflow that matches the VIPARANGE definition your TCP/IP profile.
- If you have a sysplex running and plan to move zCX instances across your participating z/OS systems, include the same VIPARANGE statements into all of your TCP/IP profile definitions.

In z/OS, zCX uses cross memory functionality to implement high-performance network access between z/OS applications and Linux Docker containers. The blue arrow between the regular z/OS address space on the left side and the zCX virtualization layer on the right side in Figure 2-11 on page 33 marks this connection.

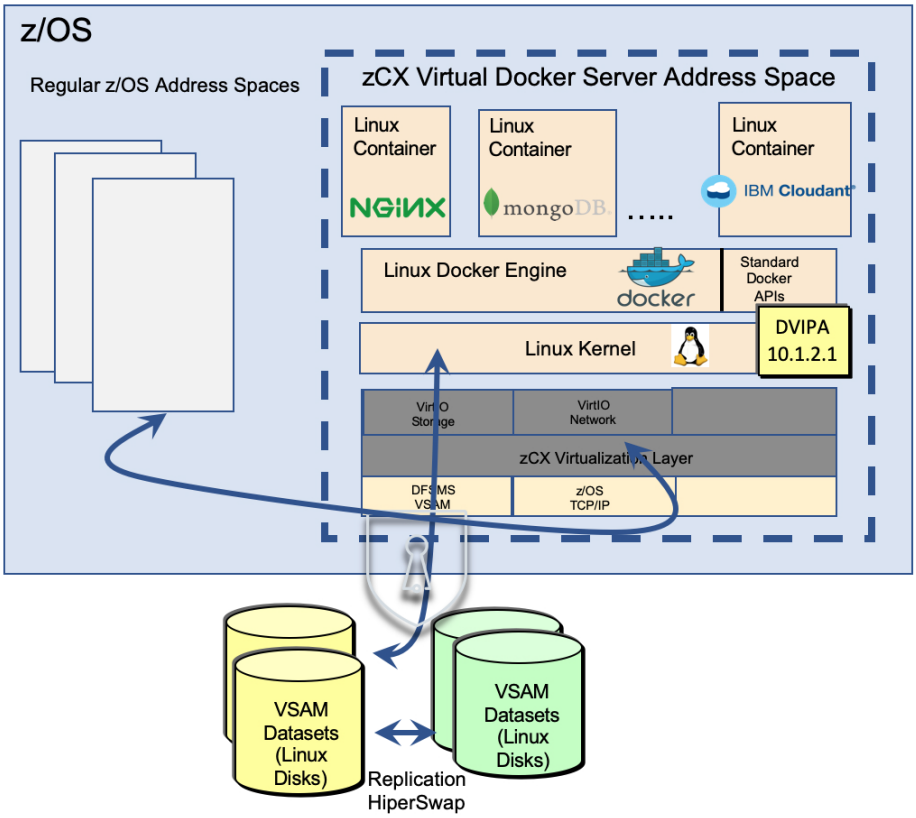


Figure 2-11 Cross memory network connection between z/OS applications and the zCX virtualization layer

The z/OS TCP/IP stack is used as a forwarder to and from zCX for access to the external TCP/IP network. Access to the external network is implemented through the z/OS TCP/IP stack. You might want to restrict access in your installation to and from the zCX instances. Therefore, your networking team might establish z/OS IP filters to prevent unauthorized network traffic.

The red arrow in Figure 2-12 on page 34 shows you the z/OS TCP/IP address, which is configured for the z/OS system itself.

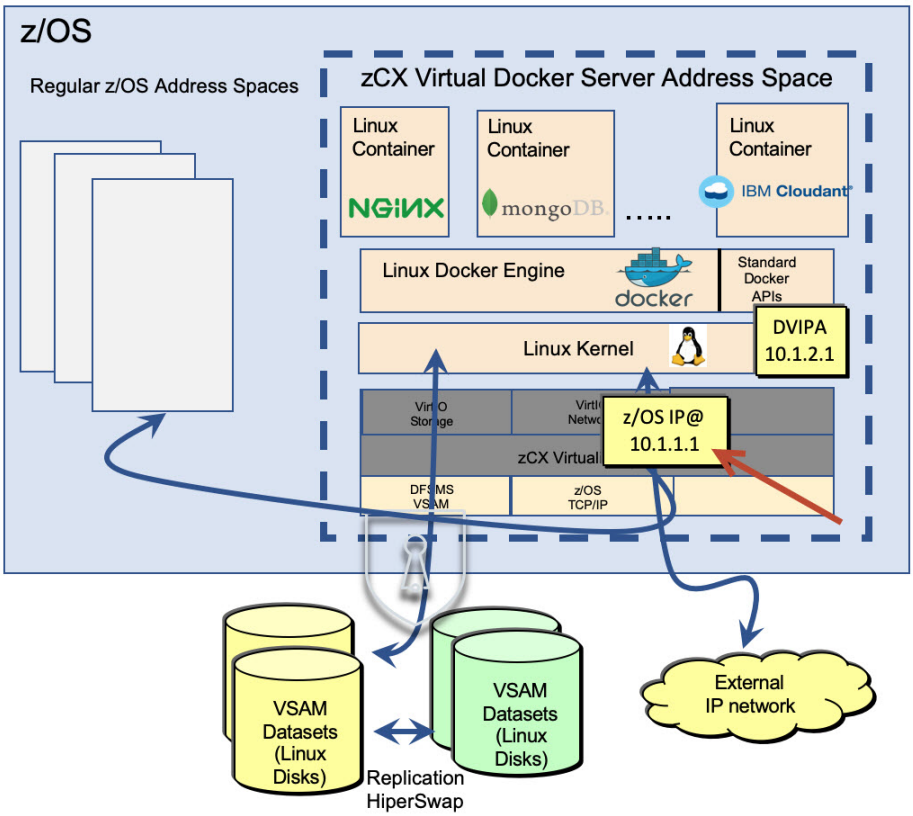


Figure 2-12 z/OS TCP/IP connection to and from and external network

When the SAMEHOST interface EZASAMEMVS is created, an EZAZCX interface is also created. This is done automatically when you use DynamicXCF in your TCP/IP profile. Example 2-9 shows you how we defined this statement. Get in touch with your networking team to figure out the correct settings for your sysplex environments.

Example 2-9 DynamicXCF definition in TCP/IP profile

```
IPCONFIG DYNAMICXCF 10.1.100.74 255.255.255.0 1
```

Figure 2-13 on page 35 shows you the SAMEHOST high-speed virtual IP network for your zCX instances. It is internally used for TCP/IP communication across zCX instances in a single z/OS system. The red arrow points to the EZAZCX interface.

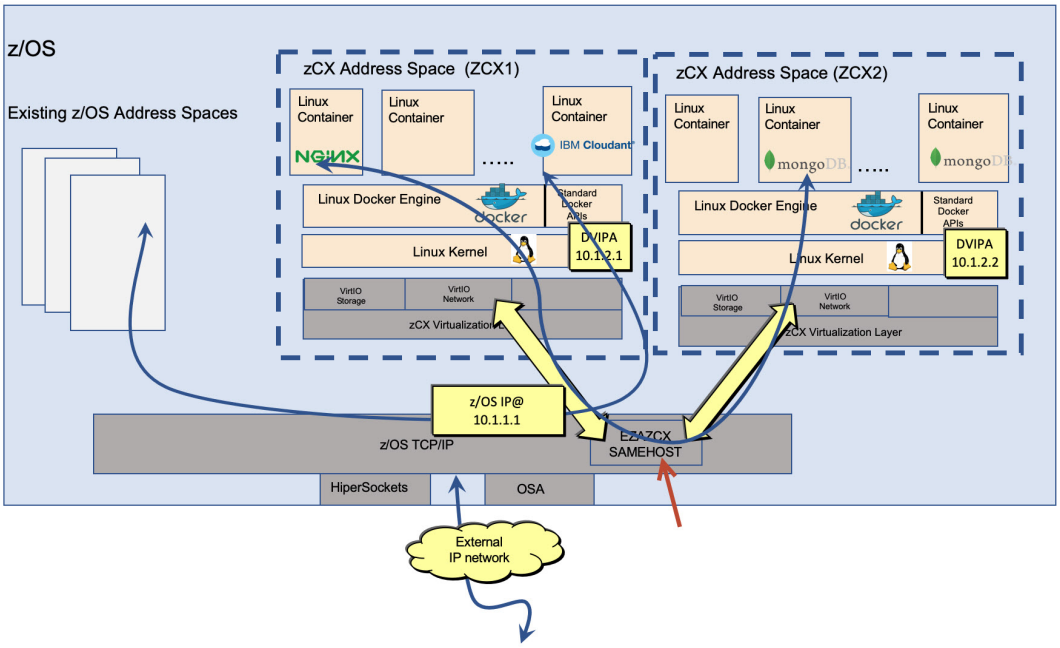


Figure 2-13 High-speed virtual IP communication across zCX instances in the same z/OS system

When you define DynamicXCF, the EZAZCX interface is automatically set up. For this reason, you can check its status only when executing the operator command:
`D TCPIP,TCPIP,N,DEVL,INTFN=EZAZCX.`

Example 2-10 on page 35 shows you the result of this command. The bold text shows, that the interface name is EZAZCX.

Example 2-10 Command output showing the status of the EZAZCX interface

```
D TCPIP,TCPIP,N,DEVL,INTFN=EZAZCX
EZZ2500I NETSTAT CS V2R4 TCPIP 081
INTFNAME: EZAZCX            INTFTYPE: ZCX        INTFSTATUS: READY
    ACTMTU: 65535
    SECCLASS: 255                  MONSYSPLEX: NO
  MULTICAST SPECIFIC:
    MULTICAST CAPABILITY: NO
  INTERFACE STATISTICS:
    BYTESIN                          = 16679449170
    INBOUND PACKETS                  = 46503145
    INBOUND PACKETS IN ERROR         = 0
    INBOUND PACKETS DISCARDED        = 0
    INBOUND PACKETS WITH NO PROTOCOL = 0
    BYTESOUT                         = 34610341556
    OUTBOUND PACKETS                 = 74200486
    OUTBOUND PACKETS IN ERROR        = 0
    OUTBOUND PACKETS DISCARDED       = 0
1 OF 1 RECORDS DISPLAYED
END OF THE REPORT
```

2.2 Planning for containers

You can deploy and run any software that is available as Docker image for IBM Z on the Dockerhub repository in your zCX instances. Go to the Dockerhub website (https://hub.docker.com/) and search for *s390x*. On Dockerhub, you find many images from different areas of interest, which you can pull into your zCX instance. Examples are:

- Python programming language.
- Node.js
- Ubuntu
- Apache Tomcat
- Apache HTTP Server
- IBM WebSphere Application Server Liberty for Developers
- Nginx
- Nextcloud
- MongoDB
- Java
- Grafana
- Jupyter Notebook

The Docker images that you deploy and run can include applications that your developers build and package as Linux on Z.

Tip: If you are wondering whether z/OS Container Extensions is a good fit for your IBM Z environment, go to the https://www.ibm.com/downloads/cas/NJZV7ELN URL, download the questionnaire, and perform a self-assessment.

2.3 Private registry considerations

The IBM Z platform including its operating system z/OS is a highly securable environment. Therefore, it is commonly expected that connections from and to the outside world should be monitored intensively.

When you plan to load Docker images into your zCX appliances, you must connect to a Docker registry. Secure and insecure configurations for connecting to the registry are supported. But in a z/OS environment, for security reasons, you typically use a zCX internal private registry. You can find details on how to set up such a private registry in “Private registry implementation” on page 107.

2.4 Backup and recovery considerations

Backup and recovery are important tasks that you must plan for. As we mentioned in “Characteristics for the zCX instance VSAM linear data sets” on page 27 and “Characteristics for a zCX instance zFS file system” on page 29, there are two types of data sets that are used for zCX setup and operations. If you use IBM components for backup and recovery, plan with your storage management team for the following z/OS components:

- DFSMShsm for automated backup of your VSAM linear data sets and your zFS instance file system. You also can do recovery if needed.
- DFSMShsm USS backup functionality, if you want your zFS file system backup on a file base. If you plan to use this new technique, you must currently either manually perform the

backups or perform them on the basis of workload scheduler. You also can do recovery as well if needed.

- DSMSdss is the underlying component that performs the backup activities under control of DFSMShsm.

We describe the details of backup and recovery tasks based on DFSMShsm and DFSMSdss usage in “Backup and recovery” on page 151.

It is also important for your zCX instances to plan for backups of your application data in your container file system or in a Docker volume. In “Backup and recovery on container level” on page 155, we also describe how you can implement these operations.

3

Security - Overview

As mentioned in Chapter 1, zCX is a new capability that is shipped for the first time with z/OS 2.4. The zCX instance that runs on z/OS runs in a Linux operating system. Inside the zCX instance, Docker containers run. Those containers run products and applications.

In turn, this new paradigm naturally raises the question,
How do we manage security for zCX on z/OS?

This chapter discusses the various aspects of security for zCX on z/OS, including these topics:

- 3.1, “zCX instance security” on page 40
- 3.2, “Security within the zCX instance” on page 43
- 3.3, “Docker in zCX versus Docker on distributed” on page 45

© Copyright IBM Corp. 2019. All rights reserved.

3.1 zCX instance security

In this section, we discuss security issues regarding a zCX instance. The next chapter discusses security within the zCX instance.

A key concept to grasp is that zCX instances run as standard started tasks, which are normal address spaces on a z/OS LPAR.

Associated with a zCX instance are the typical artifacts, which are associated with any started task that runs on z/OS. You must manage these artifacts as you would manage any other process on z/OS.

For a zCX instance, you must consider the following issues:

- RACF user IDs and groups that you must administer and manage
- zFS data sets
- USS directories and files
- zFS files and VSAM linear data sets
- TCPIP
- z/OSMF

3.1.1 Implications of zCX running as a started task

As mentioned, zCX instances run as standard started tasks on z/OS.

This means that all the normal rules that apply to a started task on z/OS apply to zCX that runs as a started task.

z/OS uses mechanisms to ensure that no process running in an address space can access any other part of z/OS outside that address space unless authorized to do so.

Access into and out of the zCX started task is done via TCPIP or virtual disk I/O.

Virtualization engine

When started, the zCX instance allocates real memory and runs the Linux operating system as a virtual machine inside this memory. Because of this, the z/OS architecture prevents any program running inside this virtual machine from being able to access any memory outside of the memory that is allocated.

3.1.2 zCX RACF planning

There are three main issues that relate to the management of zCX instances:

- Using z/OSMF to provision zCX instances
- Running zCX instances as started tasks
- User IDs to administer the running of Docker containers inside a zCX instance

Associated with the first two issues are the use of the following entities:

- zFS data sets
- VSAM data sets
- USS directories and files

3.1.3 RACF groups and user IDs for zCX

When you plan your RACF setup, it is always recommended that you define a user ID under which a started task runs on z/OS. That way, you can prevent this user ID from being used to log on to any z/OS application. This practice is a basic concept for RACF groups and user IDs for zCX instances.

The suggested approach is to create two RACF groups for the following purposes:

- A group to manage user IDs that would be used in z/OSMF to run workflows to manage the provisioning of zCX instances.
- A group to manage user IDs that would be used to run zCX instance started tasks.

Additionally user IDs are used inside zCX instances to administer Docker containers. There are two options that can be used for these user IDs:

1. During the zCX provisioning process, specify an initial administration user ID that will be defined in the zCX instance. You can then log on to the zCX instance with this user ID and define additional user IDs, if they are required. **OR**
2. Define the zCX instance to use an LDAP server as an external user repository, and log on to the zCX instance with an LDAP user ID.

If you use the first approach, you also must define a group to manage user IDs that are administration user IDs that can be used to log in to zCX instances.

We show example of creating such groups in 4.2.4, “Create RACF groups” on page 50.

Typically, you have multiple logical environments that run zCX instances, such as development, test, pre-production, and production.

You would set up the previously mentioned set of groups for each logical environment in which you plan to run zCX instances.

Within each group, you would then create at least one RACF user ID.

How many additional user IDs that you create depends on the size of your organization. For example, if you plan to have a number of people logging on to z/OSMF to run zCX workflows, you must define additional user IDs for these people.

You might decide to run all zCX instances in a logical environment under the same RACF user ID. Alternatively, you might define multiple user IDs so that different zCX instances run under these different user IDs.

In 4.2.5, “Create RACF user IDs” on page 50, we show example RACF commands that define the user IDs to manage the zCX instances that we set up in our environment.

3.1.4 zFS files and VSAM linear data sets

You provision zCX instances by running workflows in z/OSMF. During the running of the workflow, a single zFS file and five VSAM linear data sets are created.

> **Note:** You must ensure that the user ID that you use in z/OSMF has the appropriate security to create this zFS file and VSAM linear data sets. Also, you must associate RACF data set rules with these data sets to protect them.

You should keep the UACC set to NONE while you create RACF rules for these data sets. Then, permit READ access to the group that is used to run the zCX workflows in z/OSMF and the group that contains the user IDs under which you run the zCX started tasks.

See the following topic in the z/OS 2.4 Knowledge Center to learn about issues regarding user IDs:
https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.4.0/com.ibm.zos.v2r4.izso100/izso100_setupvsamzfsreqs.htm
This topic covers the following issues:

- User IDs in z/OSMF that run the zCX workflows require ALTER access to the VSAM data sets so that they can be allocated.
- User IDs for running the zCX started task must have CONTROL access to the VSAM data sets.

SMS Considerations

One of the VSAM linear data sets that is created is approximately 20 GB in size. This means that you also must plan your SMS setup to have a DATACLASS that has Extended Format and Extended Addressability.

Pervasive Encryption

See the following topic in the z/OS 2.4 Knowledge Center to learn z/OS pervasive encryption:
https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.4.0/com.ibm.zos.v2r4.izso100/izso100_setupsecurity.htm
This topic covers the following issues:

- By default, zCX instance root file systems and swap data volumes that are backed by VSAM linear data sets are encrypted by Linux, based on LUKS encryption. Pervasive encryption is not recommended for these linear VSAM data sets.
- Pervasive encryption is recommended for configuration, user data, diagnostics data, VSAM linear data sets, and your zCX instance directory zFS file system. You can define an encryption key label by using one of the following methods:
  - The RACF data set profile DFP segment method OR
  - The DATA CLASS SMS construct method

3.1.5 USS directory and files

When you run the zCX workflows in z/OSMF, a new home directory is created for the zCX instance. This directory is created either in a default directory or in a directory that you specify. This choice of directory has the following implications:

- Before you run the zCX workflow, you must plan what USS directory to use. This directory serves as the parent directory in which the zCX instance directory is created.
- You must consider what RACF user ID and group to define as the owner and group for these directories.

Keep these factors in mind regarding user IDs:

- The user ID the runs workflows in z/OSMF must be able to create a new directory.
- The user ID that runs the zCX started task must be able to read files from this directory and write files to a sub-directory called FFDC.

In Section 4.3.1, “zCX directories” on page 52, we show how we set up our USS directory to support such a configuration.

Sysplex considerations

If you would like to be able to start the zCX instances that you define on any z/OS LPAR in the sysplex, you must ensure that the directory that you use to store the zCX instance configuration is shared across all LPARs in the sysplex.

In 4.3.1, “zCX directories” on page 52, we show how we used the default z/OS global directory that is located in the root directory as the location to achieve this goal.

3.1.6 TCPIP Networking

TCPIP is used to give access into and out of a zCX instance. The following link discusses the use of z/OS TCPIP and communication to zCX instances, including mentioning how z/OS IP filters can be used to restrict access to and from zCX:
https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.4.0/com.ibm.zos.v2r4.izso100/izso100_setupnetworking.htm

3.1.7 z/OSMF

You can provision zCX instances by using only the zCX workflows that are supplied in z/OSMF.

The workflow performs many tasks. One of these tasks is the mounting of a newly defined zFS to a new USS directory. This task requires proper configuration of the z/OSMF environment. See Section 4.4, “z/OSMF” on page 53 describes how we did this.

We recommend that you review z/OSMF set up by reading the various articles in the z/OS 2.4 Knowledge Center that describe the setup of z/OS, including this one:
https://www.ibm.com/support/knowledgecenter/SSLTBW_2.3.0/com.ibm.zos.v2r3.izua300/izuconfig_SecurityStructuresForZosmf.htm#DefaultSecuritySetupForZosmf__SecuritySetupRequirementsForCoreFun

3.1.8 Other data sets

As mentioned, the zCX instance uses 1 zFS file system and 5 VSAM linear data sets. There is no mechanism that allows any container — including the zCX Docker CLI SSH container in the zCX instance — to access any other data set, USS directory, or file that resides in z/OS.

3.2 Security within the zCX instance

In this section, we discuss security considerations within the zCX instance.

3.2.1 Linux in the zCX instance

When started, each zCX instance runs a provisioned Linux operating system. There is no direct access to this underlying Linux operating system.

3.2.2 The SSH container

To do the administration of the Docker containers that exist inside a zCX instance, you log in to the zCX instance with an administration user ID. When you do such a login, you are logging

in to an SSH container in the zCX instance. Figure 3-1 shows the SSH container in a zCX instance.

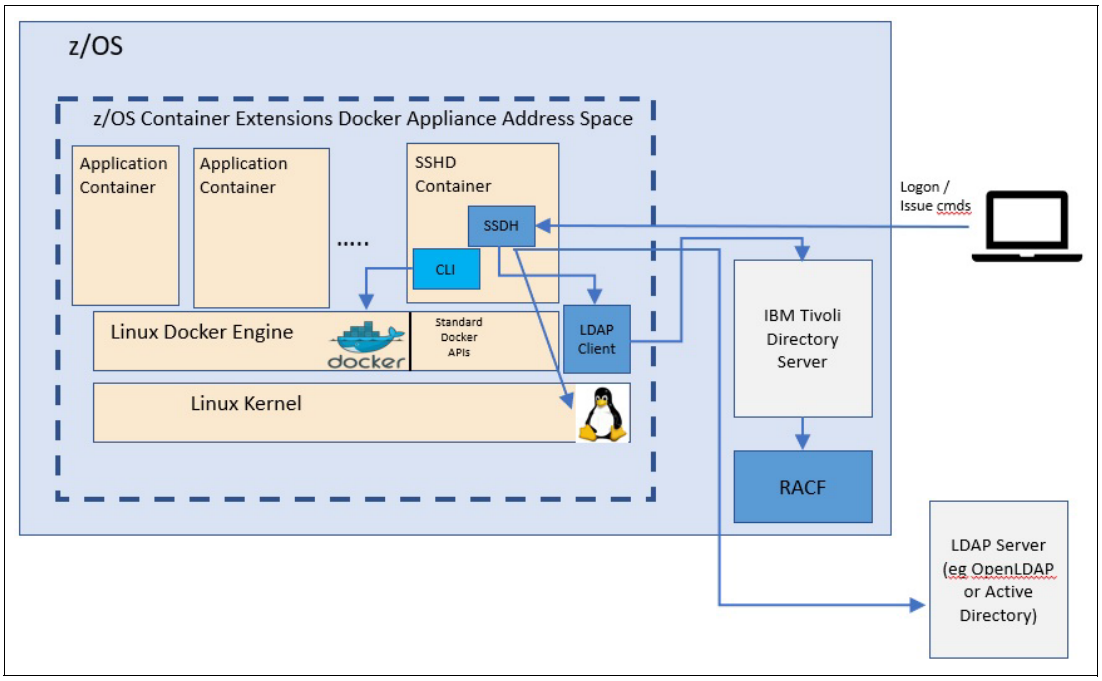


Figure 3-1 SSH Container in a zCX instance

All such connections to this SSH container can be done over SSH only, which ensures that any data that flows over this connection is encrypted.

From this SSH container, you issue Docker commands to administer the Docker containers that you want to run in the zCX instance. This user ID for administering the zCX instance is not a UNIX root user ID.

3.2.3 zCX administration user IDs

To administer Docker containers in a zCX instance, you use administration user IDs. You can set up these user IDs in the zCX instance through either of these methods:

- Defined as local user IDs in the zCX instance
- Configuring the zCX instance to use an external LDAP server

This link from the z/OS 2.4 Knowledge Center describes these options in further detail: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.4.0/com.ibm.zos.v2r4.izso100/izso100_usermanagementchapterintro.htm

Section 4.2.6, “Create SSH Keys” on page 51, shows how we set up access to a local administrator user ID in the zCX instance.

Section 9.2, “Configuring zCX to use an LDAP server” on page 222 shows how we configure a zCX instance to use an LDAP server for the administration user IDs.

Adding additional local administration user IDs

If you use local administration user IDs in a zCX instance, you can define additional local administration user IDs, as shown in Section 9.1.1, “Adding Docker users” on page 220.

Note: When you use the local administration user ID approach, there are in effect two types of users:

- Users who can do Docker commands and user administration.
- User IDs that are created by the administration user IDs, who can do Docker commands but not user administration.

3.3 Docker in zCX versus Docker on distributed

This section discusses some differences between running Docker in zCX and on distributed systems.

3.3.1 Docker capabilities of the SSH container

There are some restrictions on which Docker commands can be issued in a zCX instance. These are documented in the following topic of the z/OS 2.4 Knowledge Center under the heading “Restrictions in the zCX Docker CLI SSH container”:
https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.4.0/com.ibm.zos.v2r4.izso100/izso100_whatissshd.htm

Two restrictions from the list are worth repeating here.

- Privileged Docker containers are considered insecure and not supported in zCX.
- Changing the namespace to host is not supported in zCX.

3.3.2 Considerations regarding the mounting of file systems

On a distributed system, you could mount any directory of the host system in Read Only or Read Write mode to a container. If the directory is mounted as Read Write, this configuration would allow a program in the container to directly update the file system of the hosting system.

In zCX instance, when you log on, you log in to the SSH container. You might try to mount a directory when you issue the **`docker run`** command with the **`-v`** or **`--mount`** option. However, you will never get an error message to indicate that there was an issue with the mount. The actual result depends on the directory that you specified.

Note: You cannot mount any directory that exists in the SSH container to a new Docker container that you are starting.

If the directory that you specify exists in the underlying Linux operating system, that directory is mounted in Read Only mode in the new container.

4

Provisioning and managing your first z/OS Container

This chapter shows how to provision and manage your first z/OS Container Extension server. At the end of this process, you have a zCX instance that is ready to run Docker containers as described in Chapter 5, “Your first running Docker container in zCX” on page 95.

- These sections describe how to plan and get your first zCX instance up and running:
  - 4.1, “Overview of zCX provisioning process” on page 48
  - 4.2, “RACF planning” on page 48
  - 4.3, “USS planning” on page 52
  - 4.4, “z/OSMF” on page 53
  - 4.5, “Configure TCPIP addresses” on page 55
  - 4.6, “SMS planning” on page 57
  - 4.7, “zCX property file” on page 58
  - 4.8, “Run zCX provisioning workflow” on page 60
  - 4.9, “Starting the zCX instance” on page 71
  - 4.10, “Log on to the zCX instance” on page 77
- 4.11, “Reconfiguring a zCX instance” on page 77 describes how to make changes to the zCX instance configuration.
- 4.12, “Adding a volume to a zCX instance” on page 82 shows you how to add additional swap and user volumes to a zCX instance.
- 4.13, “Deprovisioning a zCX instance” on page 89 shows how to deprovision a zCX instance.
- The current chapter also covers these topics:
  - 4.14, “Other provisioning workflows” on page 92
  - 4.15, “How to rerun a workflow” on page 92

© Copyright IBM Corp. 2019. All rights reserved.

4.1 Overview of zCX provisioning process

To provision the zCX instance, you run a workflow in z/OSMF. However, before you run the workflow you first must plan and prepare a number of items in z/OS, as described in this section.

Table 4-1 lists the section that describe how to prepare for and then run the workflow to provision a zCX Instance.

Table 4-1 Chapters that explain the zCX provisioning process

Section	Topic
4.2, “RACF planning”	How to set up the RACF environment
4.3, “USS planning” on page 52	What directories in USS are required
4.4, “z/OSMF” on page 53	How z/OSMF is required
4.5, “Configure TCPIP addresses” on page 55	TCPIP requirements and how to configure
4.6, “SMS planning” on page 57	SMS requirements for VSAM linear data sets that are used
4.7, “zCX property file” on page 58	Preparation of the property file to use in the workflow
4.8, “Run zCX provisioning workflow” on page 60	How to execute the zCX provisioning workflow
4.9, “Starting the zCX instance” on page 71	How to start the zCX instance
4.10, “Log on to the zCX instance” on page 77	How to log on to the zCX instance

4.2 RACF planning

Prior to creating zCX instances, you must consider what RACF user IDs and groups are required to manage the provisioning and operation of those instances.

The three user IDs and groups that we use in our environment have these distinct roles:

- Provision the zCX instance in z/OSMF.
- Run the zCX instance started task.
- Administer the zCX instance.

4.2.1 Overview

You provision a zCX instance in three main steps:

1. Use z/OSMF to run the zCX provisioning workflow.
2. Run the zCX instance as a started task on z/OS.
3. Access the zCX instance by using the admin user ID.

Before you start the preceding process, you must plan what user IDs and groups to use for the different parts. You could use a single user ID for all steps of provisioning a zCX instance, but that is not recommended.

When you plan your RACF setup, it is always recommended that you define a user ID under which a started task runs on z/OS. That way, you can prevent this user ID from being used to log on to any z/OS application.

The suggested approach is as follows:

1. Define user IDs that can be used to execute the zCX workflows in z/OSMF.
2. Define user IDs that the zCX instance started tasks run under.
3. Define user IDs that are used to connect to admin user ID in the zCX instance.
4. Each of the preceding groups of user IDs would be in their own RACF groups.

4.2.2 The zCX Admin user ID

One concept to grasp in relation to zCX instances, is that of the zCX admin user ID.

When you provision a zCX instance, one of the properties that you specify is the admin user ID in the zCX instance. This admin user ID is defined in the zCX instance during the provisioning process and resides in the user repository of the zCX instance.

There is no connection between this admin user ID that is defined in the zCX instance and RACF.

4.2.3 Planning the first logon to a zCX instance

When the zCX instance is started for the first time, you can log on only through the admin user ID that you specified in the properties file.

Additionally, the only way you can log on is via an SSH connection that uses certificates. You cannot log on to the zCX admin user ID using a password.

There is an exception to this rule. If you provisioned the zCX instance to use an LDAP server for user authentication, then the first logon could be done with an LDAP user ID and password. However, when you set up your first zCX instance it is recommended that you use the SSH approach, and consider use of LDAP at a later stage.

Before you run the zCX provisioning workflow in z/OSMF, you must generate an SSH key pair, to supply the public SSH key as a property to the workflow. In this context, you must answer two questions:

- For first logon, from where do I want to connect to the zCX instance?
- Where do I want to create the SSH certificates?

Choose one of the following methods to answer the preceding questions:

- **z/OS USS:**
  a. Log on to the z/OS USS (z/OS UNIX System Services) environment with a user ID, and run the SSH keygen utility there.
  b. Provision the zCX instance.
  c. After it is started, issue the **ssh** command to log on from that USS session.
- **Distributed server:**
  a. Run the SSH utility on some distributed server.
  b. Provision the zCX instance.
  c. After it is started, issue the **ssh** command from that distributed server to log on.

Assuming that you have access, you can copy the SSH private key from z/OS to a distributed server or vice versa. That way, you can SSH into the zCX instance from both z/OS and a distributed server.

The approach that we describe in this Redbooks document is to do the SSH key generation in z/OS USS.

4.2.4 Create RACF groups

We created three RACF groups by using the commands that are shown in Example 4-1:

Example 4-1 Create three RACF groups

```
ADDGROUP zcxprvg OMVS(AUTOGID)  DATA('zCX Prov Group')
ADDGROUP zcxstcg OMVS(AUTOGID)  DATA('zCX STC Group')
ADDGROUP zcxadmg OMVS(AUTOGID)  DATA('zCX Admin Group')
```

4.2.5 Create RACF user IDs

We then created the user IDs shown in Table 4-2.

Table 4-2 User IDs to manage zCX

User ID	Group	Description
ZCXPRV1	ZCXPRVG	User ID to use in z/OSMF to provision zCX instance
ZCXSTC1	ZCXSTCG	User ID to run the zCX instance started task under
ZCXADM1	ZCXADMG	User ID to use to do first logon to zCX instance

We issued the RACF commands that are shown in Example 4-2 to define the three user IDs.

Example 4-2 RACF commands to define the user IDs

```
ADDUSER ZCXPRV1 PASSWORD(NEW2DAY) +
  NAME('zCX Provision User 1') +
  OWNER(HAIMO) UACC(ALTER) DFLTGRP(ZCXPRVG) +
  AUTHORITY(JOIN) GRPACC  +
  TSO(ACCTNUM(ACCNT#) PROC(IKJACCNT) SIZE(2096128) MAXSIZE(0) +
  UNIT(SYSALLDA) COMMAND(ISPPDF)) +
  OMVS(AUTOUID HOME(/u/zcxprv1) PROGRAM(/bin/sh))

ADDUSER ZCXSTC1 +
  NAME('zCX STC User 1') +
  OWNER(HAIMO) UACC(ALTER) DFLTGRP(ZCXSTCG) +
  AUTHORITY(JOIN) GRPACC  +
NOPASSWORD NOOIDCARD +
  TSO(ACCTNUM(ACCNT#) PROC(IKJACCNT) SIZE(2096128) MAXSIZE(0) +
  UNIT(SYSALLDA) COMMAND(ISPPDF)) +
  OMVS(AUTOUID HOME(/u/zcxstc1) PROGRAM(/bin/sh))

ADDUSER ZCXADM1 PASSWORD(NEW2DAY) +
  NAME('zCX Admin User 1') +
  OWNER(HAIMO) UACC(ALTER) DFLTGRP(ZCXADMG) +
  AUTHORITY(JOIN) GRPACC  +
  TSO(ACCTNUM(ACCNT#) PROC(IKJACCNT) SIZE(2096128) MAXSIZE(0) +
  UNIT(SYSALLDA) COMMAND(ISPPDF)) +
  OMVS(AUTOUID HOME(/u/zcxprv1) PROGRAM(/bin/sh))
```

Note: The RACF command that is used to define the ZCXSTC1 user ID has the keywords NOPASSWORD NOOIDCARD, which mean that the user ID cannot log on to z/OS.

4.2.6 Create SSH Keys

As mentioned earlier in this section, the first logon to the zCX instance must be done via SSH using certificates.

To do this, log on to USS using the ZCXADM1 user ID. Then, enter the following command to start the SSH key generation process:

```
ssh-keygen
```

The output from this command is shown in Example 4-3.

Example 4-3 Output from first log on to the zCX instance

```
ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/u/zcxadm1/.ssh/id_rsa):
Created directory '/u/zcxadm1/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /u/zcxadm1/.ssh/id_rsa.
Your public key has been saved in /u/zcxadm1/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:JoAF20dFWAfRrtyaF3HRoTRr4Czvrw8wFVRXUxwjS2A ZCXADM1@WTSC74
The key's randomart image is:
+---[RSA 2048]----+
|  ... .=*==E=+oB=|
|   = ..  +o+o=+ +|
|  o o . ..+ +o   |
|     o   +o..    |
|      ..So.o     |
|       oo+o      |
|         oo.     |
|        o .o     |
|         ..oo    |
+----[SHA256]-----+
```

Note: When we received the *Enter passphrase* message, we did not supply any value and just pressed the Enter key.

The preceding process creates a hidden directory called .ssh.

Note: Do not change the directory permissions that are set on the created .ssh directory. Otherwise, any attempt to use SSH to log on with certificates fails.

Example 4-4 shows the directory and file permissions of the created .ssh directory:

Example 4-4 .ssh directory settings

```
ls -lrta | grep .ssh
drwx------   2 ZCXADM1  ZCXADMG     8192 Aug 15 21:51 .ssh
ZCXADM1 @ SC74:/u/zcxadm1>ls -lrtR .ssh
```

```
.ssh:
total 48
-rw-r-----   1 ZCXADM1  ZCXADMG      396 Aug 15 21:30 id_rsa.pub
-rw-------   1 ZCXADM1  ZCXADMG     1679 Aug 15 21:30 id_rsa
-rw-r--r--   1 ZCXADM1  ZCXADMG      543 Sep  9 09:10 known_hosts
```

The id_ras.pub file contains the public key certificate and you need this later in Section 4.7.2, “Setting the SSH key in property file” on page 59.

4.3 USS planning

Prior to running the zCX provisioning workflow in zOMSF, you must consider these factors:

- What USS directories are involved?
- What user IDs will access them?
- What access is required by these user IDs?

4.3.1 zCX directories

When you run the zCX workflow in z/OSMF, a directory of the name of the zCX instance that you are creating is created in a USS directory that you specify. A number of files are also written to this directory when the workflow runs.

This implies that the user ID that runs the z/OSMF workflow needs to have update access to the USS directory that you specify for creation of the instance directory.

If you have a sysplex, you also must consider whether you want the flexibility to start the zCX instances on any LPAR in the sysplex. If you do want this flexibility, specify a USS directory that is visible to all LPARs in the sysplex.

From z/OS 2.3 on, there is a directory called **global** that is defined under the root directory. Thus, by default your z/OS sysplex has a directory that is common to all LPARs in that sysplex. We used **global** as the parent directory to store files that are related to the configuration of our zCX instances. That way, we could start the zCX instances on any LPAR in our sysplex as needed.

In our environment, we created a directory called **zcx** under the **global** directory. Then, we defined a zFS and mounted it at the **/global/zcx** directory prior to the creation of the lower-level directories.

Table 4-3 shows the directories that we then created in our environment and their purpose.

Table 4-3 zCX configuration directories

Directory	Purpose
`/global/zcx/cfg`	Parent directory for various files related to configuring zCX instances
`/global/zcx/cfg/properties`	Directory to store customized property files for use in zCX workflows
`/global/zcx/instances`	Parent directory for each created zCX instance directory

We wanted to use the ZCXPRV1 user ID to run the zCX workflow in z/OSMF. So, we set that user ID and its group as the owner of the directories that we want to use, as shown in Table 4-4.

Table 4-4 Directory settings

Directory	Owner	Group	Permissions
`/global/zcx`	`zcxprv1`	`zcxprvg`	755
`/global/zcx/cfg`	`zcxprv1`	`zcxprvg`	770
`/global/zcx/cfg/properties`	`zcxprv1`	`zcxprvg`	770
`/global/zcx/instances`	`zcxprv1`	`zcxprvg`	770

4.4 z/OSMF

The only way to provision zCX instances is to use z/OSMF to execute the zCX workflow that z/OSMF supplies. For this reason, you must have a working z/OSMF environment on the z/OS LPAR in the z/OS LPAR or sysplex where you plan to run the zCX instances.

4.4.1 z/OSMF configuration

It is beyond the scope of this document to explain how to set up and configure z/OSMF. Information about configuring z/OSMF can be found in the following topic in the z/OS 2.4 Knowledge Center:
https://www.ibm.com/support/knowledgecenter/SSLTBW_2.4.0/com.ibm.zos.v2r4.izua300/abstract.htm

There is a related IBM Redbooks publication, *IBM z/OS Management Facility V2R3*, SG24-7851: http://www.redbooks.ibm.com/abstracts/sg247851.html

In z/OSMF, the Security Configuration Assistant helps you confirm that the user ID that you plan to use to run the zCX workflow has the required authority. You would ask your z/OSMF administrator to log on and find the Security Configuration Assistant as shown in Figure 4-1 on page 54.

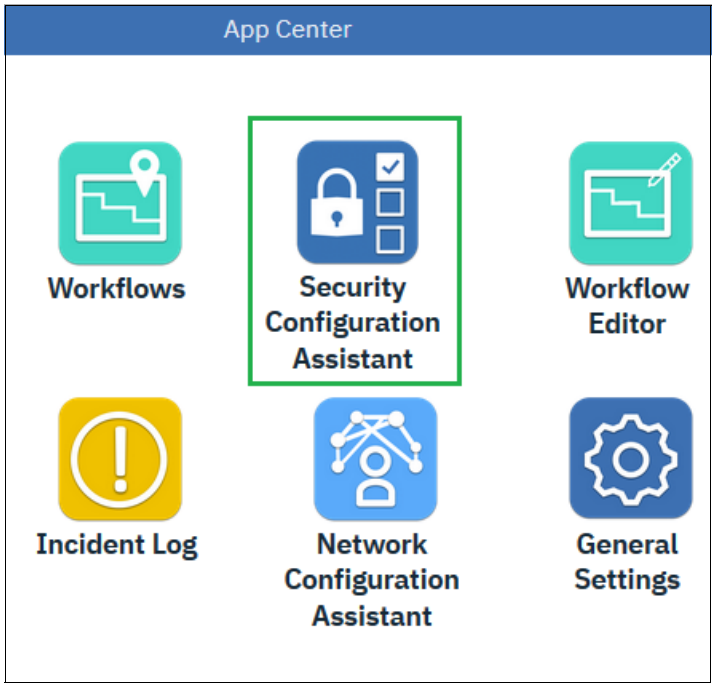


Figure 4-1 Security Configuration Assistant in z/OSMF

Double-click the Security Configuration Assistant icon. You see a display that is similar to the one that is shown in Figure 4-2.

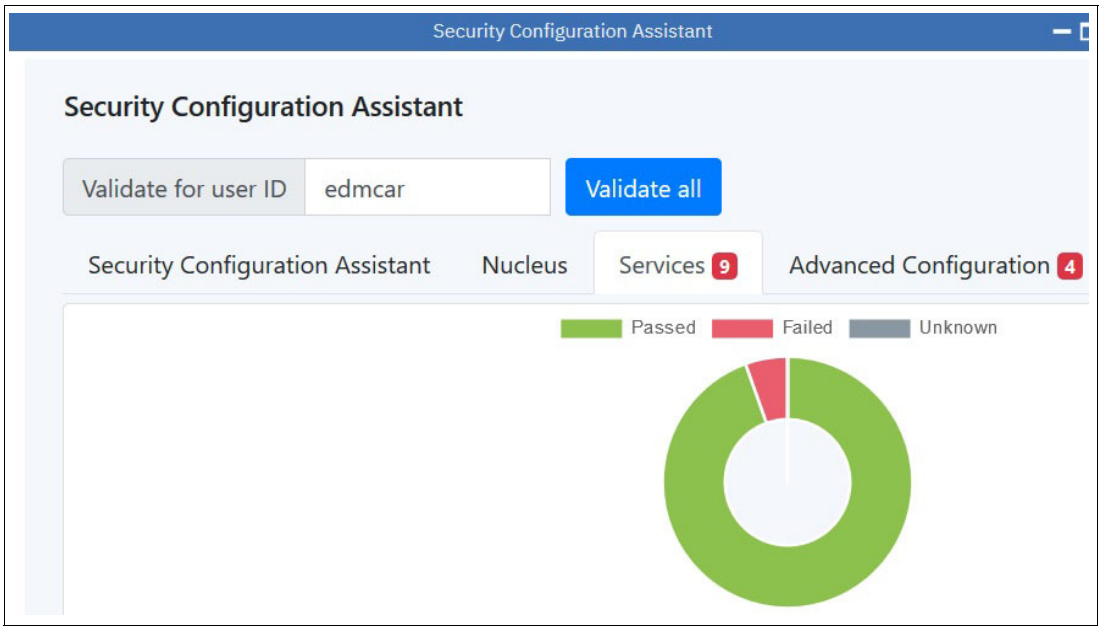


Figure 4-2 Initial display of Security Configuration Assistant

Enter the user ID that you plan to use in z/OSMF to run the zCX workflow. Then, click **Validate all**. After the validation is complete, scroll down in the display to find z/OSMF Workflows as shown in Figure 4-3.

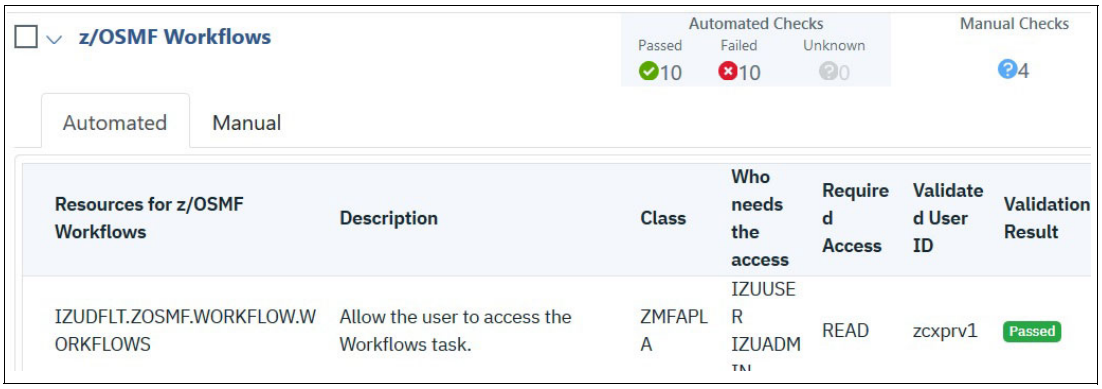


Figure 4-3 Result of z/OSMF access checks

Even though on our system the ZCXPRV1 user ID that we planned to use to run the zCX workflow did not pass all checks, it still had enough access to run the zCX workflow.

4.5 Configure TCPIP addresses

Each zCX instance requires its own TCPIP address in z/OS.These TCPIP addresses must be defined to TCPIP on z/OS with a keyword of **ZCX**.

We dynamically added TCPIP addresses for our zCX instances by using the following process.

1. We created a member in the TCPPARMS data set that our TCPIP loads configuration information from called SYS1.TCPPARMS(ZCXRBVIP).
2. In that member, we added the lines that are shown in Example 4-5 shows the lines that we added in that member. The **ZCX** keyword and the TCPIP addresses that we added on each line are bolded for your attention:

Example 4-5 Member to add TCPIP addresses for zCX instances

```
VIPADYNAMIC
  VIPARANGE DEFINE 255.255.255.255 129.40.23.68 ZCX ; zCX Instance 1
  VIPARANGE DEFINE 255.255.255.255 129.40.23.69 ZCX ; zCX Instance 2
ENDVIPADYNAMIC
```

If you do not code the ZCX parameter, the zCX instance fails to start and generates messages similar to the ones that you see in Example 4-6.

Example 4-6 zCX instance startup failure

```
GLZN004I Failed network connection at  317
IP@=129.40.23.78
for zCX instance ZCXME02. Attempt 1
of 4 will be retried.
IEF450I ZCXME02 ZCXME02 - ABEND=SA22 U0000 REASON=00000000
```

3. From SDSF, we entered the following command to dynamically add the TCPIP addresses:

   RO *ALL,V TCPIP,,OBEY,SYS1.TCPPARMS(ZCXRBVIP)

4. In the z/OS syslog, we review the messages in Example 4-7 that confirm that the TCPIP addresses were dynamically added.

Example 4-7 Successful dynamic update of TCPIP

```
SYSNAME  RESPONSES ---------------------------------------------------
SC74      EZZ0060I PROCESSING COMMAND: VARY TCPIP,,OBEY,SY
         S1.TCPPARMS(ZCXRBVIP)
          EZZ0300I OPENED OBEYFILE FILE 'SYS1.TCPPARMS(ZCXRBVIP)'
          EZZ0309I PROFILE PROCESSING BEGINNING FOR 'SYS1.
         TCPPARMS(ZCXRBVIP)'
          EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'S
         YS1.TCPPARMS(ZCXRBVIP)'
          EZZ0053I COMMAND VARY OBEY COMPLETED SUCCESSFULLY
SC75      EZZ0060I PROCESSING COMMAND: VARY TCPIP,,OBEY,SY
         S1.TCPPARMS(ZCXRBVIP)
          EZZ0300I OPENED OBEYFILE FILE 'SYS1.TCPPARMS(ZCXRBVIP)'
          EZZ0309I PROFILE PROCESSING BEGINNING FOR 'SYS1.
         TCPPARMS(ZCXRBVIP)'
          EZZ0316I PROFILE PROCESSING COMPLETE FOR FILE 'S
         YS1.TCPPARMS(ZCXRBVIP)'
          EZZ0053I COMMAND VARY OBEY COMPLETED SUCCESSFULLY
```

5. We entered the following command to check that the TCPIP address does indeed have the ZCX attribute:

   `D TCPIP,,NETSTAT,VIPADCFG,IPADDR=129.40.23.68`

   This command produced the output shown in Example 4-8.

Example 4-8 Display zCX type TCPIP address

```
EZZ2500I NETSTAT CS V2R4 TCPIP 000
DYNAMIC VIPA INFORMATION:
  VIPA RANGE:
    ADDRESSMASK       IP ADDRESS        MOVEABLE  SAF NAME  FLG
    -----------       ----------        --------  --------  ---
    255.255.255.255   129.40.23.13      NONDISR             C
1 OF 1 RECORDS DISPLAYED
END OF THE REPORT
```

The value of C under the FLG column indicates that the TCPIP address has the ZCX attribute.

This output is explained in this link from the z/OS 2.4 Knowledge Center: https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.4.0/com.ibm.zos.v2r4.halu101/vicfgrpt.htm#vicfgrpt

The preceding link includes the description that is shown in Figure 4-4, which explains the value of C and indicates that the TCPIP address has the zCX attribute.

VIPA Range

Displays the configured dynamic VIPA range information.

For a SHORT format report:

Flg

The following value can be displayed in the Flg field:

C

Indicates this VIPARANGE was defined with the ZCX keyword and is intended for use by the z/OS Container Extensions function.

Figure 4-4 Explanation for value of C in FLG column

You can also use the Network Configuration Assistant in z/OSMF to configure TCPIP addresses with the ZCX attribute. For more information about this option see this web page: https://www.ibm.com/support/knowledgecenter/SSLTBW_2.4.0/com.ibm.tcp.ipsec.ipsec.help.doc/com/ibm/tcp/ipsec/cloud/IparViewDetailZCX.html

4.6 SMS planning

The default zCX workflow creates one zFS file and five VSAM linear data sets. One of the VSAM linear data sets is approximately 20 GB in size.

This link from the z/OS 2.4 Knowledge Center describes the requirements for these data sets: https://www.ibm.com/support/knowledgecenter/SSLTBW_2.4.0/com.ibm.zos.v2r4.izso100/izso100_setupvsamzfsreqs.htm

> **Note:** DATACLASS — with Extended Format and Extended Addressability — must be enabled to allow allocation of data sets greater than 4 GB.

In our environment, we defined an SMS Data Class called VSAMLSX with the attributes that are required to support large VSAM linear data sets as shown in Example 4-9.

Example 4-9 SMS Data Class with extended attributes

```
DATACLAS                              EXTENDED
NAME     RECORG  DATA SET NAME TYPE   ADDRESSABILITY
--(2)--- -(3)--  -------(26)-------   -----(27)-----
VSAMLSX  LS      EXTENDED REQUIRED    YES
```

We decided that the high-level qualifier for our data sets would be ZCX.REDB. We then defined SMS ACS rules so that if a VSAM data set was being allocated for that high-level qualifier it would be allocated to the VSAMLSX SMS data class. Otherwise, non-VSAM data sets would go to a standard SMS data class. The ACS rules that we defined are shown in Example 4-10.

Example 4-10 SMS ACS rules to control file allocation

```
FILTLIST ZCX         INCLUDE(ZCX.**)

  SELECT
    ...
    WHEN (&DSN EQ &ZCX)
      DO
        IF &DSORG = 'VS' THEN
          SET &DATACLAS EQ 'VSAMLSX'
        ELSE
          SET &DATACLAS EQ ''
      END
    ...
  END
```

We then allocated a number of volumes to the VSAMLSX data class.

This approach had the following advantage:
There is no need to supply SMS data class values in the zCX provisioning workflow.

4.7 zCX property file

When you execute the zCX provisioning workflow in z/OSMF, you must supply several parameters. You can manually enter values for these parameters in the workflow panels.

However, it is recommended that you make a copy of the supplied sample zCX properties file, and set your custom values in that file. That way, you can reuse that property file as input to other supplied provisioning workflows for your instance, such as deprovision or reconfiguration. When you start any of these zCX provisioning workflows, you can supply your customized property file as an input file, and avoid manually typing in the values again.

The sample zCX properties file is typically supplied on your z/OS LPAR in this path:

`/usr/lpp/zcx_zos/properties/workflow_variables.properties`

4.7.1 Customizing the properties file

We copied the supplied zCX property file to the location that is shown in Example 4-11 and gave it a name that reflects the name of the zCX instance that we planned to create:

Example 4-11 Property file

```
/global/zcx/cfg/properties/zcxed01_workflow_vars.properties
```

Notice that the file is stored in EBCDIC in the z/OS USS environment.

We then edited that file, making the changes shown in Table 4-5.

Table 4-5 Modified properties

Property	Value	Description
ZCX_REGISTRY_DIR	**/global/zcx/instances**	Uncommented this line to use default location
ZCX_INSTNAME	**ZCXED01**	Name of zCX instance to be created
ZCX_SAVE_PROPERTIES	**/global/zcx/cfg/properties**	Uncomment this line and set to directory where workflow can save copy of properties file
ZCX_HOSTNAME	**sc74cn01.pbm.ihost.com**	DNS name of TCPIP address that zCX instance uses
ZCX_GUESTIPV4	**129.40.23.68**	TCPIP address that zCX instance uses
ZCX_HOSTDNS1	**129.40.106.1**	TCPIP address of DBNS server
ZCX_HLQ	**ZCX.REDB**	High-level qualifier to use for data sets that are created by workflow
_IZU_VARIABLE_SUBSTITUTION_ON		Uncomment this line
ZCX_ZFS_FILESYSTEM_HLQ	**${ZCX_HLQ}**	Uncomment this line
ZCX_ZFS_GROUP_NAME	**ZCXPRVG**	RACF group to set as owner of the instance sub-directory that is created
ZCX_DOCKER_ADMIN	**admin**	Initial default administration user ID to be defined in the zCX instance

4.7.2 Setting the SSH key in property file

One property that is not mentioned in the preceding table is **ZCX_DOCKER_ADMIN_SSH_KEY**. We updated this property to be the public key that was generated when we ran SSH keygen in Section 4.2.6, “Create SSH Keys” on page 51.

To do this, you run the following command that is found in **/u/zcxadm1/.ssh/id_rsa.pub**:

```
cat /u/zcxadm1/.ssh/id_rsa.pub
```

Example 4-12 shows typical output for this command.

Example 4-12 Listing the ssh public key


```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDHu/Hky1Vmc2s8Phk3Mmq+4x7Ojti9rZtOUoeD1DqPIWfnJAf2wn1BQz9TMHp
+MKOzGReaP3DF8lgkEHdR2LXvleoxEh2RbFKURyn91JYIETuqtlvwPMKVOh4SXy987xUAwxSdFd5aOJ4xbJbQI8jRXQ
EFbKWLVFNozWWvOPnVFlzcI3nbA+Hb9guVbraXuvdXQIjjQuq+jJhqUKncgVimygHR45dnEbmiOCSyYG3B1rAzEa1c5
2GZxJlfD3MhOn9EhQeFsFOJNzsr1O+uV3eHm3urewbQdZh3VG2rwqfmjwoiR61RhBhGnYc1Zg1ZDZxHq3pF7xAYAqYx
DXceZz5B ZCXADM1@WTSC74
```


You can then set the SSH public key as the value for the **ZCX_DOCKER_ADMIN_SSH_KEY** property in your custom property file by using an editor. This property can be uncommented and modified in the custom property file as needed.

Given the length of the SSH public key, it can be problematic to do the edit process to set this value in the property file. So, instead of the preceding manual cut-and-paste operation, we used the following approach:

1. Log on with the **zcxadm1** user ID, and run the following command to copy the SSH public key to a file in the **tmp** directory, and make the file readable to all:

   ```
   cp /u/zcxadm1/.ssh/id_rsa.pub /tmp/id_rsa.pub
   chmod 664 /tmp/id_rsa.pub
   ```

2. With the **ZCXPRV1** user ID, issue the following command as a single command in the USS environment:

   ```
   export propFile=/global/zcx/cfg/properties/zcxed01_workflow_vars.properties;
   export sshKey="$( cat /tmp/id_rsa.pub )" ; sed "s|#ZCX_DOCKER_ADMIN_SSH_KEY
   :|ZCX_DOCKER_ADMIN_SSH_KEYS : $sshKey|" $propFile > p.tmp ; cp p.tmp $propFile
   ; cat $propFile | grep SSH_KEY ;
   ```

   The command works as follows:

   – Runs an export command to set a propFile environment variable to the name of the file that we want to update.

   – Runs a second export command to set an **sshKey** environment variable to the contents of the file that contains the **ssh** public key.

   – Runs a **sed** command to set the SSH public key as the value for the **ZCX_DOCKER_ADMIN_SSH_KEY** property and uncomment it, and then writes the updated file to a temp file.

   – Copies the temp file back over the original input file.

   – Uses **cat** and **grep** to display the updated property to verify that it has been set correctly.

4.8 Run zCX provisioning workflow

We can now use z/OSMF to run the zCX provisioning workflow to create the zCX instance.

Note: Some additional information before continuing:

- Leading slashes are required for path/file input workflow variables
- Trailing slashes are not allowed for path/file input workflow variables
- Maximum VSAM LDS allocation size is 46000 MB and enforced by workflows
- Minimum zCX Appliance memory is 2 GB
- Time zone can be set manually in the zCX Docker CLI

4.8.1 Log on to z/OSMF

Log on to z/OSMF by using the provisioning user ID that you defined to RACF.

In our environment, the URL to access zOMSF was:

```
https://wtsc74.pbm.ihost.com:2443/zosmf
```

Figure 4-5 on page 60 shows the initial z/OSMF logon screen.

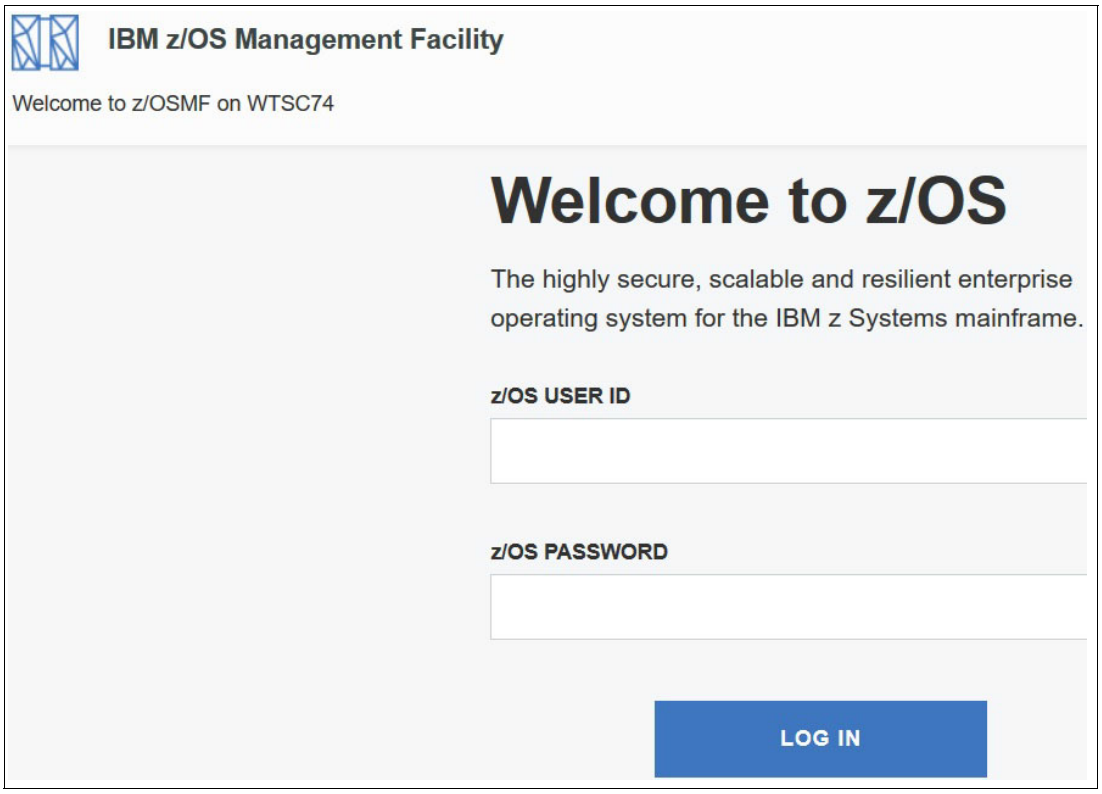


Figure 4-5 z/OSMF logon screen

Enter your provisioning user ID and password. In our case, we used the **zcxprv1** user ID.

z/OSMF now has two styles of interface. One is the original classical interface and the other is a new desktop interface, which is the one we used.

4.8.2 Select the zCX provisioning workflow

In the desktop interface, locate the Workflows icon as shown in Figure 4-6, and double-click it.

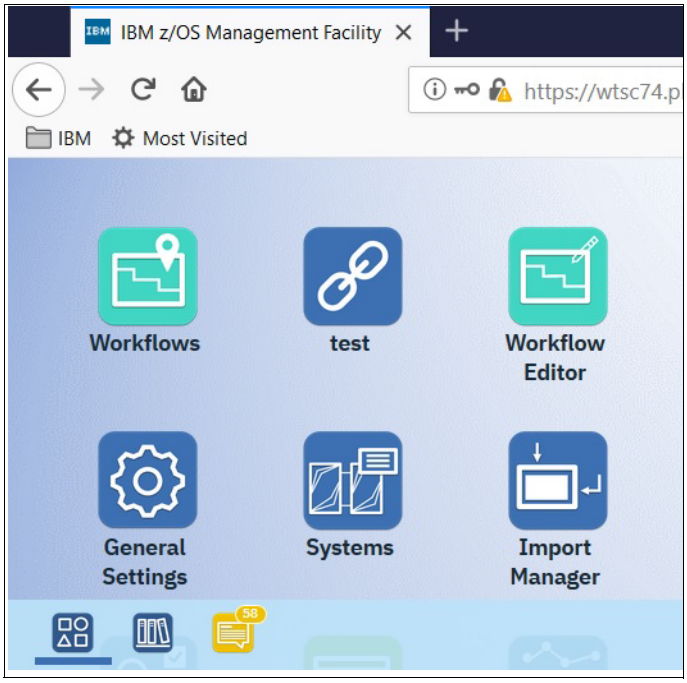


Figure 4-6 z/OSMF desktop interface

A page is displayed where you can perform workflow actions.

In that display, click **Actions** and select Create Workflow as shown in Figure 4-7 on page 62.

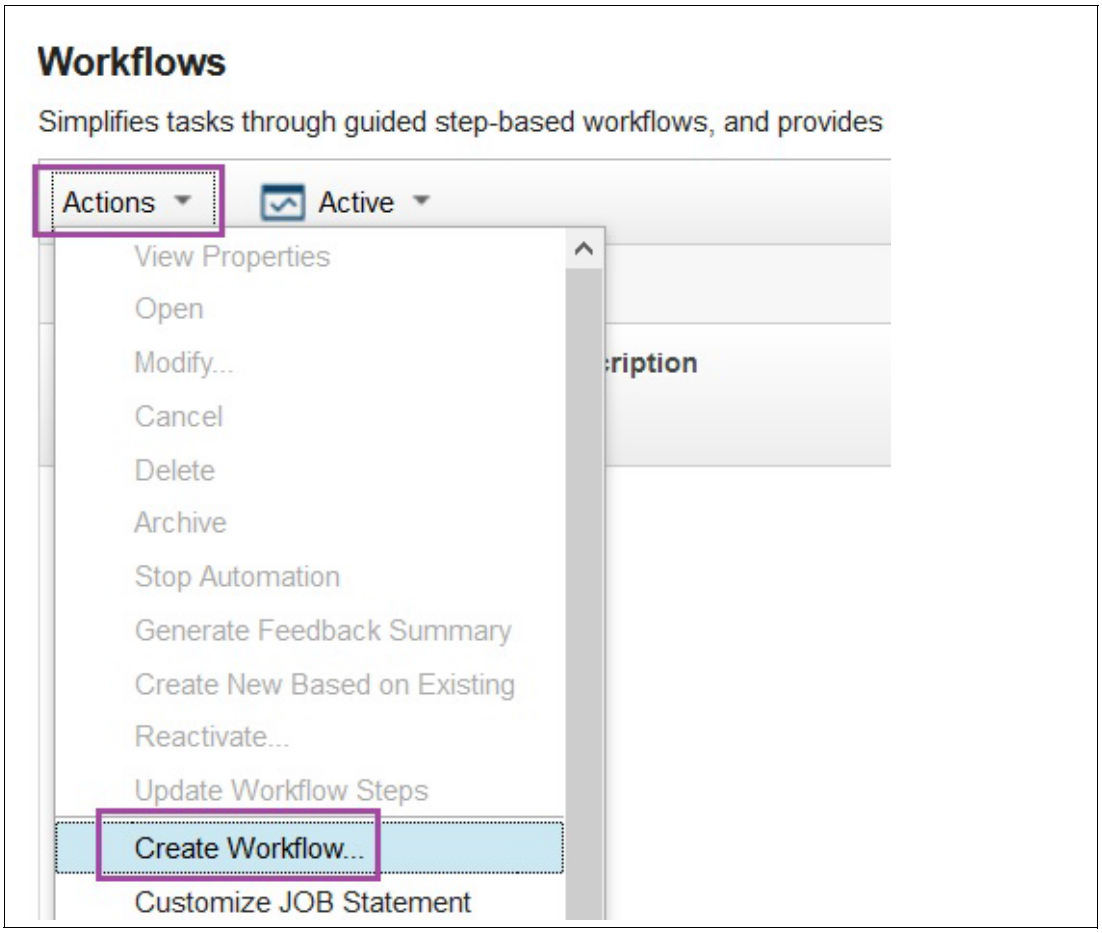


Figure 4-7 Creating new workflow

The Create Workflow panel is displayed. There, you set the input fields as shown in Table 4-6.

Table 4-6 Input values for workflow

Input field	Value	Description
Workflow definition file:	/usr/lpp/zcx_zos/workflows/provision.xml	Location of supplied zCX provisioning workflow
Workflow variable input file	/global/zcx/cfg/properties/zcxed01_workflow_vars.properties	Location of our customized property file
System	SC74	LPAR to run zCX instance on

Figure 4-8 on page 63 shows the updated panel after we entered the preceding values.

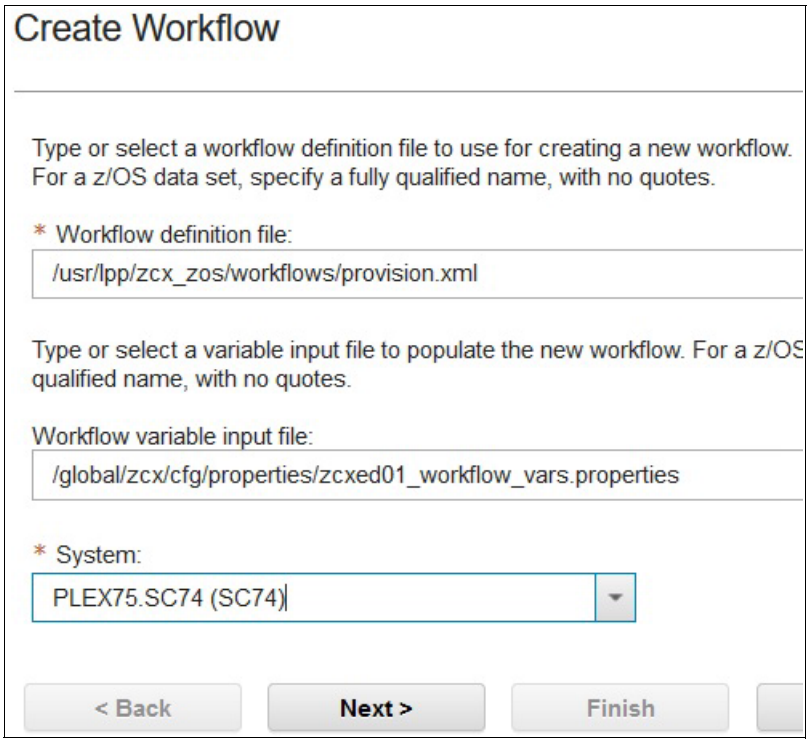


Figure 4-8 Setting initial properties to start workflow

Click **Next** to see a summary screen also called **Create Workflow**, which shows further detail about the workflow that you are about to start running. In that display, do the following actions:

- Set the name of the workflow to something meaningful. In our case, we called the workflow Provision ZCXED01.
- Select the **Assign all steps to the owner user ID** checkbox.

Figure 4-9 on page 63 shows how our display looked after we made these changes:

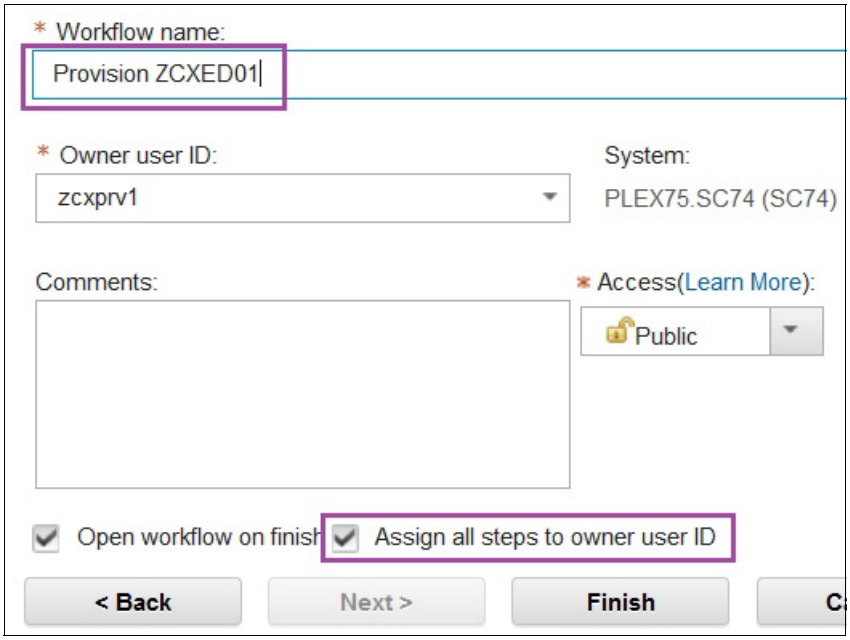


Figure 4-9 Assigning all steps to owner user ID

Click **Finish**. The display in Figure 4-10 on page 64 shows all the steps of the workflow:

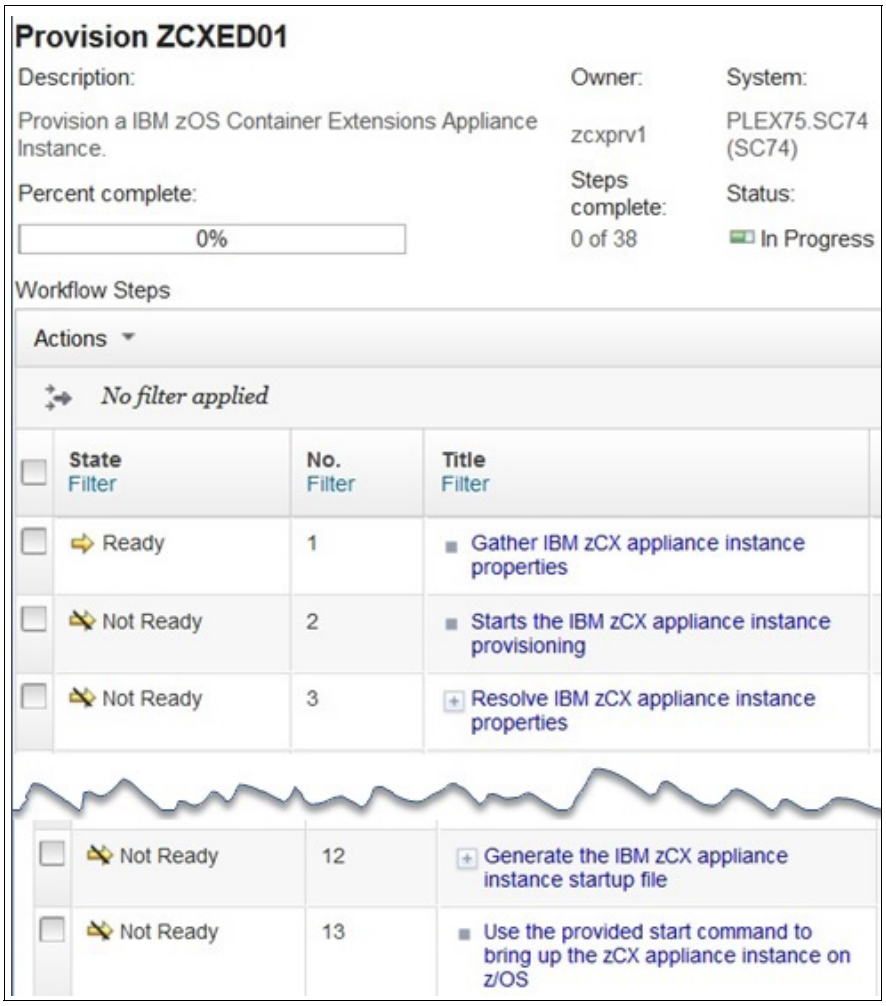


Figure 4-10 Display of steps in the workflow

4.8.3 Execute the first step of the workflow

You are now at the stage where you can run the steps of the zCX provisioning workflow.

Many of the steps run automatically. However, you must run the first step manually, because it is the step that verifies the input parameters.

Click the checkbox to the left of the first step as shown in Figure 4-11 on page 65.

Figure 4-11 Selecting the first step of the workflow to action

Then, click the **Actions** drop-down menu, and select **Perform** as shown in Figure 4-12.

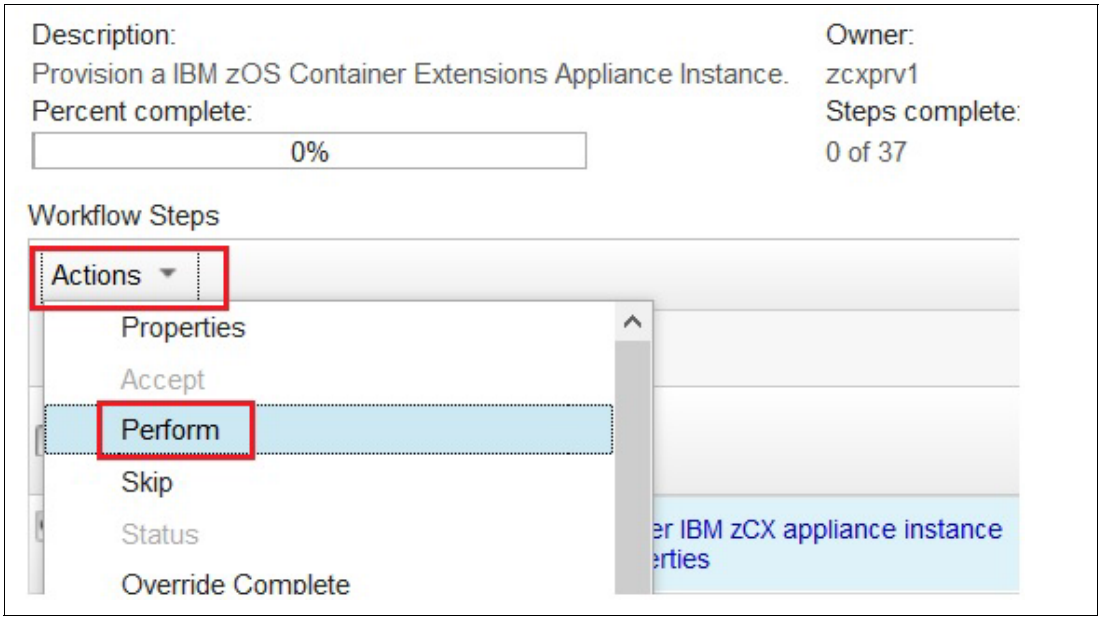


Figure 4-12 Selecting the Perform action for the first step

When you perform this first step, you are taken into the display shown in Figure 4-13 on page 66.

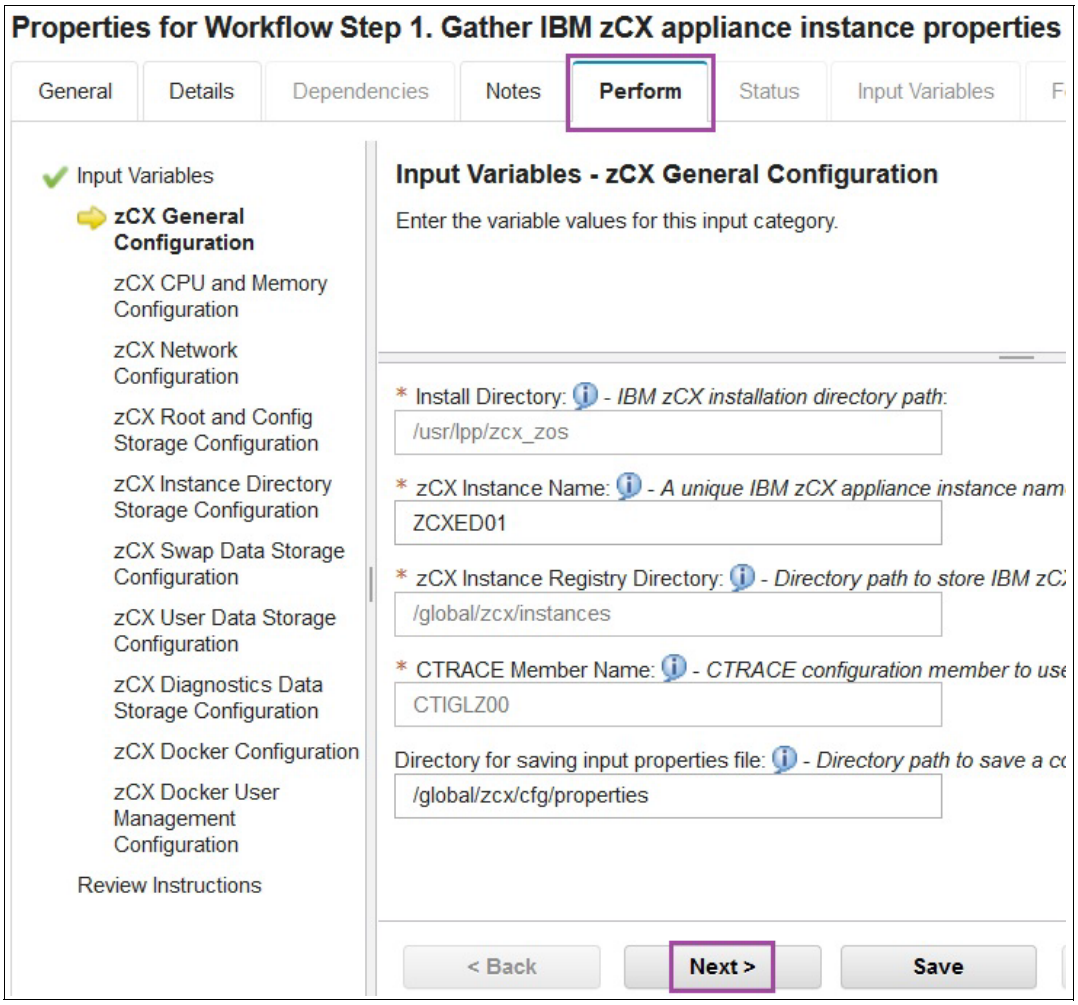


Figure 4-13 Initial display of performing the first step

This first step takes you through a number of displays to show all the variables that can be set in the workflow. As we had set up the supplied properties file with all the values we required, these values show up in the displays.

Fields that are marked with an asterisk are required fields for which a value must be supplied.

Assuming that you do the same, supplying a property file with preset values for the required properties, you can click **Next** to go through the variable input panels.

When you have completed going through all the variable checking displays, you see the display that is shown in Figure 4-14 on page 67.

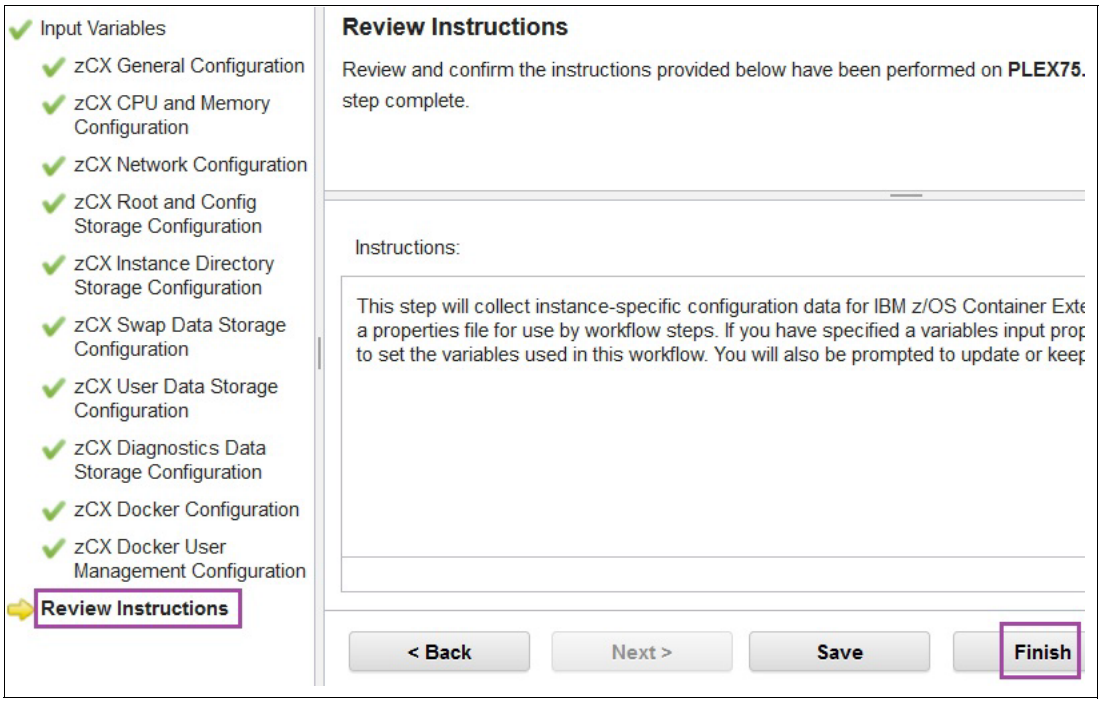


Figure 4-14 After checking values for all properties in first step

Optionally click **Save** to have z/OSMF save a copy of the properties to a file.

Click **Finish**, and you then see the display that is shown in Figure 4-15 on page 67 and see that the first step has been completed.

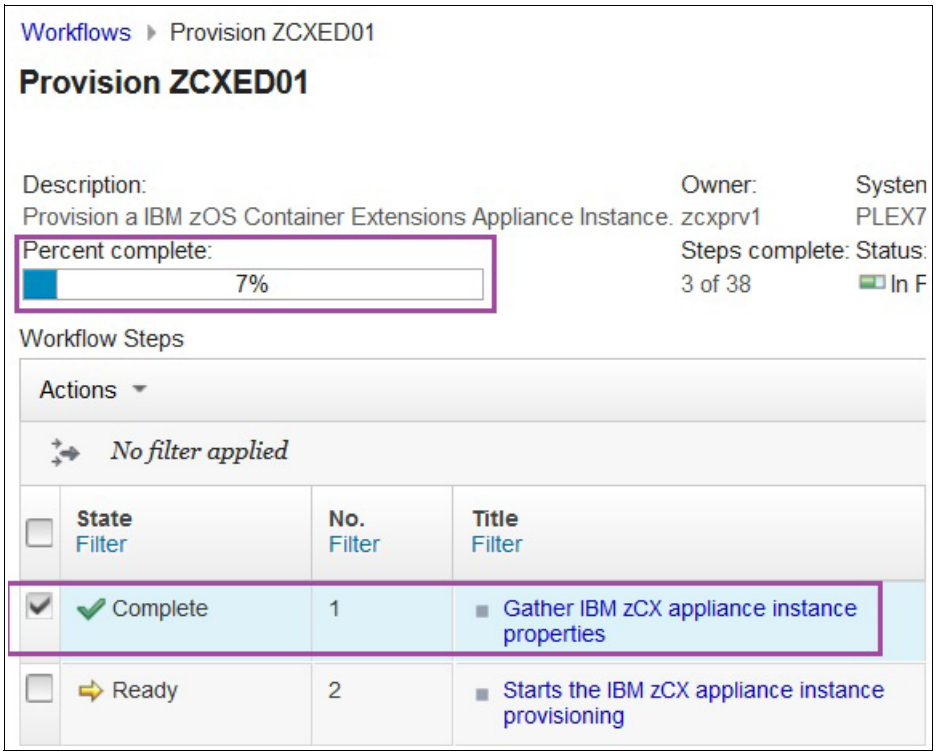


Figure 4-15 First step complete

4.8.4 Execute remaining workflow steps

You can now execute all the remaining steps of the workflow.

Unselect the first step and select the second step, then select **Perform** in the **Actions** drop-down menu. You see the display that is shown in Figure 4-16 on page 68.

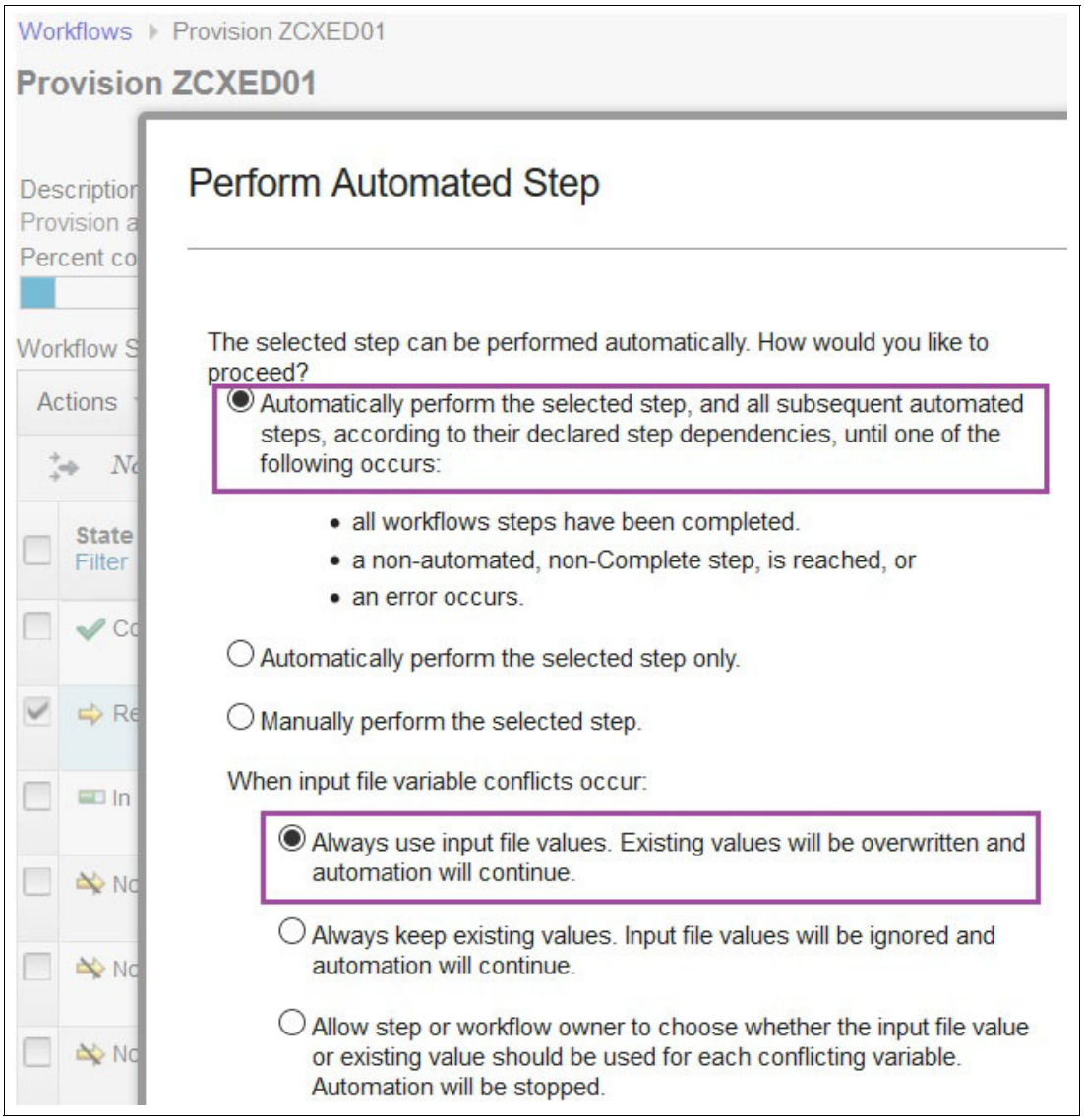


Figure 4-16 Selecting to run remaining steps automatically

The options shown as selected in Figure 4-16 should be automatically selected. Select them if that is not the case.

Selecting these options tells z/OSMF to automatically run the remaining steps, but to stop if an error occurs in a step.

Click **OK** at the bottom of the display, and the steps start to be executed.

In the workflow display, you see the completion status for the various steps change from Not Ready to Complete as shown in Figure 4-17 on page 69 and the percent complete value increase as steps complete.

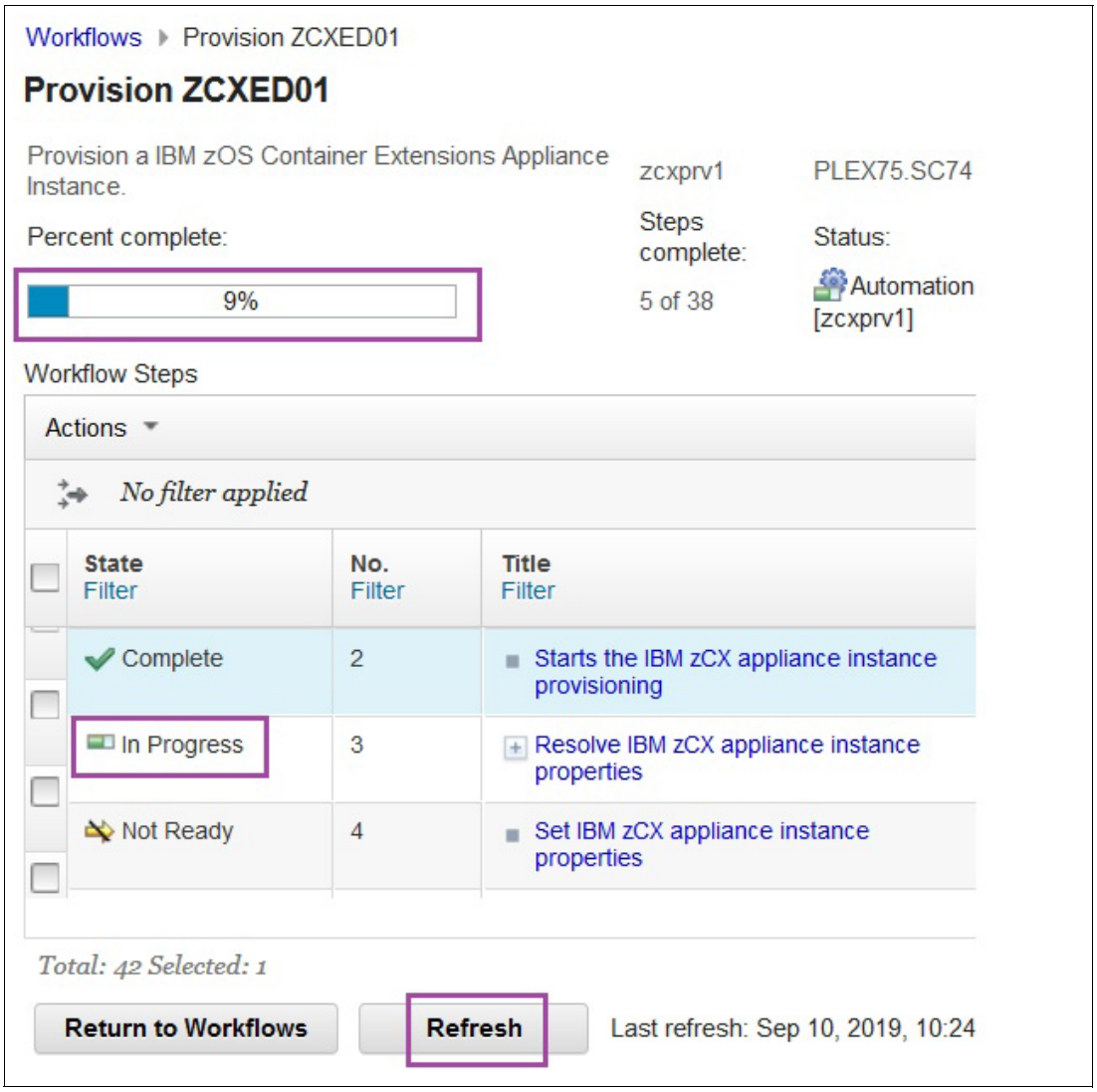


Figure 4-17 Steps in the workflow starting to execute

It takes a few minutes for all the automatic steps in the workflow to complete. You can update the status indicator of the steps by clicking **Refresh** at the bottom of the display.

When all the automatic steps complete successfully, you see this reflected in the display as shown in Figure 4-18 on page 70.

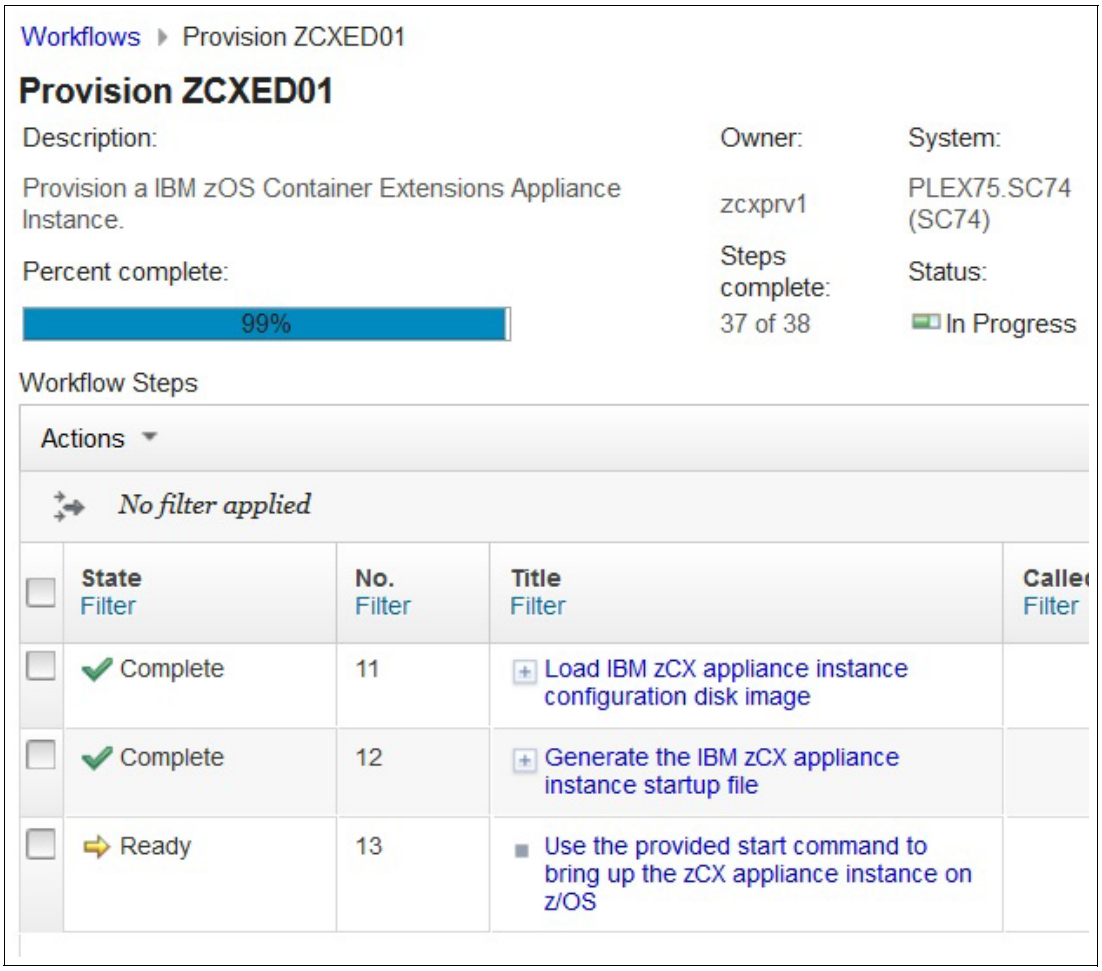


Figure 4-18 All automatic steps completed

4.8.5 Viewing created instance directory

Example 4-13 shows the files and directories that are created by the workflow.

Example 4-13 Content of directory created for the zCX instance

```
ZCXED01:
total 64
drwxrwx---   2 ZCXPRV1  ZCXPRVG        0 Sep 10 10:27 FFDC
-rw-r--r--   1 ZCXPRV1  ZCXPRVG     2214 Sep 10 10:25
ZCX-ZCXED01-user.properties
-rw-r--r--   1 ZCXPRV1  ZCXPRVG      255 Aug 27 11:09 ZCX-appliance-version
drwxrwx---   2 ZCXPRV1  ZCXPRVG     8192 Sep 10 10:27 config
-rw-rw----   1 ZCXPRV1  ZCXPRVG      637 Sep 10 10:27 start.json
drwxrwx---   2 ZCXPRV1  ZCXPRVG        0 Sep 10 10:25 tmp

ZCXED01/FFDC:
total 0

ZCXED01/config:
total 64
-rw-rw----   1 ZCXPRV1  ZCXPRVG      253 Sep 10 10:27
ZCX-ZCXED01-appliance.json
```

```
-rw-rw----   1 ZCXPRV1  ZCXPRVG      215 Sep 10 10:27
ZCX-ZCXED01-daemon.json
-rw-rw----   1 ZCXPRV1  ZCXPRVG      608 Sep 10 10:27 ZCX-ZCXED01-sshd.json
-rw-rw----   1 ZCXPRV1  ZCXPRVG      132 Sep 10 10:27 ZCX-ZCXED01-zcx.json

ZCXED01/tmp:
total 0
```

4.8.6 Obtain the command to start the zCX instance

The last step of the workflow is an information step. It provides the command that you can use to start the zCX instance that you have just created.

Click the step in the Title column to enter into the step, and click the Perform tab. You then see the command that you can use to start the zCX instance you have created as shown in Figure 4-19 on page 71.

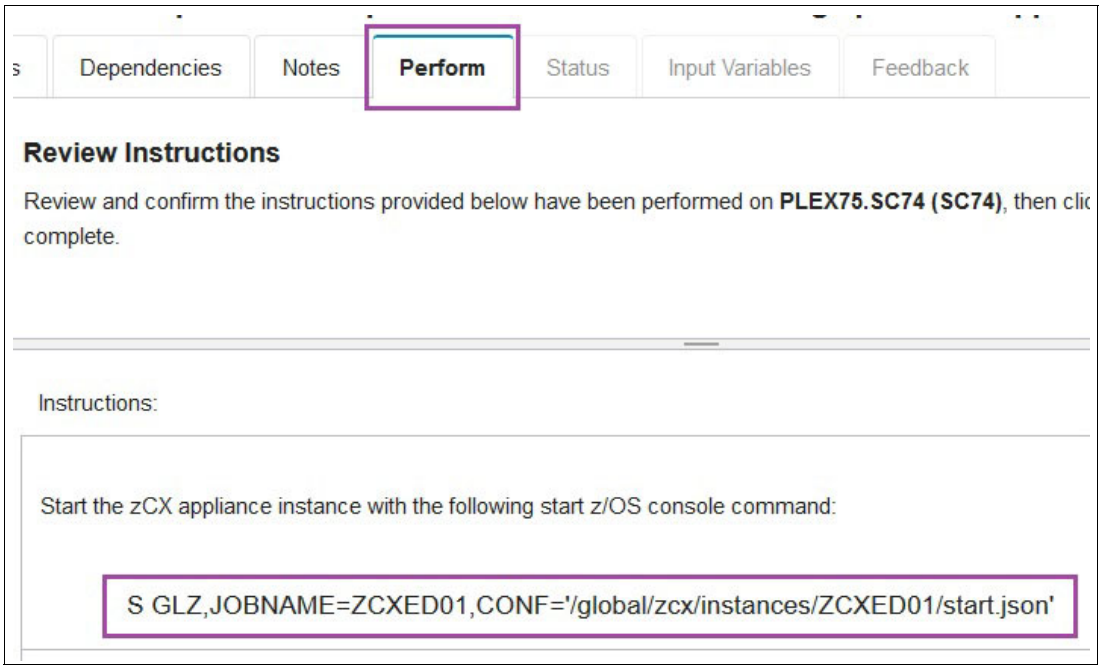


Figure 4-19 Viewing the command to start the zCX instance

In our case the command is,

```
S GLZ,JOBNAME=ZCXED01,CONF='/global/zcx/instances/ZCXED01/start.json'
```

4.9 Starting the zCX instance

Having created the zCX instance, we can now issue the command to start the instance. However, there are a few things to consider and to prepare before doing so.

4.9.1 Assigning user ID that zCX instance runs under

z/OS 2.4 includes a supplied JCL procedure called GLZ in SYS1.PROCLIB to run zCX instances.

This procedure (not counting the comments) consists of only the three lines shown in Example 4-14.

Example 4-14 The GLZ procedure JCL

```
//GLZ      PROC CONF=''
//GLZBAIN  EXEC PGM=GLZBAIN,PARM='&CONF.',REGION=0K,TIME=1440
//SYSPRINT DD SYSOUT=*
```

You could copy this JCL to create a specific member in your PROCLIB that is called the name of the zCX instance that you have created. However, this is not necessary.

Notice that we did change the supplied GLZ JCL to add the **TIME=1440** parameter. That way, the started task of the zCX instance is not automatically cancelled by JES2.

We saw in 4.8.6, “Obtain the command to start the zCX instance” on page 71 that the start command takes this form:

```
S GLZ,JOBNAME=ZCXED01,CONF='/global/zcx/instances/ZCXED01/start.json'
```

When the preceding command is issued, JES2 reads the member called GLZ from SYS1.PROCLIB and starts a started task called ZCXED01.

To make that started task run under the ZCXSTC1 user ID, we issued the RACF commands shown in Example 4-15.

Example 4-15

```
RDefine STARTED GLZ.ZCXED* STData( User (ZCXSTC1) Group (ZCXSTCG) )
SETROPTS RACLIST(STARTED) REFresh
```

The preceding RACF command causes any started task that begins with **ZCXED** to be run under the **ZCXSTC1** user ID.

4.9.2 Access to zCX instance directory by the zCX started task

The zCX instance started task needs to be able to read the configuration files in the instance directory of the zCX instance that it is supporting. It also needs to be able to create First Failure Data Capture (FFDC) files if an error occurs in the zCX started task.

As mentioned in 4.2.1, “Overview” on page 48, we recommended to run the zCX started tasks under a user ID that is different from the one that you used to provision the zCX instance.

At this point, the files in the zCX instance directory have the owner and group set to ZCXPRV1 and ZCXPRVG respectively. As a result, if you tried to start the zCX instance started task, it would fail because the ZCXSTC1 user ID would not have permission to access the zCX instance directory and files.

One way to allow the user ID under which the zCX started task is running to be able to read the instance configuration files is as follows:
Connect that user ID to the RACF group that the provisioning user IDs are defined in.

As a result, the user ID of the zCX started task would have update access to the zCX configuration files.

You would issue the following RACF command to achieve that:

```
CONNECT ZCXSTC1 GROUP(ZCXPRVG)
```

Using ACLs to control access

For some reason, you might not want to let the zCX started task have this update access to the configuration files. If so, you typically must consider using ACLs to give the ZCXSTC1 user ID read access to the directory.

In that case, you do not connect the started task user IDs to the ZCXPRVG group.

Instead, you must activate the FSSEC class in RACF if it is not already active. If the FSSEC RACF class is not already active in your z/OS LPAR, you must review whether activation of this class can have any unintended consequences.

Then, you would use the **setfacl** command to define ACL rules to allow the user ID under which the zCX started task runs to have these privileges:

- Read access to the zCX instance directory
- Update access to FFDC subdirectory

We tested this approach in our environment.

We issued **`setfacl`** commands by running the JCL shown in Example 4-16.

Example 4-16 Define ACLs

```
//DEFACL   EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//SYSTSIN DD *
  BPXBATCH SH +
  setfacl -m group:zcxstcg:+rx /global/zcx/instances/ZCXED01; +
  setfacl -m group:zcxstcg:+r /global/zcx/instances/ZCXED01/*; +
  setfacl -m group:zcxstcg:+rwx /global/zcx/instances/ZCXED01/FFDC; +
  setfacl -m f:group:zcxstcg:+r /global/zcx/instances/ZCXED01; +
  setfacl -m d:group:zcxstcg:+rx /global/zcx/instances/ZCXED01; +
  getfacl /global/zcx/instances/ZCXED01; +
  getfacl /global/zcx/instances/ZCXED01/FFDC
```

- The first two **setfacl** commands give the ZCXSTC1 user ID read access to the instance directory and files.
- The third **setfacl** command gives the ZCXSTC1 user ID write permission to the FFDC subdirectory, which is necessary to allow the started task to create FFDC files in this directory.
- The fourth and fifth **setfacl** commands add ACLs to any new directories or files that are subsequently created in the directory.

The JCL that we ran also included **getfacl** commands to display the ACLs that had been set. Example 4-17 shows the output.

Example 4-17 Displaying the ACLs that have been set

```
#file:  /global/zcx/instances/ZCXED01/
#owner: ZCXPRV1
#group: ZCXPRVG
user::rwx
```

```
group::rwx
other::---
group:ZCXSTCG:r-x

#file:  /global/zcx/instances/ZCXED01/FFDC/
#owner: ZCXPRV1
#group: ZCXPRVG
user::rwx
group::rwx
other::---
group:ZCXSTCG:rwx
```

4.9.3 Start the zCX instance

We then issued the command to start the zCX instance in SDSF as shown in Figure 4-20:

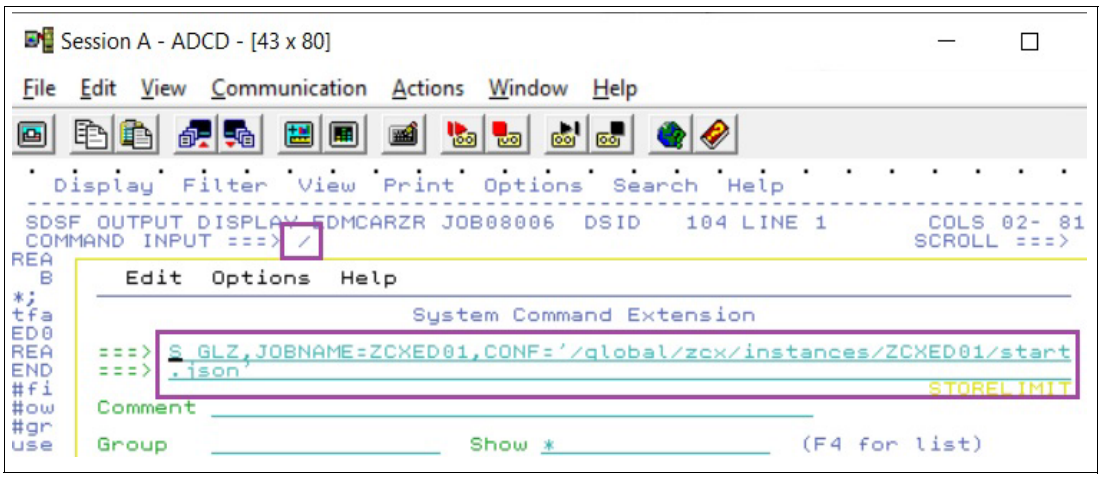


Figure 4-20 Issuing the command to start the zCX instance

Because the command requires the use mixed-case characters, you must enter a slash symbol (/) and press **Enter** to access a small **System Command Extension** window. You enter the start command in that window.

4.9.4 Check zCX startup messages

In the joblog for the zCX instance, we saw the messages shown in Example 4-18.

Example 4-18 Messages for first start of zCX instance

```
11.31.27 STC08009  $HASP373 ZCXED01  STARTED
 11.31.27 STC08009  GLZB015I Not licensed for zCX. Diag=0000.  605
    605             Temporary unlicensed use expires 12/31/2019.
 11.31.27 STC08009  IEE252I MEMBER CTIGLZ00 FOUND IN SYS1.IBM.PARMLIB
 11.31.27 STC08009  GLZV002I zCX ZCXED01 is formatting available space for use by  609
    609             DSN=ZCX.REDB.ZCXED01.ROOT
 11.31.31 STC08009  GLZV003I zCX ZCXED01 formatting complete for  610
    610             DSN=ZCX.REDB.ZCXED01.ROOT
 11.31.31 STC08009  GLZV002I zCX ZCXED01 is formatting available space for use by  611
    611             DSN=ZCX.REDB.ZCXED01.CONF
 11.31.31 STC08009  GLZV003I zCX ZCXED01 formatting complete for  612
    612             DSN=ZCX.REDB.ZCXED01.CONF
 11.31.31 STC08009  GLZV002I zCX ZCXED01 is formatting available space for use by  613
```

```
   613             DSN=ZCX.REDB.ZCXED01.SWAP1
 11.31.41 STC08009  GLZV003I zCX ZCXED01 formatting complete for  614
   614             DSN=ZCX.REDB.ZCXED01.SWAP1
 11.31.41 STC08009  GLZV002I zCX ZCXED01 is formatting available space for use by  615
   615             DSN=ZCX.REDB.ZCXED01.DATA1
 11.33.18 STC08009  GLZV003I zCX ZCXED01 formatting complete for  616
   616             DSN=ZCX.REDB.ZCXED01.DATA1
 11.33.18 STC08009  GLZV002I zCX ZCXED01 is formatting available space for use by  617
   617             DSN=ZCX.REDB.ZCXED01.DLOG1
 11.33.23 STC08009  GLZV003I zCX ZCXED01 formatting complete for  618
   618             DSN=ZCX.REDB.ZCXED01.DLOG1
 11.33.23 STC08009  GLZB001I zCX instance ZCXED01 initialization is complete. Code date
08/08/19.
 11.34.20 STC08009  GLZM004I zCX Docker services for instance ZCXED01 are available.
```

The last message is of greatest interest. It indicates that the zCX instance is ready for use, specifically this message:

```
GLZM004I zCX Docker services for instance ZCXED01 are available
```

When you start a zCX instance for the first time, the VSAM linear data sets that are used by the instance are formatted. This means that the first startup of the zCX instance will take several minutes. In our case, it took approximately two and a half minutes.

The other key messages to look for are the ones that advise that the zCX instance is listening on a port of the TCPIP address that you specified for the container. These are shown in Example 4-19:

Example 4-19 Messages showing network connectivity for the instance

```
Please connect to IBM z/OS Container Extensions Docker CLI via your SSH
client u
 sing port 8022
 The server is listening on: 129.40.23.68


 Initial Network Interface Summary:

 Interface    State      IP Address
=====================================================================
 enc0             UP               IPv4! 129.40.23.68
```

Your zCX instance is now up and ready to be used.

Note: If your zCX does not have APAR OA58236 applied, you see output for the `GLZM004I` instance before the SSH container has completed startup. If you do not have this APAR applied, you typically must wait a few minutes before you try to log on. That way, you give the SSH container time to complete startup.

4.9.5 Step failure in the workflow

If a step fails in a workflow, it is flagged as shown in Figure 4-21.

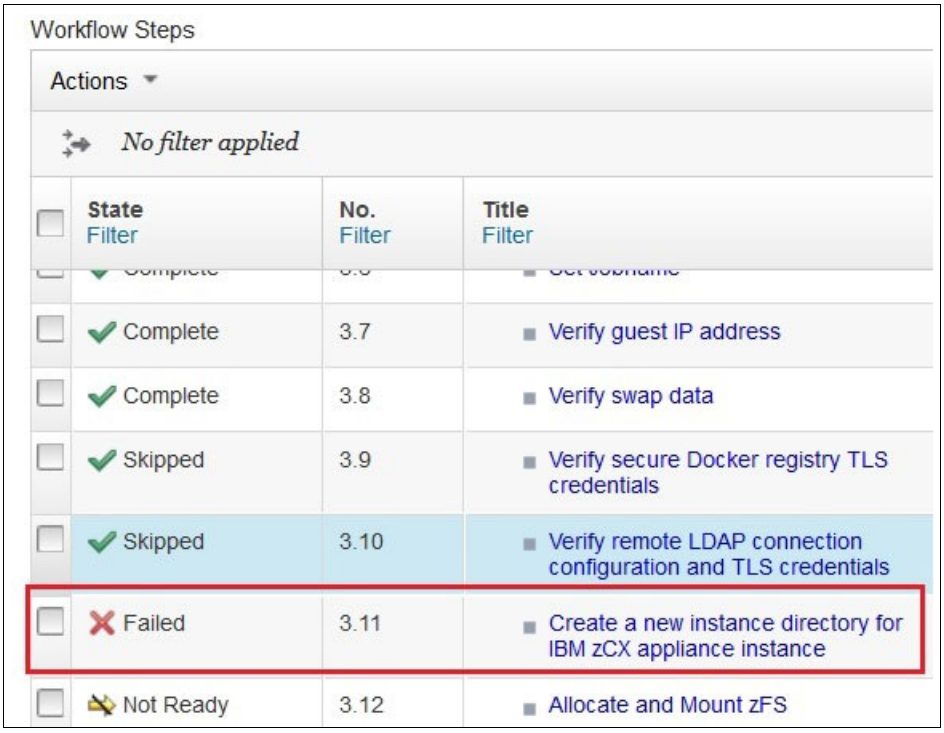


Figure 4-21 Failed step in a workflow

To determine why a step failed, click the step name. The details of the step are displayed. Select the Status tab. If the step ran a batch job, all the sections of the job are displayed in tabular format. You can click each tab to view that section of the job as shown in Figure 4-22 on page 76.

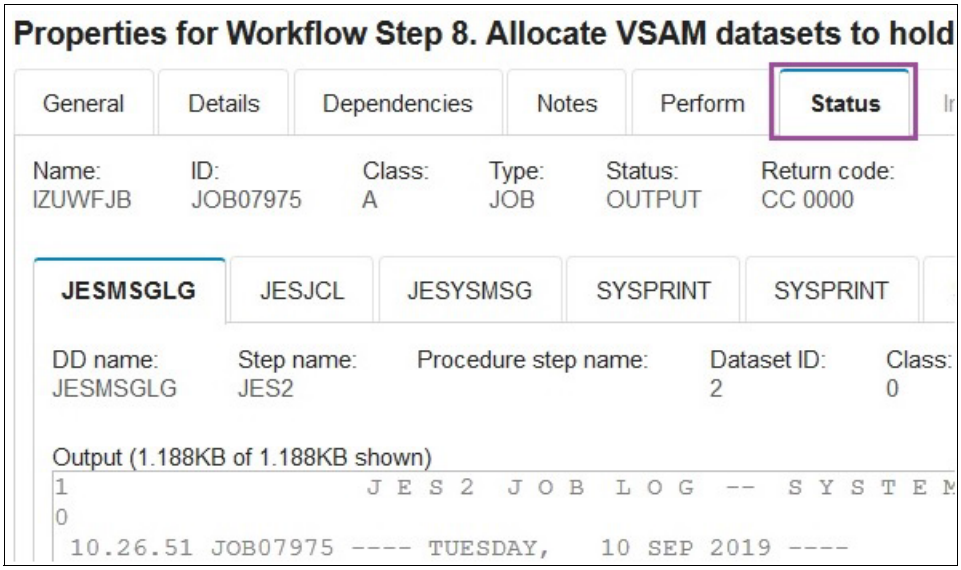


Figure 4-22 Checking job result for a step

4.10 Log on to the zCX instance

This section describes how to log on to the zCX instance where you can use ssh. Here is the procedure that we followed:

1. Log on to the USS environment of z/OS using the RACF administrator user ID that you created in 4.2.5, “Create RACF user IDs” on page 50. We logged on to USS by using ZCXADM1.
2. Enter this command to log on to the zCXinstance:
   ssh admin@sc74cn01.pbm.ihost.com -p 8022

   For this first logon, you get the response shown in Example 4-20.

Example 4-20 Response connecting to zCX instance the first time via ssh

```
The authenticity of host '[sc74cn01.pbm.ihost.com]:8022 ([129.40.23.68]:8022)'
can't be established.
ECDSA key fingerprint is SHA256:IA5Sw4oNK7wODhFGkH4ayyJNvT73m1uvQprHjqOC41M.
Are you sure you want to continue connecting (yes/no)?
```

You can disregard the warning message about the authenticity of the host, as this is expected ssh behavior. Respond ‘yes’ to the prompt, and press **Enter**. The logon should be complete and shows output similar to that in Example 4-21.

Example 4-21 Messages showing successful logon to the zCX instance

```
FOTS2274 Warning: Permanently added
'[sc74cn01.pbm.ihost.com]:8022,[129.40.23.68]:8022' (ECDSA) to the list of known
hosts.

Welcome to the IBM z/OS Container Extensions (IBM zCX) shell that provides access
to Docker commands.
For more information on how to use this shell to execute Docker commands refer to
"IBM z/OS Container Extensions Guide".
Sudo only permits specific User Management functions.  See additional
documentation for details.


admin@d377e12aa94b:~$ id
uid=1001(admin) gid=1001(admin) groups=1001(admin),27(sudo),109(docker)
```

3. Issue the id command to confirm that the user ID we have logged in to the zCX instance as is called **admin**. This was the value that we set for the property ZCX_DOCKER_ADMIN as described in 4.7.1, “Customizing the properties file” on page 58.

Now that you are logged in to the zCX instance, you can follow the steps in Chapter 5., “Your first running Docker container in zCX” on page 95 to create and run your first Docker containers.

4.11 Reconfiguring a zCX instance

In this section, we describe how to reconfigure an existing zCX instance to change settings such as number of virtual CPs and memory size.

4.11.1 What is reconfigured?

To show how to reconfigure an instance, we describe the following exercise:
Change the number of virtual CPs and available memory that the zCX instance has.

Specifically, we make the following changes:

- increase number of virtual CPs from default of 1 to 2
- increase default memory from 2 GB to 4 GB

4.11.2 Checking current settings

To check how many virtual CPs that the current zCX instance has, we issued this command:

F ZCXED01,DISPLAY,CONFIG

Example 4-22 shows the output of this command, including the current number of virtual CPs and allocated memory.

Example 4-22 Output from lscpu command

```
GLZC003I Configuration information for zCX instance ZCXED01
File Path: /global/zcx/instances/ZCXED01/start.json
FFDC Path: /global/zcx/instances/ZCXED01/FFDC
Dump Path: /global/zcx/instances/ZCXED01/FFDC/zcx-guest.dmp
Memory size:           2GB
Number of CPUs:        1
Number of Disks:       5
Number of Networks:    1
```

4.11.3 Create reconfiguration zCX workflow

To create the reconfiguration zCX workflow, in z/OSMF select the option to create a new workflow as described in 4.8.2, “Select the zCX provisioning workflow” on page 61.

Then, in the field called Workflow definition file, enter the location of the supplied zCX reconfiguration workflow, which on our system was here:

/usr/lpp/zcx_zos/workflows/reconfigure.xml

Supply the location of the property file that was used to provision the zCX instance, and select the same z/OS LPAR. The display should look similar to Figure 4-23.

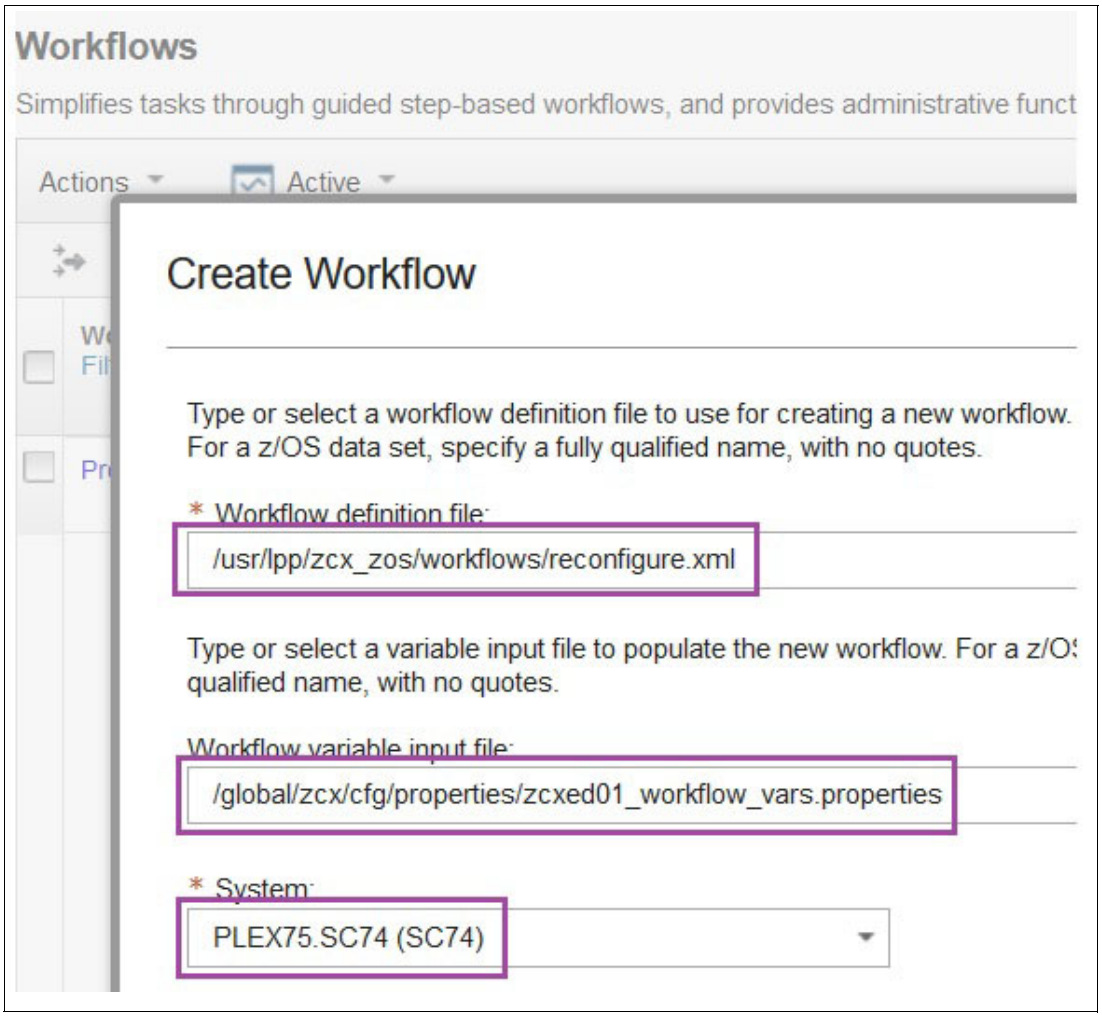


Figure 4-23 Setting up reconfiguration workflow

Click **Next**. You see a screen like the one in Figure 4-9 on page 63. We set the Workflow name field to **Reconfigure ZCXED01** and selected the **Assign all steps to owner user ID** checkbox, then clicked **Finish**.

4.11.4 Execute the reconfiguration workflow

You are now ready to run the workflow. Figure 4-24 on page 80 shows the initial view of the workflow.

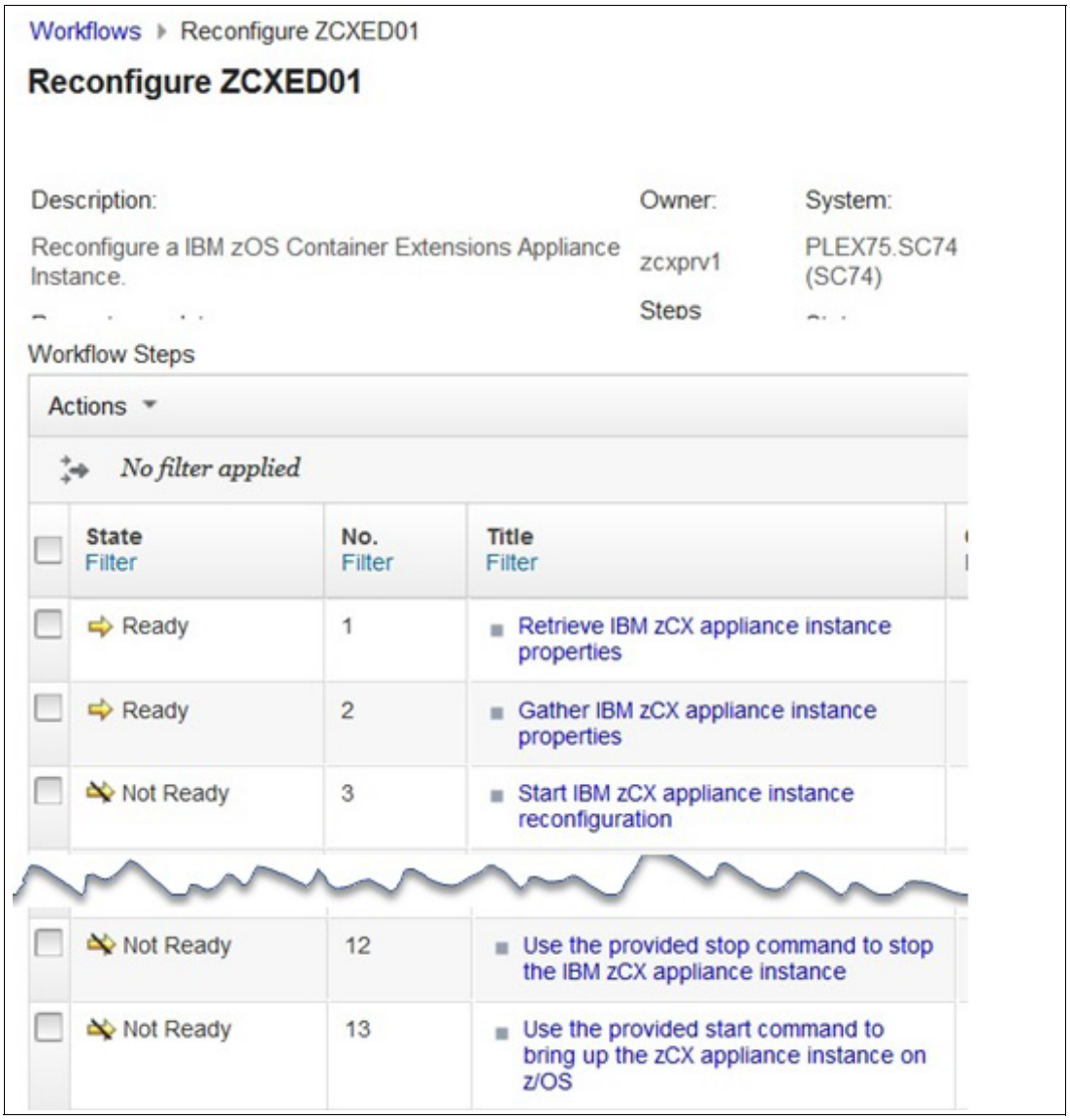


Figure 4-24 Steps of the zCX reconfiguration workflow

The first two steps are manual steps.

Select the checkbox for the first step, then select **Actions > Perform** in the drop-down menu.

Click **Next** to perform the actions of the step, which is to eventually run a batch job to collect the current zCX instance settings.

Next, select the second step and select the **Perform** action for this step. During this step, you can manually change settings of the zCX instance.

Step through the actions of the step until you get to the display for zCX CPU and Memory configuration. We increased the number of virtual CPs from 1 to 2 and the memory from 2 GB to 4 GB as shown in Figure 4-25 on page 81.

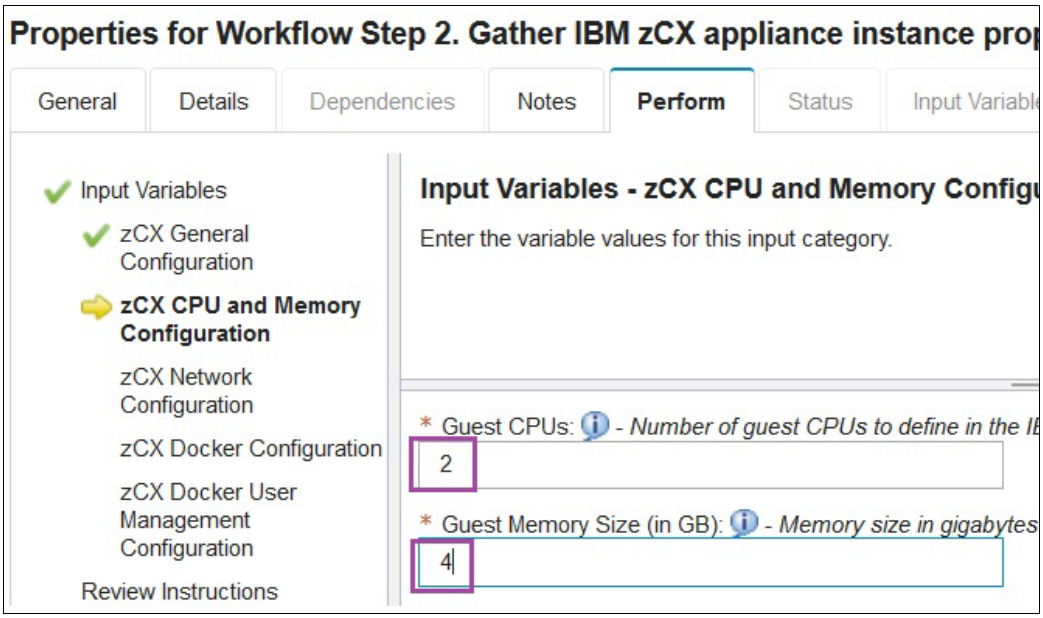


Figure 4-25 Adjusting the virtual CPs and memory size

Step through the remaining actions and click **Finish** to complete the step.

Next, do step 3, for which you get a prompt that is similar to that shown in Figure 4-16 on page 68. Click **OK** to allow the execution of this and subsequent steps. The steps should all run to completion. Click **Refresh** to update the status of the steps, until you see display shown in Figure 4-26.

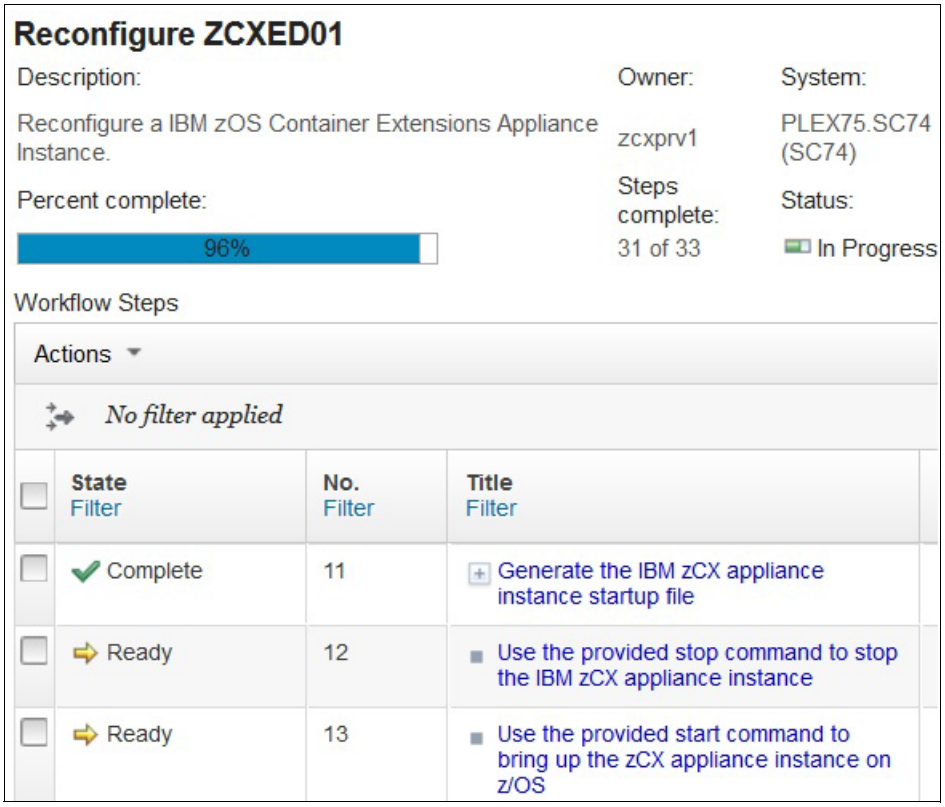


Figure 4-26 All automated steps of the reconfiguration workflow completed

4.11.5 Stop and start the zCX instance

You can now restart the zCX instance to pick up the changes. The commands to do this can be viewed by doing **Perform** action on the last two steps of the workflow.

We issued this command to stop our zCX instance: **`P ZCXED01`**

Example 4-23 shows the messages that are written when the zCX instance terminates:

Example 4-23 zCX instance shutdown messages

```
08.34.48 STC08009  GLZC006I A stop command for zCX instance ZCXED01 has been accepted.
 08.34.50 STC08009  GLZM005I zCX Docker services for instance ZCXED01 are not available.
 08.36.23 STC08009  GLZB002I zCX instance ZCXED01 has ended.  974
 08.36.24 STC08009  Jobname  Procstep Stepname  CPU Time      EXCPs     RC
 08.36.24 STC08009  ZCXED01  STARTING GLZBAIN   00:00:03     15,819     00
 08.36.24 STC08009  $HASP395 ZCXED01  ENDED - RC=0000
```

We then issued the start command shown in Section 4.8.6, “Obtain the command to start the zCX instance” on page 71.

4.11.6 Verifying reconfiguration changes

After the zCX instance had started, we issued the command to display the configuration and confirmed that our changes had taken effect as shown in Example 4-24.

Example 4-24 Display showing changes have taken effect

```
GLZC003I Configuration information for zCX instance ZCXED01
File Path: /global/zcx/instances/ZCXED01/start.json
FFDC Path: /global/zcx/instances/ZCXED01/FFDC
Dump Path: /global/zcx/instances/ZCXED01/FFDC/zcx-guest.dmp
Memory size:           4GB
Number of CPUs:        2
Number of Disks:       5
Number of Networks:    1
```

4.12 Adding a volume to a zCX instance

In this section, we describe how to add a volume to an existing zCX instance.

4.12.1 Overview

The VSAM linear data sets that are used in a zCX instance are not allowed to increase in size through the addition of extents. To make more disk space available to the zCX instance, you must instead run the zCX workflow to provision a new VSAM linear data set to add to the zCX instance.

When the zCX instance is restarted to pick up this change, the new VSAM linear data set is in effect added to the existing VSAM linear data set. As a result, the zCX instance treats this as a single disk area that it can use.

4.12.2 Checking default disk space allocation

When we created the zCX instance, we used the default settings, which created five VSAM linear data sets.

To check what disk space is currently available to the zCX instance, log on to the zCX instance and issue the command **lsblk**. Example 4-25 shows the output of this command our zCX instance.

Example 4-25 Output from lsblk command

```
NAME   MAJ:MIN RM    SIZE RO TYPE MOUNTPOINT
vda    252:0    0    3.9G  0 disk
|-vda1 252:1         512M  0 part
`-vda2 252:2    0    3.4G  0 part
vdc    252:32   0      2G  0 disk
`-vdc1 252:33   0      2G  0 part
vdd    252:48   0 1000.6M  0 disk
vde    252:64   0   19.5G  0 disk
`-vde1 252:65   0   19.5G  0 part
```

Each of the main entries in the NAME column corresponds to one of the VSAM linear data sets. There are only four entries, because the VSAM linear data set that contains the config is dropped shortly after startup.

The entry for **vde** is the large 20 GB VSAM linear data set that was allocated when the zCX instance was created.

Note: The values in the NAME column can change when the zCX instance is restarted.

To see details, you can also issue the following command:
df -kh
Example 4-26 show typical output for this command.

Example 4-26 Output of the df-kh command

```
Filesystem                      Size  Used Avail Use% Mounted on
overlay                          20G  385M   18G   3% /
tmpfs                            64M     0   64M   0% /dev
tmpfs                           1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/mapper/data_VG_15403-data   20G  385M   18G   3% /home
shm                              64M     0   64M   0% /dev/shm
tmpfs                           1.9G  336K  1.9G   1% /run/docker.sock
devtmpfs                        1.8G     0  1.8G   0% /dev/tty
tmpfs                           1.9G     0  1.9G   0% /proc/scsi
tmpfs                           1.9G     0  1.9G   0% /sys/firmware
```

In Example 4-26, the entry for **/dev/mapper/data_VG_15403-data** corresponds to the 20 GB VSAM linear data set.

4.12.3 Create add data disks zCX workflow

To create the **add data disks** zCX workflow, in z/OSMF select the option to create a new workflow as described in 4.8.2, “Select the zCX provisioning workflow” on page 61.

Then, in the **Workflow definition file** field, enter the location of the supplied zCX **add data disks** workflow, which on our system was located here:

```
/usr/lpp/zcx_zos/workflows/add_data_disks.xml
```

Enter the location of the property file that was used to provision the zCX instance, and select the same z/OS LPAR.

The display should look similar to that in Figure 4-27 on page 84.

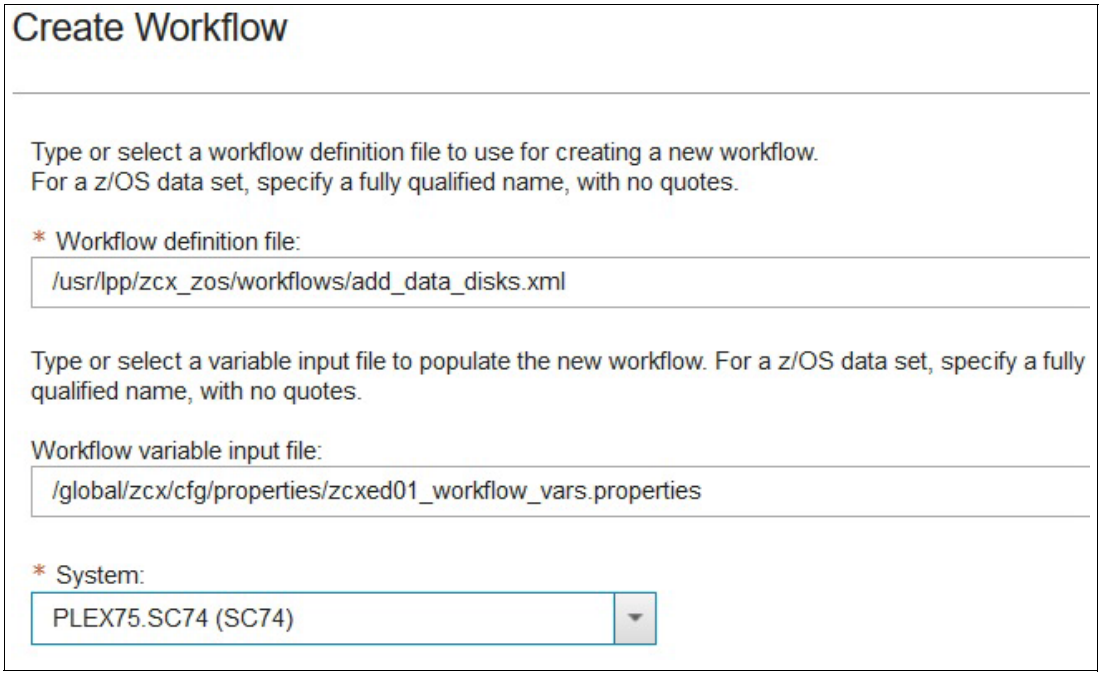


Figure 4-27 Creating add data disks workflow

Click **Next** and you see a screen that is similar to the one in Figure 4-9 on page 63. We set the **Workflow name** field to **Add Data Disks to ZCXED01**, selected the **Assign all steps to owner user ID** checkbox, and then clicked **Finish**.

4.12.4 Run the Add Data Disks workflow

You are now ready to run the workflow. Figure 4-28 on page 85 shows the initial view of the workflow.

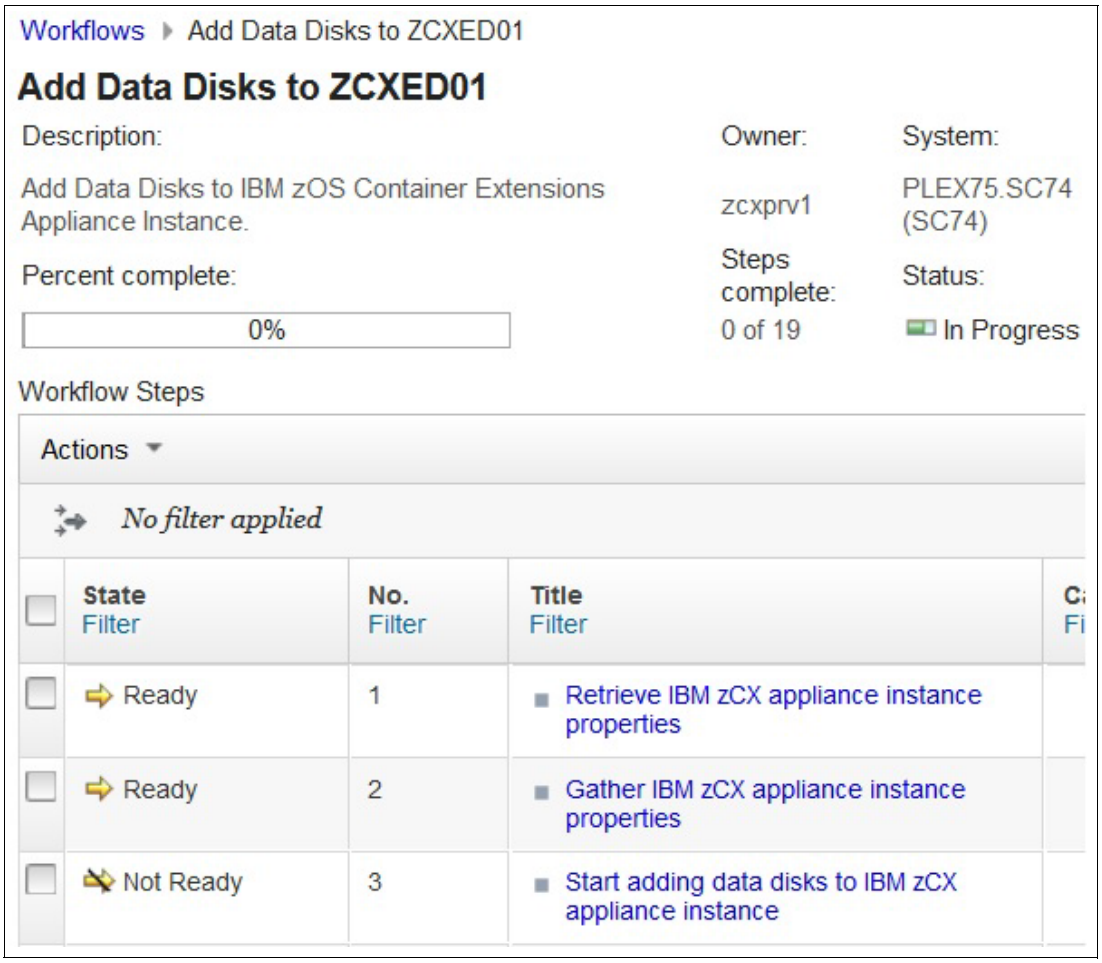


Figure 4-28 Steps of the Add Data Disks workflow

The first two steps are manual steps.

1. Select the checkbox for the first step, and select **Action > Perform** in the drop-down menu.
2. Click **Next** to perform the actions of the step, which is to eventually run a batch job to collect the current zCX instance settings.
3. Select the second step of the workflow, and do a **Perform** action on it. This is the step where you can manually define additional data volumes to be added to the zCX instance.
4. Step through the actions of the step until you get to the display for zCX Swap Data Storage Configuration. We specified to add one on a swap volume of 2 GB in size as shown in Figure 4-29 on page 86.

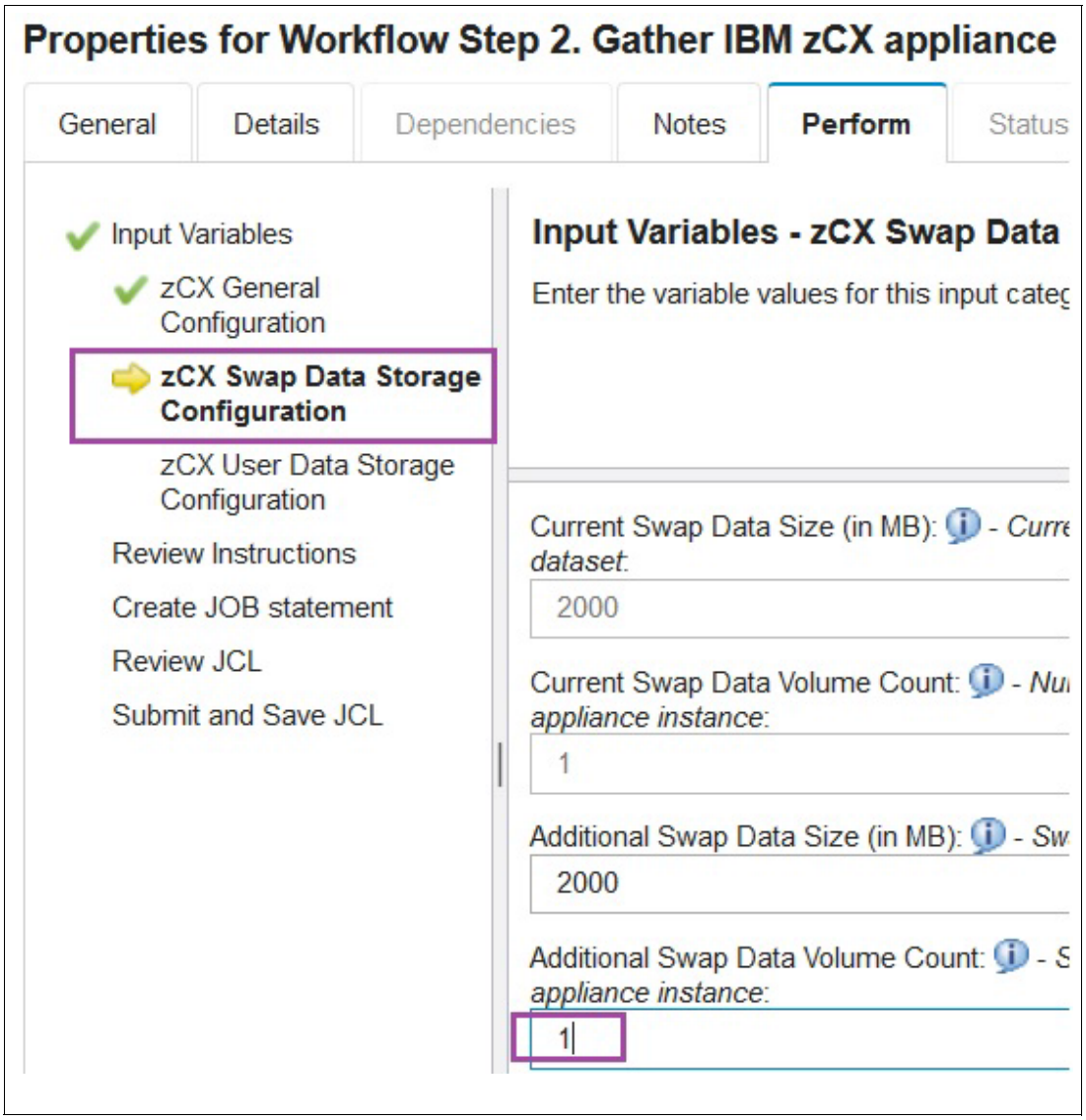


Figure 4-29 Adding one additional Swap Data Volume

5. In the zCX User Data Storage Configuration step, we added one User Data Volume of 20 GB as shown in Figure 4-30 on page 87.

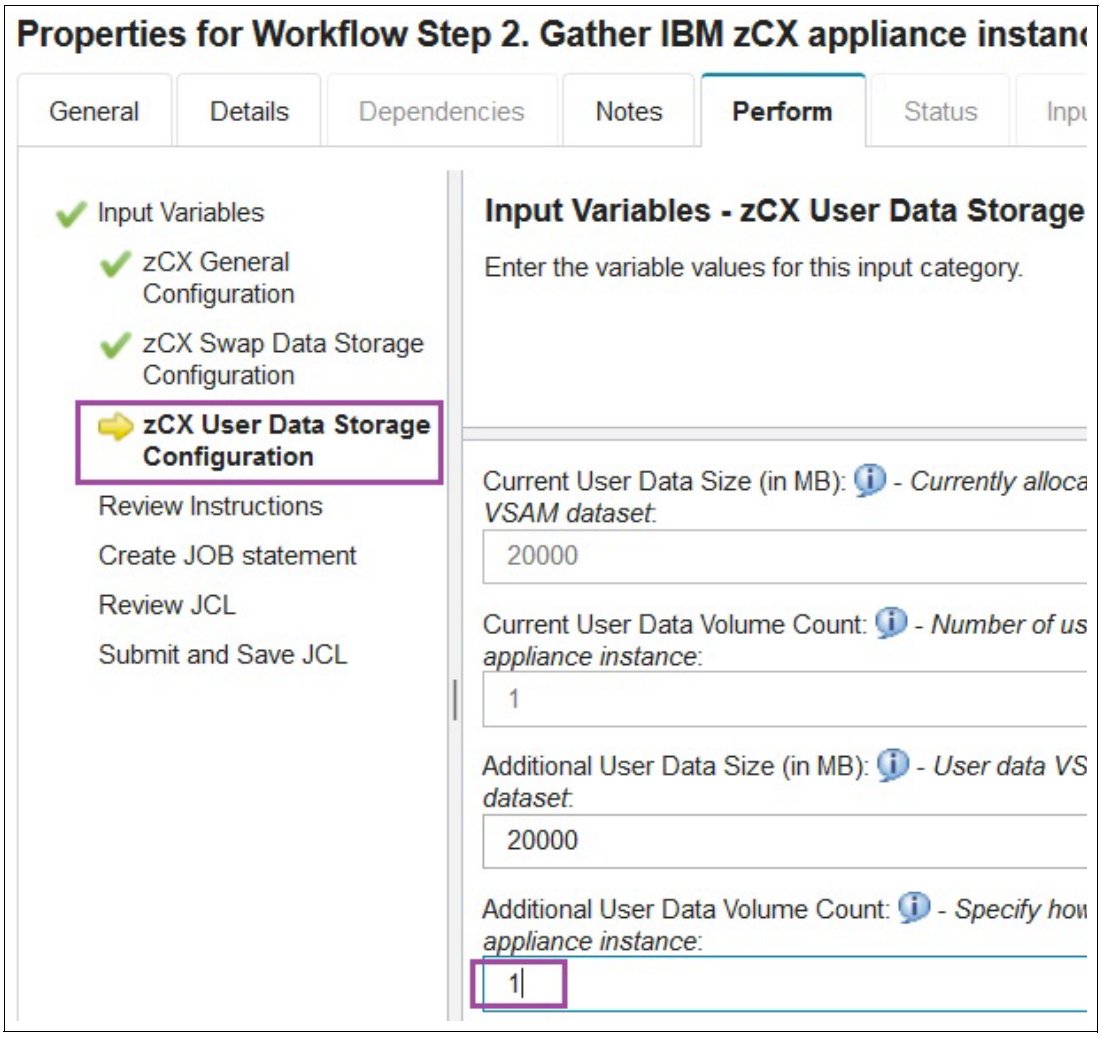


Figure 4-30 Adding one User Data Volume of 20 GB

6. Next, do the third step of the workflow, after which you see a prompt that is similar to the one in Figure 4-16 on page 68. Click **OK** to allow execution of this and subsequent steps.

The steps should all run to completion, click **Refresh** to update the status of the steps, until you see display shown in Figure 4-31 on page 88.

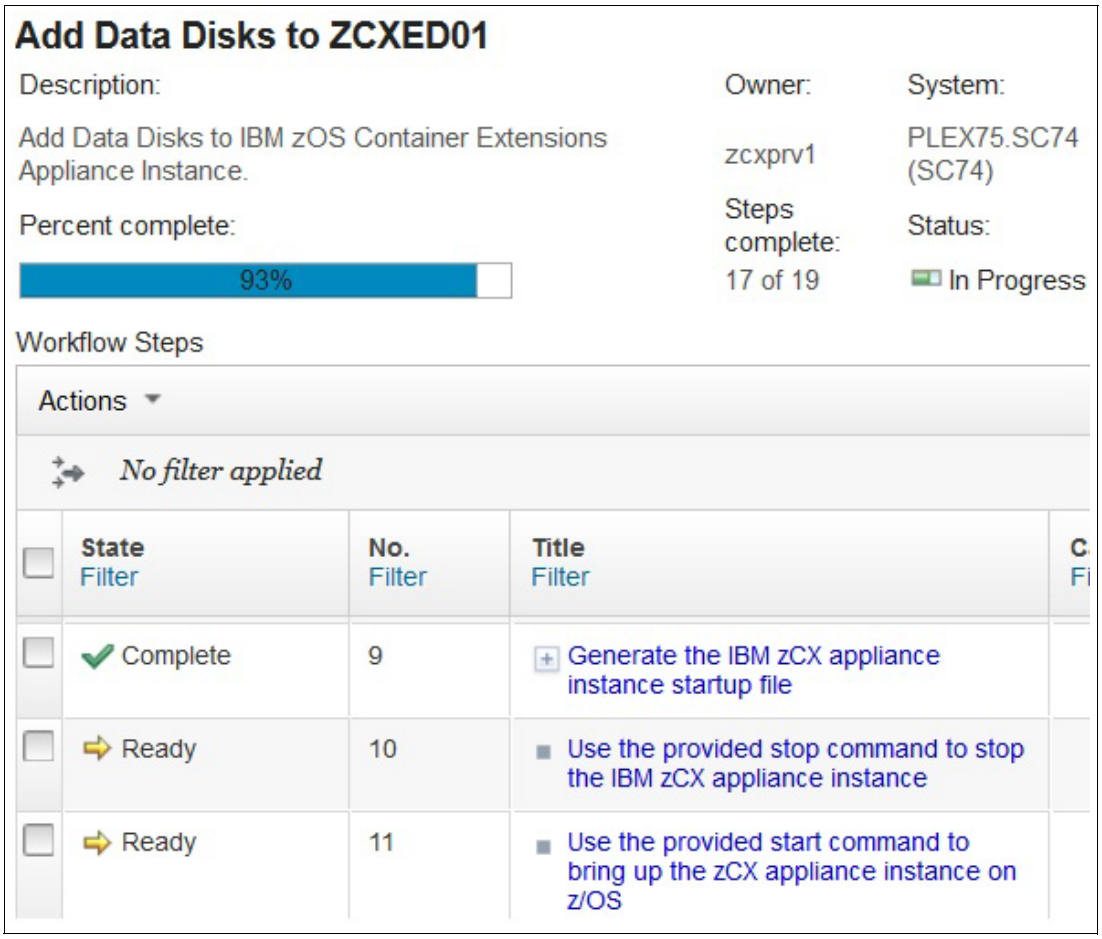


Figure 4-31 Completion of Add Data Disks workflow

In checking the VSAM data sets for our instance, we confirmed that we now had two new VSAM data sets called:

```
ZCX.REDB.ZCXED01.DATA2
ZCX.REDB.ZCXED01.SWAP2
```

4.12.5 Stop and start the zCX instance

The zCX instance can now be restarted to pick up the changes. The commands to do this can be viewed by doing a **Perform** action on the last two steps of the workflow.

We issued this command to stop our zCX instance: **`P ZCXED01`**

We then issued the start command shown in 4.8.6, “Obtain the command to start the zCX instance” on page 71.

4.12.6 Verifying Add Data Disk changes

When the zCX instance is restarted, it detects that the configuration has been changed by the addition of new volumes and that it must format these new volumes. Messages similar to the ones in Example 4-27 report the changes.

Example 4-27 Formatting of new volumes during zCX instance startup

```
10.46.44 STC08186  GLZV002I zCX ZCXED01 is formatting available space for use by  667
    667             DSN=ZCX.REDB.ZCXED01.SWAP2
 10.46.54 STC08186  GLZV003I zCX ZCXED01 formatting complete for  668
    668             DSN=ZCX.REDB.ZCXED01.SWAP2
 10.46.54 STC08186  GLZV002I zCX ZCXED01 is formatting available space for use by  669
    669             DSN=ZCX.REDB.ZCXED01.DATA2
 10.48.32 STC08186  GLZV003I zCX ZCXED01 formatting complete for  673
    673             DSN=ZCX.REDB.ZCXED01.DATA2
```

We then logged on to the zCX instance and issued the **lsblk** command, which produced the output shown in Example 4-28.

Example 4-28 Output of lsblk command

```
NAME   MAJ:MIN RM    SIZE RO TYPE MOUNTPOINT
vda    252:0    0    3.9G  0 disk
|-vda1 252:1         512M  0 part
`-vda2 252:2    0    3.4G  0 part
vdc    252:32   0      2G  0 disk
`-vdc1 252:33   0      2G  0 part
vdd    252:48   0      2G  0 disk
`-vdd1 252:49   0      2G  0 part
vde    252:64   0   19.5G  0 disk
`-vde1 252:65   0   19.5G  0 part
vdf    252:80   0   19.5G  0 disk
`-vdf1 252:81   0   19.5G  0 part
vdg    252:96   0 1000.6M  0 disk
```

The output shows the additional 2 GB swap volume and the 20 GB user volume.

The **df -kh** command generated the output that is shown in Example 4-29.

Example 4-29 Output of df -kh command

```
Filesystem                      Size  Used Avail Use% Mounted on
overlay                          40G  385M   37G   2% /
tmpfs                            64M     0   64M   0% /dev
tmpfs                           1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/mapper/data_VG_15403-data   40G  385M   37G   2% /home
shm                              64M     0   64M   0% /dev/shm
tmpfs                           1.9G  408K  1.9G   1% /run/docker.sock
devtmpfs                        1.8G     0  1.8G   0% /dev/tty
tmpfs                           1.9G     0  1.9G   0% /proc/scsi
tmpfs                           1.9G     0  1.9G   0% /sys/firmware
```

This output confirms that the available space for the **/dev/mapper/data_VG_15403-data** mount point is now 40 GB.

4.13 Deprovisioning a zCX instance

In this section, we describe how to deprovision a zCX instance. The deprovisioning of a zCX instance deletes all VSAM and zFS files that were created for the instance and removes the instance directory.

4.13.1 Create deprovisioning zCX workflow

To create the deprovisioning zCX workflow, in z/OSMF select the option to create a new workflow as described in 4.8.2, “Select the zCX provisioning workflow” on page 61.

Then, in the **Workflow definition file** field, enter the location of the supplied zCX deprovision workflow. On our system the workflow was located here:

```
/usr/lpp/zcx_zos/workflows/deprovision.xml
```

Enter the location of the property file that was used to provision the zCX instance, and select the same z/OS LPAR. The resulting display should look similar to Figure 4-32.

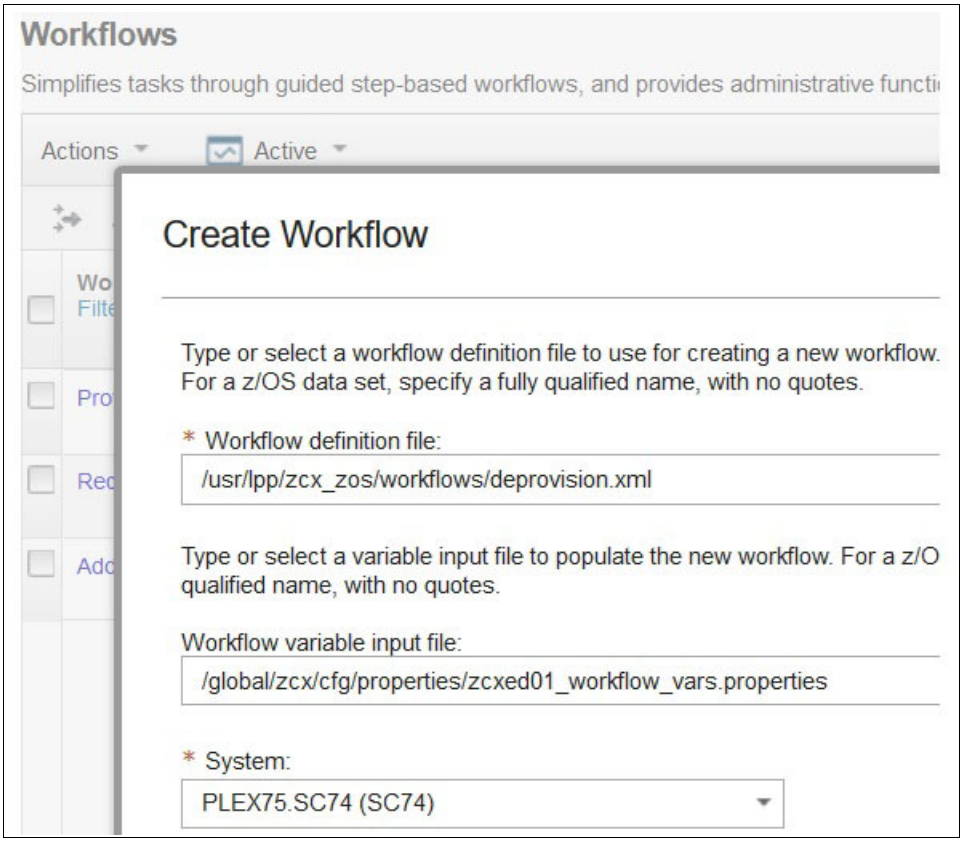


Figure 4-32 Creating deprovision workflow

Click **Next**. You see a screen like the one in Figure 4-9 on page 63. We set the Workflow name field to **Deprovision ZCXED01**, selected the **Assign all steps to owner user ID** checkbox, and then clicked **Finish**.

4.13.2 Execute the deprovision workflow

You are now ready to run the workflow. Figure 4-33 on page 91 shows the initial view of the workflow.

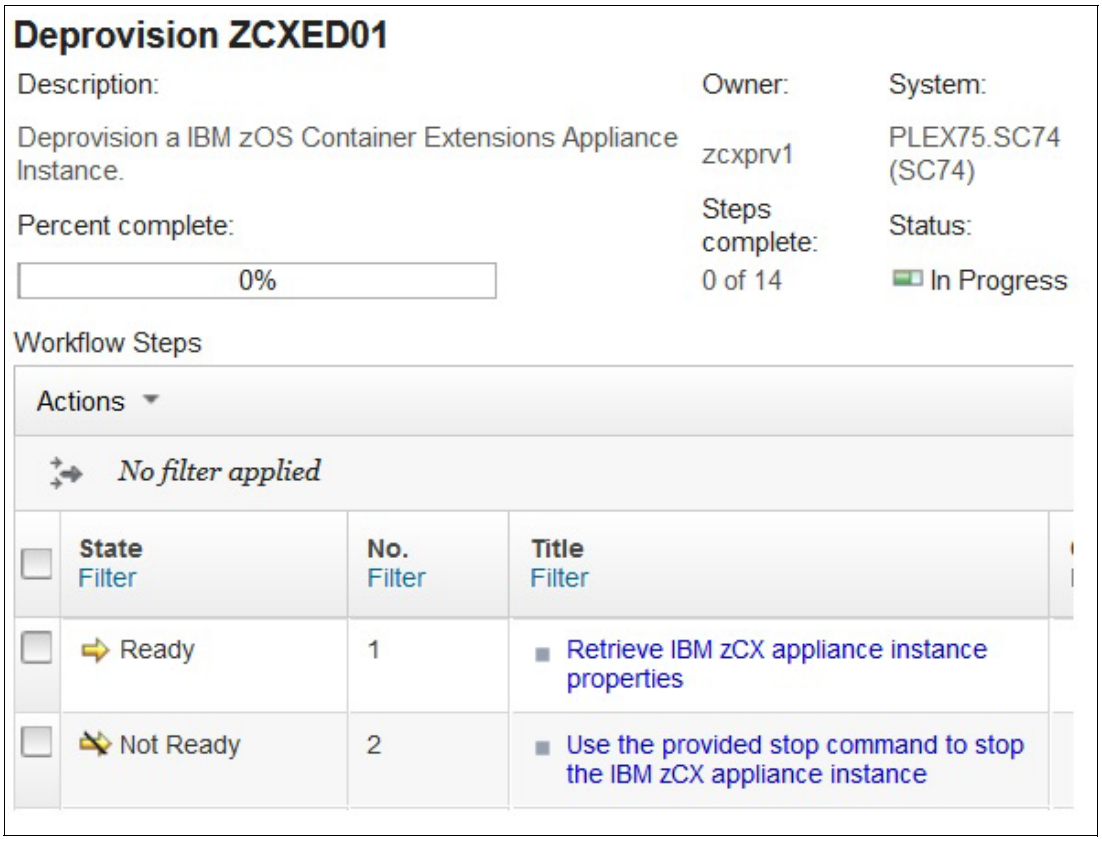


Figure 4-33 Steps of the deprovision workflow

The first step is a manual step.

1. Select the checkbox for Step 1 of the workflow, then select **Actions > Perform** in the drop-down menu.
2. Click **Next** to run the actions of the step, which eventually run a batch job to collect the current zCX instance settings.
3. Perform step 2 of the workflow, which shows the command that you run to stop the zCX instance. In our case, this was the command: `P ZCXED01`
4. Perform step 3 of the workflow. You see a prompt that is similar to the one in Figure 4-16 on page 68. Click **OK** to allow execution of this and subsequent steps.
5. After the steps run to completion, click **Refresh** to update the status of the steps, until you see display shown in Figure 4-34 on page 92.

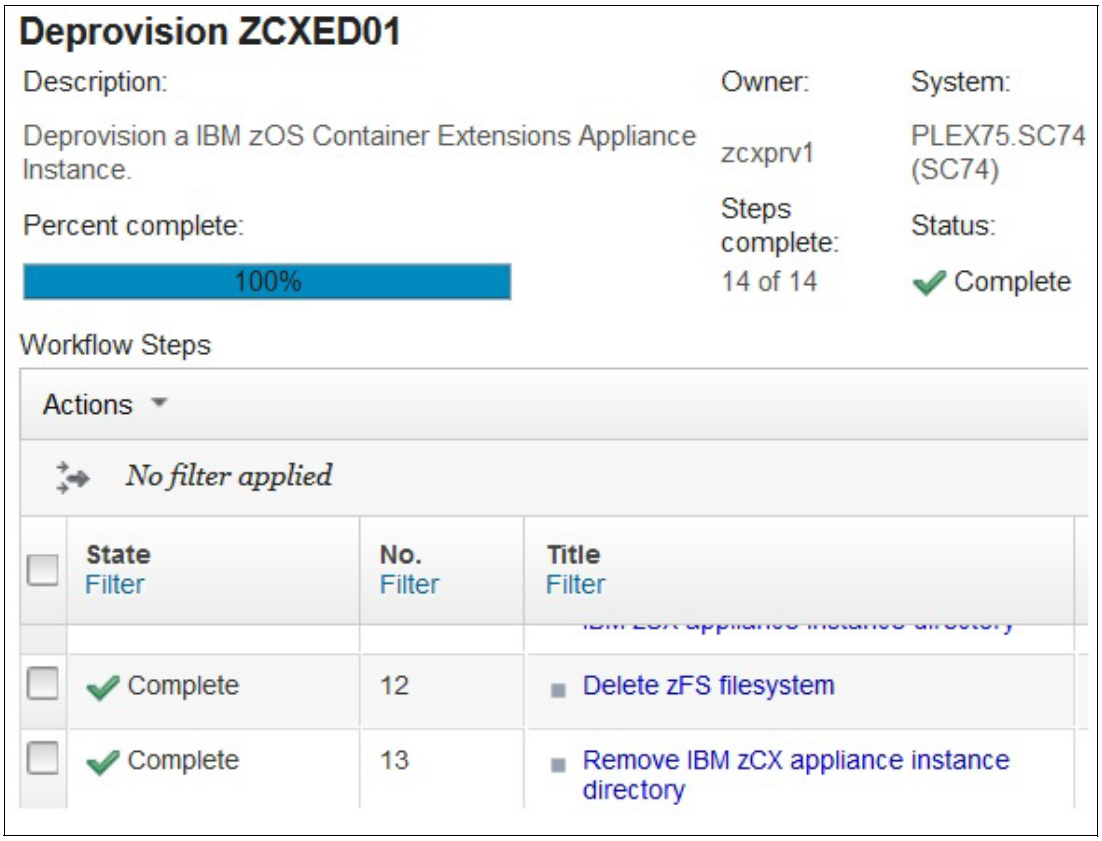


Figure 4-34 Completion of the deprovisioning workflow

4.13.3 Verify that the zCX instance has been deprovisioned

We confirmed that the zCX instance had been deprovisioned by verifying that the VSAM linear data sets, zFS, and instance directory were all deleted.

4.14 Other provisioning workflows

A number of other workflows are supplied for zCX. In Chapter 7., “Operation” on page 141, we explain the use of the upgrade and rollback workflows to apply and restore maintenance to the zCX instance. Also supplied are a backup and restore workflow to back up and restore the zCX instance configuration.

4.15 How to rerun a workflow

After you create a workflow and run it to completion, you might want to rerun that workflow. You have two options:

- Create a new workflow with a different name that is based on the workflow XML file that you want to run. OR
- Directly rerun a completed workflow, as described here:

1. In the Workflow display, click the name of your workflow to show all the steps.

2. Select all steps in the workflow, and then select **Assignment and Ownership > Return** in the **Actions** drop-down menu, as shown in Figure 4-35.

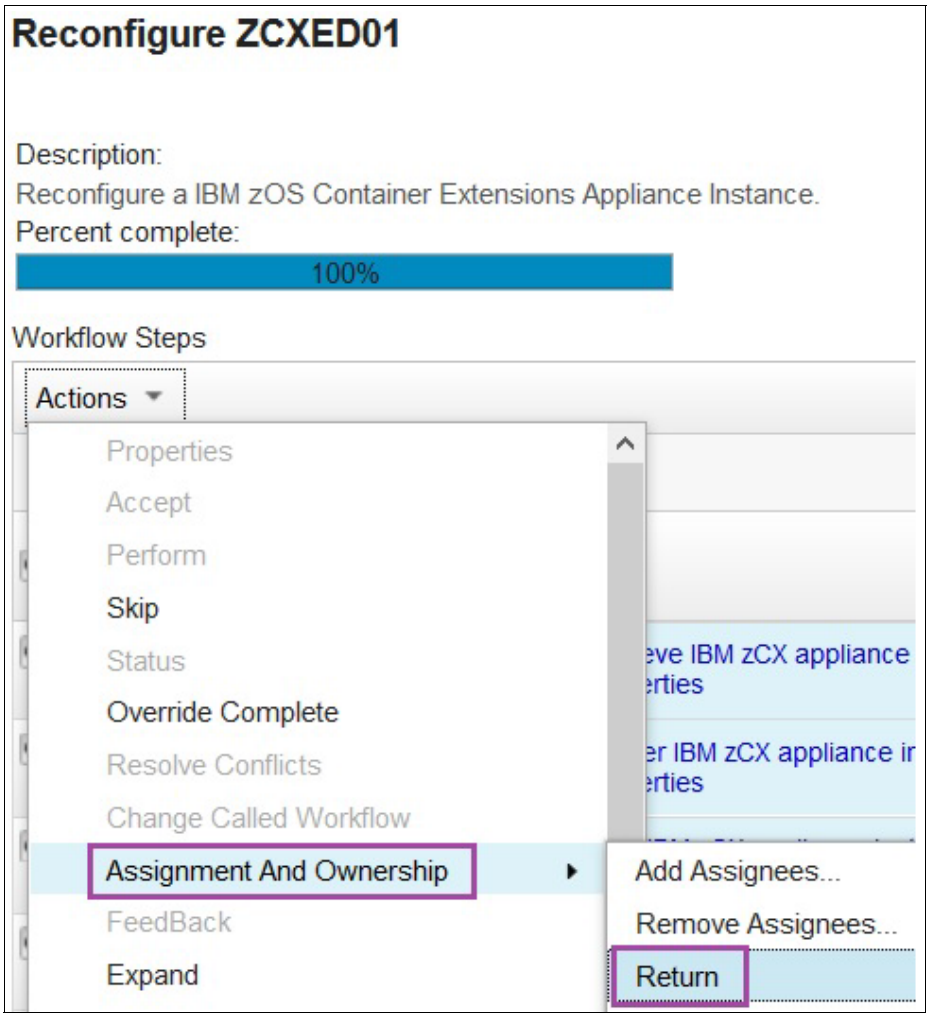


Figure 4-35 Selecting the Return action for all steps

3. Click **OK** in the confirmation window that is displayed.

   Figure 4-36 on page 94 shows how the workflow display is redisplayed with all the steps showing as **Assigned**.

Workflow Steps

Actions

No filter applied

State Filter	No. Filter	Title Filter
Assigned	1	Retrieve IBM zCX appliance instance properties
Assigned	2	Gather IBM zCX appliance instance properties
Assigned	3	Start IBM zCX appliance instance reconfiguration

Figure 4-36 Status of steps after doing Return action

4. Be sure that the steps are still all selected and select **Actions > Accept** in the drop-down menu.
5. Click **OK** in the **Accept step** window that is displayed.

The worklfow display is updated and the steps now show as **Ready**.

You are now able to start the workflow again by starting at the first step.

5

Your first running Docker container in zCX

This chapter describes how to get an image running in your zCX instance. The sections start with an easy example and continue to a more sophisticated level where you start and configure an HTTP server.

It provides the following sections:

- 5.1, “Overview of the process” on page 96
- 5.2, “Get an image from Docker Hub” on page 96
- 5.4, “Managing your Docker image” on page 104

© Copyright IBM Corp. 2019. All rights reserved.

5.1 Overview of the process

This section reviews the process for getting an image running in your z/OS installation. Detailed commands and descriptions can be found in the upcoming sections.

5.1.1 Provision your instance

To be able to start a Docker image, a running zCX instance is required. Detailed description on how to configure and start a zCX instance can be found in the previous Chapter 4, “Provisioning and managing your first z/OS Container” on page 47

5.1.2 Get a Docker image to run

Choose an available image on **hub.docker.com** matching the s390x architecture to download into your zCX instance. Use of the Docker pull command downloads the selected image to the local zCX directory.

5.1.3 Start your image

Start the downloaded image with the **`docker run`** command. To skip the previous step, you can directly issue the **`docker run`** command. If the image is not available locally, Docker automatically downloads the image from **hub.docker.com** and starts it, after the download is complete.

5.1.4 Access the provided service by the image

Depending on the service that is provided by the image, there are different ways to access them. For the chosen example with the HTTP server, the website can be access via the configured IP or DNS name from the zCX instance with the configured Port.

5.2 Get an image from Docker Hub

There are three different ways to get an image from Docker Hub into the running zCX instance. Both approaches are discussed in detail in this section. The simple way is to download it directly from Docker Hub but that requires a connection to the internet.

5.2.1 Download from Docker Hub

When the system your zCX instance is running does have access to the internet, you can directly download an image from **hub.docker.com**.

```
docker pull nginx
```

That command downloads the image into the zCX instance. Then, it is available to be started.

5.2.2 Download via a local Repository

The following command assumes that you have a registry running, called **myreg.company.com**. So, you can download an image through reference to that registry.

```
docker pull docker-hub.myreg.company.com/s390x/nginx
```

5.2.3 Load the image from a .tar file

If neither of the previous steps is possible, the third option would be to install Docker for Windows. If the prerequisites are fulfilled, follow the steps that are described here: https://docs.docker.com/docker-for-windows/install/

After it is installed and running, you can download the image the same way as in the zCX instance:
docker pull nginx

When the image is downloaded and available, the file must be exported in the tar format by executing the following command:
docker save s390x/nginx -o s390-nginx.tar

The generated file **s390-nginx.tar** is then available locally in the **C:\users\<*your_user*>** directory. That file must be uploaded in binary format to the zCX instance. Use your preferred file transfer manager to upload the file. You can upload the file into the **/tmp** directory from the zCX instance. After it is uploaded, the file must be extracted with this command:
docker -i /tmp/s390-nginx.tar

To check and verify the untar operation, display the available images with this command:
docker image ls

Example 5-1 shows typical output that lists the images.

Example 5-1 docker image ls

```
REPOSITORY              TAG     IMAGE ID       CREATED      SIZE
ibm_zcx_zos_cli_image   latest  a486d0c108ce   3 days ago   363M
nginx                   latest  a7b54eb09cb6   3 weeks ago  125MB
```

5.3 Run your Docker image on z/OS

This section describes how to get an image to run in your zCX instance. Two installations are described:

- A very simple “hello-world” use case.
- A more sophisticated use case with an HTTP server that is based on the nginx image.

Before you work with these use cases, the zCX instance must be running and you must have access to z/OS USS.

5.3.1 hello-world

The easiest way to start the first image in your zCX instance is to run the following command:
docker run hello-world

When you run the **docker run** command for the first time, it confirms the availability of the requested image: **hello-world**. If the image is available, Docker would start it right away. If the image is not available, Docker connects to **hub.docker.com** and downloads the image, and starts it immediately after successful completion of the download.

When the preceding command runs for the first time, the output on the Docker console looks similar to what is shown in Example 5-2.

Example 5-2 **docker run hello-world**

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
d39d9a31884d: Pull complete
Digest: sha256:451ce787d12369c5df2a32c85e5a03d52cbcef6eb3586dd03075f3034f10adcd
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/
```

You can split the **docker run** command into its two parts. To download the image first, you issue this command: **docker pull hello-world**

After the download is complete, and the image is available locally, issue the **docker run hello-world** command. Both approaches lead to the same result.

The previously described steps require either a direct connection to the internet to pull it from **hub.docker.com** or if the image is available in a local registry. When neither way is an option, the **hello-world** image must first be downloaded as described in 5.2.3, “Load the image from a .tar file” on page 97 where the file is imported from a local tar file.

5.3.2 HTTP Server (nginx)

This section goes more deeply into Docker and describes how to install and get an nginx-based HTTP server up and running. There might be some restrictions in publishing HTTP services to the outside. Firewall configuration changes might be necessary, depending on your local security policies.

Base installation

Get the nginx image in the appropriate way for your installation. The following process reviews the steps to take when you have direct access to the internet. To get the image, issue the **docker pull nginx** command, which generates the output that you see in Example 5-3.

Example 5-3 docker pull nginx

```
Using default tag: latest
latest: Pulling from library/nginx
d3a78a1911c6: Pull complete
f7c23a53129b: Pull complete
f1887ddbfe67: Pull complete
Digest: sha256:53ddb41e46de3d63376579acf46f9a41a8d7de33645db47a486de9769201fec9
Status: Downloaded newer image for nginx:latest
```

When the image is successfully downloaded, you start it by using the following command: **docker run -p 8080:80 -d nginx**

To confirm that the service is active and running, there are two steps to verify proper execution. Issue **docker ps** to list all running services.

Example 5-4 **docker ps**

```
CONTAINER ID     IMAGE                    COMMAND                  CREATED             STATUS
PORTS                            NAMES
9207d6711a33     nginx                    "nginx -g 'daemon of…"   8 seconds ago       Up 7 seconds
0.0.0.0:8080->80/tcp             quizzical_leavitt
1edc6a418cd8     ibm_zcx_zos_cli_image    "sudo /usr/sbin/sshd…"   3 days ago          Up 2 hours
8022/tcp, 0.0.0.0:8022->22/tcp   ibm_zcx_zos_cli
```

This output from Example 5-4 shows us that our nginx instance got the name **quizzical_leavitt** and is listening on port 8080. Then, the port 8080 is internally linked to port 80.

To access the content of the web server, you need to know the IP address of the zCX instance or the registered hostname. If the hostname is not known, you can issue the command **docker info** and look for the **name**: line, where the hostname is displayed, as shown in bold text in Example 5-5.

Example 5-5 ***docker info*** *output truncated to show only relevant information for this situation*

```
[.....]
Architecture: s390x
CPUs: 1
Total Memory: 1.675GiB
Name: sc74cn12.pbm.ihost.com
ID: SSDL:E72H:GSF4:MNUD:ZHK3:CO6D:3YKR:JPTP:3TS5:TRW7:FA5C:3SL5
[......]
```

By combining the information about the hostname and the defined port, the website can be accessed through any browser as shown in Example 5-1 on page 100.

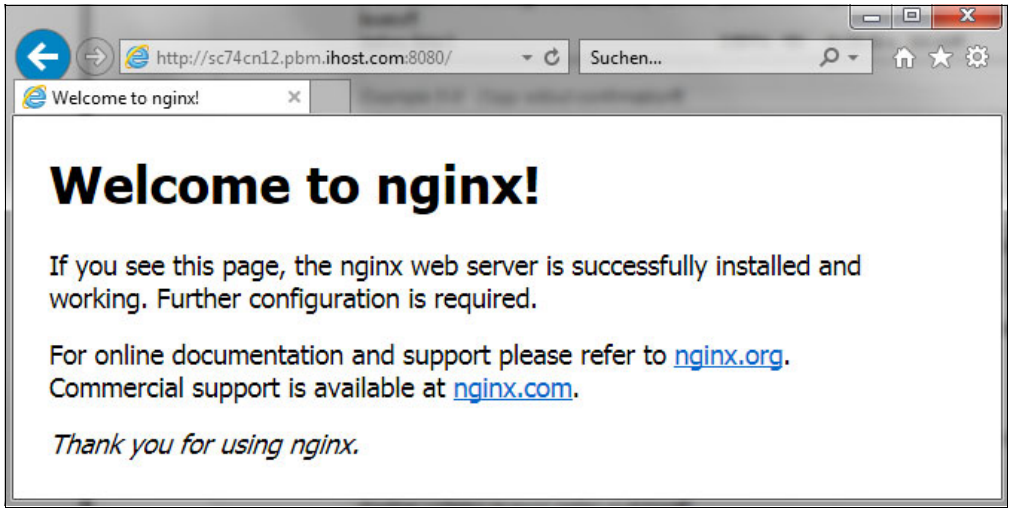


Figure 5-1 Default output from the nginx web server

With this verification, the base installation of the nginx web server is complete. The upcoming section goes into more detail about configuration of the web service with custom content.

Configure the base installation

The previous section describes how to pull the image, so that description is not repeated in this section. This section focuses on creation of content for publication in the web server.

Preparation

First, we create the content that we want to publish in our web server. Following example shows an index.html page. Feel free to adjust the text that is inside the **<body>** tag set to any content that you like.

Example 5-6 index.html

```
<html>
<body>
This is my first website in a zCX Docker image.
 </body>
</html>
```

This file can either be created in the zCX instance or as a z/OS USS file, as described here:

File creation in the zCX instance:

1. Within the Docker CLI, run the following command to access the **admin** home directory:
   `cd /home/admin`
2. Run the following command to open an editor session with vi:
   `vi index.html`
3. Click the `i` button to enter **insert** mode, and paste the content from Example 5-6.
4. Press `ESC` to escape from the insert mode, and then type `:wq` to write and close the file.
5. Run the `ls -l` command to confirm the content of your updated **index.html** file. Example 5-7 shows typical output.

Example 5-7 ls -l

```
-rw-rw-r-- 1 admin admin 83 Sep  9 15:32 index.html
```

File creation in the z/OS USS:
You can use the ISPF interface or log on through an SSH session.

- In an SSH session, just follow the same steps that you took to create the file locally in the zCX instance. Using the ISPF interface, you must navigate to **`z/OS UNIX Directory List Utility`** (issuing 3.17 as line command).
- In the z/OS USS environment, you must copy the file that you create into the zCX instance. A simple way to achieve that goal would be to use **scp** (secure copy) from an SSH session, as shown by Example 5-8.

Here is an example **scp** command:

```
scp -P 8022 /u/zcxadm6/index.html admin@129.40.23.79:/home/admin/index.html
```

If you are connecting to the zCX instance for the first time, you must enter **yes** at the prompt to confirm the fingerprint, as shown by the **bold** text in Example 5-8:

If you connected before and already saved the fingerprint, the response looks similar to what you see in Example 5-9.

Example 5-8 Copy with confirmation

```
The authenticity of host '[129.40.23.78]:8022 ([129.40.23.78]:8022)' can't be established.
ECDSA key fingerprint is SHA256:IA5Sw4oNK7wODhFGkH4ayyJNvT73m1uvQprHjqOC41M.
Are you sure you want to continue connecting (yes/no)? yes
FOTS2274 Warning: Permanently added '[129.40.23.78]:8022' (ECDSA) to list of known hosts.
index.html                                100%   90     0.1KB/s   00:00
```

Example 5-9 Copy withouth confirmation

```
index.html                                100%   90     0.1KB/s   00:00
```

After the successful copy, you can log off from z/OS USS and log on to the zCX instance.

The generated file can now be copied into the running nginx instance. To do so, issue the following **copy** command.

```
docker cp /home/admin/index.html quizzical_leavitt:/usr/share/nginx/html
```

This command copies the **`index.html`** file from the local directory into the **`quizzical_leavitt`** **containers** directory **`/usr/share/nginx/html`**.

When the **copy** command is complete, you can refresh the website in a browser. The new content looks similar to what you see in Figure 5-2.

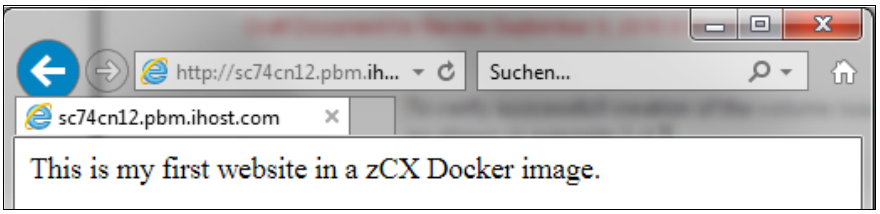


Figure 5-2 Personal output from the nginx web server

Further information: With the steps that were performed to this point, an nginx web server is running with its dedicated content. The copied **index.html** file is not accessible from other containers. That means that if the instance **quizzical_leavitt** is removed, the configured **index.html** page also vanishes. The solution for that situation is described in the next section,

In this section, we installed, ran, and configured a web server in a basic way. The upcoming section presents a similar process in a more sophistcated way that guarantees data sharing and persistence.

5.3.3 Enhanced installation

In the next step, the web server could be started with the following default command from Docker Hub:

```
docker run --name some-nginx -v /some/content:/usr/share/nginx/html:ro -d nginx
```

However, this command is not valid in the zCX environment, due to the security policy. Specifically, the preceding command has the following mapping:

- From **/usr/share/nginx/html** (within the nginx instance).
- To the local folder **/some/content** (within the zCX instance).

For this reason, the command must be changed to adhere the zCX environment restrictions. To achieve that, you must create a Docker volume. A volume can be reused by various instances, so the volume enables a form of data persistence. For example, the user data is still available if a running instance is deleted, because the data is stored in the Docker volume. Further instances can started and can use the same data source from that Docker volume.

Important Note: Due to security restrictions in zCX, it is not possible to link the zCX local folders to the container. You must create a Docker volume first. To copy files into a volume, the volume must be mounted to a running instance before files can be copied into it.

To create a Docker volume, issue the following command, where *nginx-webdata* is a unique text name that you choose.

```
docker volume create nginx-webdata
```

To confirm the successful creation of the volume, issue the following command and verify the output as shown in Example 5-10.

```
docker volume inspect nginx-webdata
```

Example 5-10 Personal output from the nginx web server on a new instance

```
[
    {
        "CreatedAt": "2019-09-09T19:29:07Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/media/data/docker/24000.109/volumes/nginx-webdata/_data",
        "Name": "nginx-webdata",
        "Options": {},
        "Scope": "local"
    }
]
```

While you are still logged in to the zCX instance, you can run the following command to start the web server.

```
docker run --name nginx-HTTP-Server -v nginx-webdata:/usr/share/nginx/html:rw -p 8081:80 -d
nginx
```

The name ***nginx-HTTP-Server*** is a unique text name that you can choose. If more than one instance of a webservice runs in the zCX instance, you must set a different name and a unique port.

When the **docker run** command was successful, the instance can be displayed with the following command:
docker container ps
The output as shown in Example 5-11 on page 103 shows that the **nginx-HTTP-Server** server is running and listening on port 8081. The instance from the base installation is still running and listening on port 8080.

Example 5-11 Output from docker ps command

```
CONTAINER ID    IMAGE                   COMMAND                  CREATED          STATUS          PORTS
NAMES
ecaaaddcef02    nginx                   "nginx -g 'daemon of…"   13 seconds ago   Up 13 seconds   0.0.0.0:8081->80/tcp
nginx-HTTP-Server
9207d6711a33    nginx                   "nginx -g 'daemon of…"   5 hours ago      Up 5 hours      0.0.0.0:8080->80/tcp
quizzical_leavitt
1edc6a418cd8    ibm_zcx_zos_cli_image   "sudo /usr/sbin/sshd…"   3 days ago       Up 7 hours      8022/tcp, 0.0.0.0:8022->22/tcp
ibm_zcx_zos_cli
```

The new web server, named **nginx-HTTP-Server** is using the previously created volume. You must copy the **index.html** file into that directory, too, by running the following command:

```
docker cp /home/admin/index.html nginx-HTTP-Server:/usr/share/nginx/html
```

Because of the mapping that is done during the starting of the instance, the copy of **index.html** file is stored in the **nginx-webdata** volume. If another nginx instance is started and it uses the same Docker volume mapping, that instance shows exactly the same content. The following example command starts another instance that listens on port 8082.

```
docker run --name nginx-HTTP-Server1 -v nginx-webdata:/usr/share/nginx/html:rw -p
8082:80 -d nginx
```

If no further **copy** command is run, the website continues to show the web content from the **index.html** file that was copied in a previous step. The command in Example 5-11 can be used at this point for reference and to check the used port **8082**.

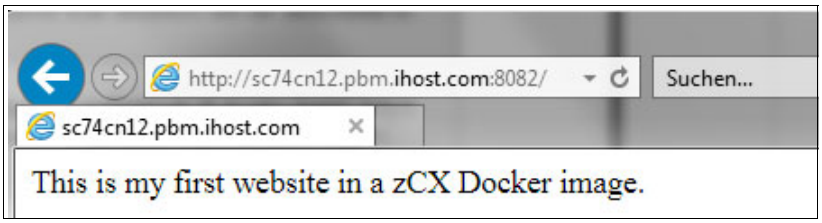


Figure 5-3 Personal content from another instance that is sharing the same volume

Now, you can start to manage the data in the Docker volume, and the content is reflected in all instances that were started with the same volume mapping.

5.4 Managing your Docker image

Many commands are available to manage a Docker image. The following list is not comprehensive, but lists some that are useful for your first interactions with Docker. All available commands can be found here:
https://docs.docker.com/engine/reference/commandline/cli/

5.4.1 Start

This section lists basic start commands.

docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
This command starts a new container from an available image.

Example 5-12 Example command to run a Docker image

```
docker run --name nginx-HTTP-Server -v nginx-webdata:/usr/share/nginx/html:rw -p
8080:80 -d nginx
```

docker unpause [OPTIONS] CONTAINER [CONTAINER]
This command starts one or more containers that are paused.

Example 5-13 Example command to unpause a paused container

```
docker unpause nginx-HTTP-Server
```

docker restart [OPTIONS] CONTAINER [CONTAINER]
This command restarts one or more running containers.

Example 5-14 Example command to restart a running container

```
docker restart nginx-HTTP-Server
```

5.4.2 Display

docker ps [OPTIONS]

This command lists all running containers. You add the option **-a** to list all containers.

Example 5-15 Output from issued **docker ps -a** *command*

```
CONTAINER ID     IMAGE              COMMAND                  CREATED         STATUS                    PORTS
NAMES
8dc7e93c35ec     nginx              "nginx -g 'daemon of…"   18 hours ago    Up 18 hours               0.0.0.0:8082->80/tcp
nginx-HTTP-Server1
ecaaaddcef02     nginx              "nginx -g 'daemon of…"   18 hours ago    Up 5 minutes (Paused)     0.0.0.0:8081->80/tcp
nginx-HTTP-Server
9207d6711a33     nginx              "nginx -g 'daemon of…"   23 hours ago    Up 23 hours               0.0.0.0:8080->80/tcp
quizzical_leavitt
9661b7d5062d     hello-world        "/hello"                 24 hours ago    Exited (0) 24 hours ago
gallant_rubin
1edc6a418cd8     ibm_zcx_zos_cli_image   "sudo /usr/sbin/sshd…"   4 days ago      Up 25 hours          8022/tcp, 0.0.0.0:8022->22/tcp
ibm_zcx_zos_cli
```

The output confirms that the **nging-HTTP-Server** server (highlighted in **bold**) is in **paused** status.

5.4.3 Stop

docker stop [OPTIONS] CONTAINER [CONTAINERS]

This command stops one or more running containers.

Example 5-16 Example command to stop a running container

```
docker stop nginx-HTTP-Server
```

docker pause CONTAINER [CONTAINER]

This command pauses one or more running containers.

Example 5-17 Example command to pause a running container

```
docker pause nginx-HTTP-Server
```

5.4.4 Terminate

docker rm [OPTIONS] CONTAINER [CONTAINER]

This command removes one or more (running) containers. If used with the **-f** parameter, even a running container is removed. When you add the **-v** parameter, any associated volumes are also removed.

Example 5-18 Example to remove a running container

```
docker rm nginx-HTTP-Server -f
```

6

Private registry implementation

One of the main components in a container environment is a Docker registry. For security and administrative reasons, it is good practice to build a private registry before you proceed with Docker implementation.

Developers must store and retrieve images in a registry when they deploy their applications. It is important to note that Docker maintains a public registry at Docker Hub. Other organizations including IBM and Redhat provide registries for different collections of images, too.

This chapter is for those who have the following goals:

- Build a private registry to share Docker images with users.
- Control what images can be deployed in the environment.
- Provide an alternative for zCX instances that run behind firewalls.
- Increase security where approved and certificated images can be deployed.

Although starting a registry is not complex, you must understand some concepts for pushing and pulling images. The concepts help you to build your Private Registry service.

This chapter provides important information about installation, including basic scenarios, prerequisites, and links to other reference sources.

This chapter includes the following topics:

- 6.1, “Private registry overview” on page 108
- 6.2, “Tag discussion for images” on page 109
- 6.3, “Building a private registry image” on page 112
- 6.4, “Running a local private registry” on page 117
- 6.5, “Deploying a secure private registry” on page 118
- 6.6, “Creating TLS certificates on Windows or Macintosh” on page 130
- 6.7, “Working with tags” on page 134
- 6.8, “Deleting an image from a private Docker registry” on page 137
- 6.9, “Using the private registry to create containers” on page 139

© Copyright IBM Corp. 2019. All rights reserved.

6.1 Private registry overview

A Docker Registry is a storage and distribution system for Docker images. You can store public images for free and share them with other users.

Users must access a registry to load images onto a zCX instance. It's important to note that zCX supports both secure and insecure registries.

The majority of users today use Docker Hub as their external registry for finding and sharing container images. This public registry provides thousands of images for different platforms (x86, Power, and IBM Z). The same image might have multiple different versions, as indicated by their tags.

The zCX appliance can support only base images in Docker Hub that are tagged as "s390x".

> **Note:** There is an IBM namespace called **ibmcom** that contains images for IBM products. The repository is found here: `https://registry.hub.docker.com/repos/ibmcom/`

The following list highlights key reasons to build a private registry:

- **Security:** Use Docker images privately.
- **Network availability:** zCX instances can pull/push images even if they do not have access to the internet.
- **Performance:** Images can be quickly uploaded and downloaded.
- **Control:** Specify the image that local users are able to download.

By default, the Docker Engine interacts with Docker Hub, which is Docker's public registry instance. However, it is possible to reconfigure the zCX instances to access a private Docker registry. This approach increases security and reduces the exposure of having several instances that access the internet to pull/push images from Docker Hub.

> **Note:** If your private registry instance is behind a firewall and internal Docker clients must access it, ensure that you open the 5000 TCP port in your firewall to allow communication between Docker clients and the registry.

Figure 6-1 shows the use of a private registry in the zCX environment.

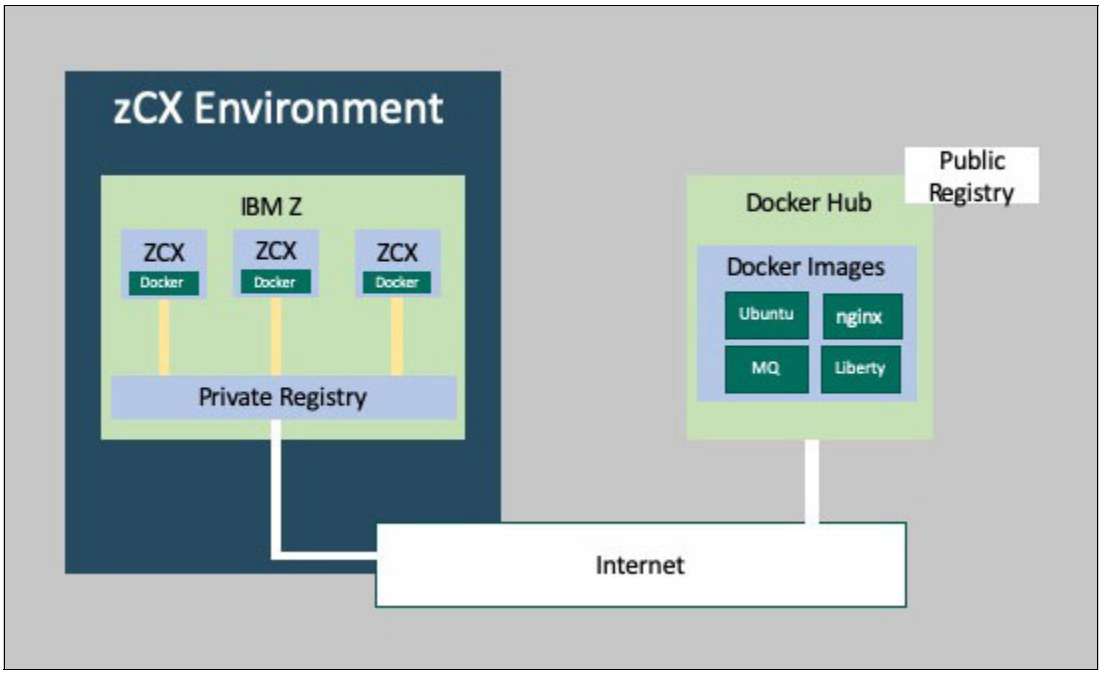


Figure 6-1 zCX Environment using a private registry

6.2 Tag discussion for images

The concept of tagging an image is important for anyone who wants to use Docker client to pull and push images from private or public registries. The **`docker tag`** command tags images and ensures that users know exactly which image they are getting from the repository.

When working with images, ensure that each image has a tag so that users can easily identify the version of the application and its purposes (for example, production or development) that they are looking for. This is especially useful when you create new images, because it helps to maintain control of your application deployments.

Docker Hub is the biggest public registry with thousands of images. Each has its own identification (tags). Figure 6-2 provides an example that shows the available tags for a PostgreSQL database image. For more information, see `https://hub.docker.com/_/postgres`

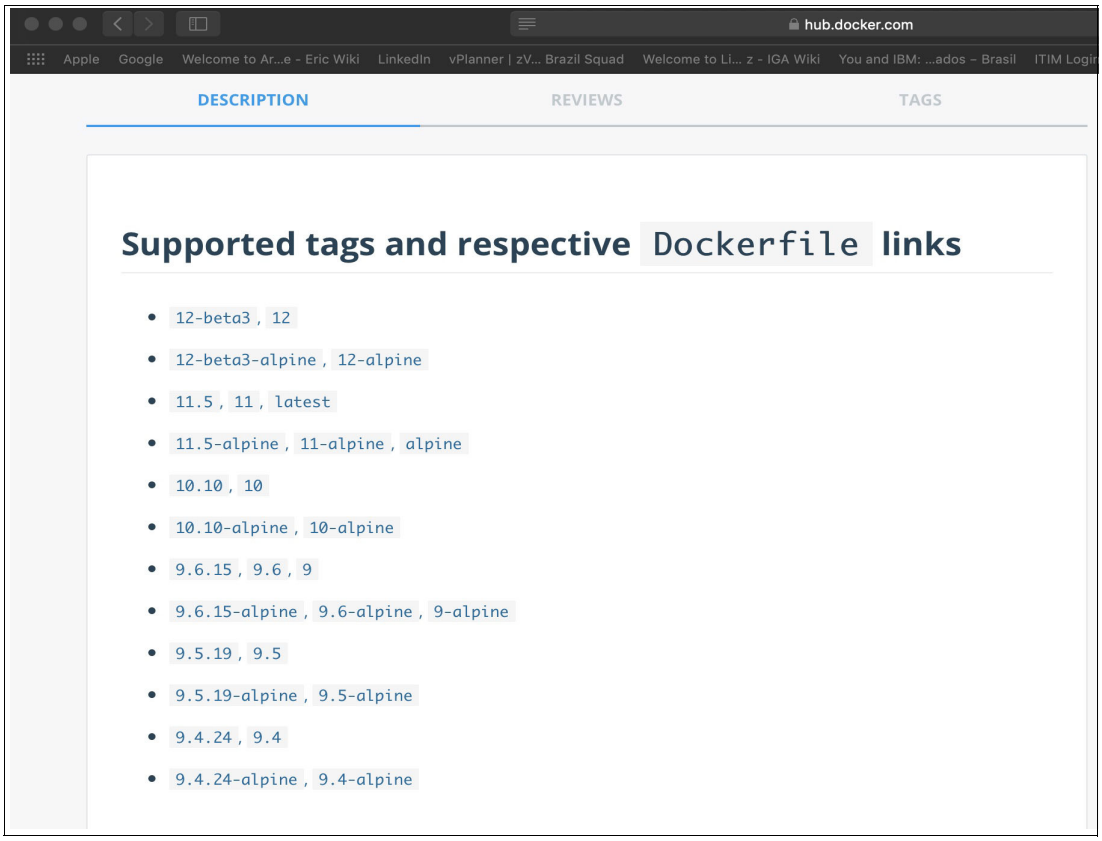


Figure 6-2 Available tags for postgres database

If you want to list the first 15 tags for the **postgres** image in Docker Hub by using a command, issue this one:

```
curl -s -k https://registry.hub.docker.com/v1/repositories/postgres/tags |
sed -e 's/[][]//g' -e 's/"//g' -e 's/ //g' | tr ',' "\n" |tr -d '}'|
grep name | awk -F\: '{print $2}' |head -n 15
```

Output for this curl command can be found in Example 6-1.

Example 6-1 Output of curl command to list the first 15 tags

```
latest
10
10-alpine
10-beta1
10-beta1-alpine
10-beta2
10-beta2-alpine
10-beta3
10-beta3-alpine
10-beta4
10-beta4-alpine
10-rc1
10-rc1-alpine
10.0
10.0-alpine
```

There are a few ways to control version for your Docker images. Tagging is one of the methods that add version control to Docker images. When you use commands to pull and push, you can specify the specific version that you want to download. Users usually use the commands that are listed in Table 6-1 to pull and push images from registry.

Table 6-1 Docker pull and push

Command	Description
docker pull <*image*>:<*tag*>	To pull an image from Docker repository
docker push <*image*>:<*tag*>	To push an image to Docker repository

Docker users can easily use tags to deploy or roll out versions for their applications at a command. If for some reason the application is not behaving well after a version upgrade, the old version can be rolled back and the application can be up in just a few minutes. That is an important reason that container technology is gaining popularity and is being adopted by more and more organizations.

Many images have a tag called **latest**, which is applied to an image by default. It means that if you pull or push an image without specify the tag, the Docker client tries to get the image by using latest tag. Be aware that using **latest** as tag has some risks. For example, if another Docker user pushes a new image to the repository and forgets to specify a tag. It overwrites the latest image in your repository.

Be cautious when you pull images into production environments to deploy applications. The latest tag might not be always the best choice. In some cases, it can bring problems.

As a best practice, complete validation and testing of new software versions is recommended in the development environment, prior to deployment to production. As mentioned previously, you have power to control which version is going to be installed and ensure that your production environment has the correct image code.

Although Docker container allows a quick image rollback, you might not want to break the production environment even for just a few minutes. So, ensure you use tags when pulling images into your production environment. Also, educate other Docker users about tagging to prevent outages and problems.

You can tag an image by using this command:

```
docker tag <image_id> image:TAG
```

Sample:

```
docker tag ubuntu:latest ubuntu:16.04
```

The following example retags the **ubuntu:latest** image as **ubuntu:16.04**

1. Use the **`docker images`** command to list the available Docker images locally.

Example 6-2 Listing available Docker Images

```
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
myregistry              latest              082be17251ee        30 hours ago        441MB
ibm_zcx_zos_cli_image   latest              00f61e376fd0        3 days ago          363MB
ubuntu                  latest              f8835575bd80        3 weeks ago         64.8MB
s390x/ubuntu            16.04               fcae39a6d474        6 weeks ago         118MB
```

2. Issue the following command to retag the Ubuntu image:

   docker tag ubuntu:latest ubuntu:16.04

3. Use the **docker images** command to confirm that a tag was added to the Ubuntu image.

Example 6-3 Listing Available Docker Images

```
admin@fcc1a71d8f61:~$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
myregistry              latest              082be17251ee        30 hours ago        441MB
ibm_zcx_zos_cli_image   latest              00f61e376fd0        3 days ago          363MB
ubuntu                  16.04               f8835575bd80        3 weeks ago         64.8MB
ubuntu                  latest              f8835575bd80        3 weeks ago         64.8MB
s390x/ubuntu            16.04               fcae39a6d474        6 weeks ago         118MB
```

4. Now, remove the tag for **ubuntu:latest** by running the following command:

   docker rmi ubuntu:latest

You should receive a message "*Untagged: ubuntu:latest*".

For more details in how to work with tags, see Section 6.7, "Working with tags" on page 134 in this publication.

6.3 Building a private registry image

There is a registry image available in Docker Hub for IBM Z platform. The name of this image is *ibmcom/registry-s390x* and its available in Docker Hub. For more details, see https://hub.docker.com/r/ibmcom/registry-s390x.

For our private registry deployment, we use *ibmcom/registry-s390x*, and the building of an image is not required. However, this section provides good concepts and commands that might be helpful for application developers who manage Docker instances. This procedure can also be used if you do not find a registry image with the version that you want to install.

The following steps work only when the zCX instance has internet access. The image build process downloads and installs some components into the new image when it executes the instructions of the Dockerfile. You see more details in Example 6-8, which shows the output of the build process.

Use the procedure in this section to build a private registry image. Again, we recommend that you use the available registry image for **s390x** (**ibmcom/registry-s390x**). If your zCX instance is behind a firewall, go to the following section.

1. Connect via SSH into your zCX instance by using the command in Example 6-4. Be sure to replace the IP address with one that is valid for your zCX environment.

Example 6-4 Connecting to zCX instance

```
erics-mbp:.ssh ibmuser$ ssh admin@129.40.23.70 -p 8022
Warning: Permanently added '[129.40.23.70]:8022' (ECDSA) to the list of known hosts.

Welcome to the IBM z/OS Container Extensions (IBM zCX) shell that provides access to Docker commands.
For more information on how to use this shell to execute Docker commands refer to "IBM z/OS Container
Extensions Guide".
Last login: Fri Sep  6 15:22:36 2019 from 9.57.181.31
```

2. Create a folder for your Dockerfile:

   mkdir -p /home/admin/myregistry
   cd /home/admin/myregistry

3. Download the Dockerfile file from GitHub. This Dockerfile contains instructions for building a registry. Example 6-5 shows typical output of the command.

   curl -o Dockerfile https://raw.githubusercontent.com/linux-on-ibm-z/dockerfile-examples/master/DockerDistribution/Dockerfile -k

Example 6-5 Pulling Docker registry image

```
admin@fcc1a71d8f61:~/myregistry$ curl -o Dockerfile
https://raw.githubusercontent.com/linux-on-ibm-z/dockerfile-examples/master/Docker
Distribution/Dockerfile -k
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2490  100  2490    0     0  99600      0 --:--:-- --:--:-- --:--:-- 99600$
```

4. Confirm that the file exists. Example 6-6 shows typical output of the command.

   ls -la

Example 6-6 Output of docker run command

```
admin@fcc1a71d8f61:~/myregistry$ ls -la Dockerfile
-rw-rw-r-- 1 admin admin 2490 Sep  7 15:46 Dockerfile$
```

5. To see the instructions that are listed in Dockerfile file, run the **cat Dockerfile** command as shown in the following example.

Example 6-7 Output of cat Dockerfile command

```
admin@570e9473805e:~/myregistry$ cat Dockerfile
# © Copyright IBM Corporation 2017, 2019.
# LICENSE: Apache License, Version 2.0 (http://www.apache.org/licenses/LICENSE-2.0)

############## Dockerfile for Docker Distribution version v2.7.1 ################
#
# This Dockerfile builds a basic installation of Docker Distribution.
#
# Docker Distribution is the Docker Registry 2.0 implementation for storing and
distributing Docker images.
#
# To build this image, from the directory containing this Dockerfile
# (assuming that the file is named Dockerfile):
# docker build -t <image_name> .
#
# To start Docker Distribution, create and start a container from above image as follows:
# docker run --name <container_name> -p 5000:5000 -d <image_name>
#
# To start Docker Distribution using sample_config.yml file using below command:
# docker run --name <container_name> -v
<path_on_host>/sample_config.yml:/etc/docker/registry/config.yml -p 5000:5000  -d
<image_name>
#
################################################################################

# Base Image
FROM s390x/ubuntu:16.04
```

```
ARG DOCKER_DISTRIBUTION_VER=2.7.1

# The author
MAINTAINER  LoZ Open Source Ecosystem
(https://www.ibm.com/developerworks/community/groups/community/lozopensource)

ENV GOPATH /go
ENV DISTRIBUTION_DIR /go/src/github.com/docker/distribution
ENV PATH=$PATH:/usr/share:/usr/local/go/bin

# Install dependencies
RUN apt-get update && apt-get install -y \
    git \
    make \
   tar \
   wget \
# Install go
 && cd /root \
 && wget https://storage.googleapis.com/golang/go1.11.4.linux-s390x.tar.gz \
 && chmod ugo+r go1.11.4.linux-s390x.tar.gz \
 && tar -C /usr/local -xzf go1.11.4.linux-s390x.tar.gz \
# Download and build source code of Docker Distribution
 && git clone https://github.com/docker/distribution.git
$GOPATH/src/github.com/docker/distribution \
 && cd $GOPATH/src/github.com/docker/distribution && git checkout
v${DOCKER_DISTRIBUTION_VER} \
 && make clean binaries \
 && mkdir -p /etc/docker/registry \
 && cp $DISTRIBUTION_DIR/cmd/registry/config-dev.yml /etc/docker/registry/config.yml \
 && cp -r $DISTRIBUTION_DIR/bin/registry /usr/share/registry \
# Tidy up (Clear cache data)
 && apt-get remove -y \
    git \
   make \
   wget \
 && apt-get autoremove -y \
 && apt-get clean \
 && rm /root/go1.11.4.linux-s390x.tar.gz \
 && rm -rf $DISTRIBUTION_DIR \
 && rm -rf /var/lib/apt/lists/*

VOLUME ["/var/lib/registry"]

EXPOSE 5000

ENTRYPOINT ["registry"]

CMD ["serve", "/etc/docker/registry/config.yml"]
# End of Dockerfile
```

6. Run the following command to make the container image. This command builds the image from the Dockerfile in the directory.

   docker build -t myregistry .

When the build is complete you see the **Successfully tagged myregistry:latest** message. Typical output is shown in Example 6-8 on page 115.

Example 6-8 Output of docker build command

```
admin@fcc1a71d8f61:~/myregistry$ docker build -t myregistry .
Sending build context to Docker daemon  4.096kB
Step 1/11 : FROM s390x/ubuntu:16.04
16.04: Pulling from s390x/ubuntu
45848351f60c: Pull complete
9d06bd1d66a7: Pull complete
46992b2c26a8: Pull complete
cee1faa53d30: Pull complete
Digest: sha256:9a3bdec139f4260688577102a1601dd75ade257aeea9db3d9500218ff1f43c90
Status: Downloaded newer image for s390x/ubuntu:16.04
 ---> fcae39a6d474
Step 2/11 : ARG DOCKER_DISTRIBUTION_VER=2.7.1
 ---> Running in 08cfc3273205
Removing intermediate container 08cfc3273205
 ---> 89eb101afacf
Step 3/11 : MAINTAINER  LoZ Open Source Ecosystem
(https://www.ibm.com/developerworks/community/groups/community/lozopensource)
 ---> Running in 4e73b435e7b3
Removing intermediate container 4e73b435e7b3
 ---> cd239c395e53
Step 4/11 : ENV GOPATH /go
 ---> Running in 3d6dab03fddd
Removing intermediate container 3d6dab03fddd
 ---> d14d8ca971f5
Step 5/11 : ENV DISTRIBUTION_DIR /go/src/github.com/docker/distribution
 ---> Running in a951813db6c8
Removing intermediate container a951813db6c8
 ---> aaabd95c867e
Step 6/11 : ENV PATH=$PATH:/usr/share:/usr/local/go/bin
 ---> Running in c0b7522b7e50
Removing intermediate container c0b7522b7e50
 ---> a58a6450d167
Step 7/11 : RUN apt-get update && apt-get install -y     git     make tar wget  && cd /root  && wget
https://storage.googleapis.com/golang/go1.11.4.linux-s390x.tar.gz  && chmod ugo+r
go1.11.4.linux-s390x.tar.gz  && tar -C /usr/local -xzf go1.11.4.linux-s390x.tar.gz  && git clone
https://github.com/docker/distribution.git $GOPATH/src/github.com/docker/distribution  && cd
$GOPATH/src/github.com/docker/distribution && git checkout v${DOCKER_DISTRIBUTION_VER}  && make clean
binaries  && mkdir -p /etc/docker/registry  && cp $DISTRIBUTION_DIR/cmd/registry/config-dev.yml
/etc/docker/registry/config.yml  && cp -r $DISTRIBUTION_DIR/bin/registry /usr/share/registry  && apt-get
remove -y     git make wget  && apt-get autoremove -y  && apt-get clean  && rm
/root/go1.11.4.linux-s390x.tar.gz  && rm -rf $DISTRIBUTION_DIR  && rm -rf /var/lib/apt/lists/*
 ---> Running in 028f2b75faab
Get:1 http://ports.ubuntu.com/ubuntu-ports xenial InRelease [247 kB]
Get:2 http://ports.ubuntu.com/ubuntu-ports xenial-updates InRelease [109 kB]
Get:3 http://ports.ubuntu.com/ubuntu-ports xenial-backports InRelease [107 kB]
Get:4 http://ports.ubuntu.com/ubuntu-ports xenial-security InRelease [109 kB]
Get:5 http://ports.ubuntu.com/ubuntu-ports xenial/main s390x Packages [1467 kB]
Get:6 http://ports.ubuntu.com/ubuntu-ports xenial/universe s390x Packages [9381 kB]
Get:7 http://ports.ubuntu.com/ubuntu-ports xenial/multiverse s390x Packages [145 kB]
Get:8 http://ports.ubuntu.com/ubuntu-ports xenial-updates/main s390x Packages [906 kB]
Get:9 http://ports.ubuntu.com/ubuntu-ports xenial-updates/universe s390x Packages [806 kB]
Get:10 http://ports.ubuntu.com/ubuntu-ports xenial-updates/multiverse s390x Packages [12.0 kB]
Get:11 http://ports.ubuntu.com/ubuntu-ports xenial-backports/main s390x Packages [7933 B]
Get:12 http://ports.ubuntu.com/ubuntu-ports xenial-backports/universe s390x Packages [7381 B]
Get:13 http://ports.ubuntu.com/ubuntu-ports xenial-security/main s390x Packages [612 kB]
Get:14 http://ports.ubuntu.com/ubuntu-ports xenial-security/universe s390x Packages [454 kB]
Get:15 http://ports.ubuntu.com/ubuntu-ports xenial-security/multiverse s390x Packages [1698 B]
Fetched 14.4 MB in 2s (5856 kB/s)
Reading package lists...
```

```
Reading package lists...
Building dependency tree...
Reading state information...
tar is already the newest version (1.28-2.1ubuntu0.1).
The following additional packages will be installed:
  ca-certificates git-man ifupdown iproute2 isc-dhcp-client isc-dhcp-common
  krb5-locales less libasn1-8-heimdal libatm1 libbsd0 libcurl3-gnutls
  libdns-export162 libedit2 liberror-perl libexpat1 libffi6 libgdbm3 libgmp10

...Snippet Output....

Removing libxcb1:s390x (1.11.1-1ubuntu1) ...
Removing libxau6:s390x (1:1.0.8-1) ...
Removing libxdmcp6:s390x (1:1.1.2-1.1) ...
Processing triggers for libc-bin (2.23-0ubuntu11) ...
Removing intermediate container 028f2b75faab
 ---> 603885dff855
Step 8/11 : VOLUME ["/var/lib/registry"]
 ---> Running in db6c5e37753a
Removing intermediate container db6c5e37753a
 ---> 9a024309e044
Step 9/11 : EXPOSE 5000
 ---> Running in e01278e2a6a9
Removing intermediate container e01278e2a6a9
 ---> 18abf2b64922
Step 10/11 : ENTRYPOINT ["registry"]
 ---> Running in 7b8f07664b09
Removing intermediate container 7b8f07664b09
 ---> 25ad496b77d1
Step 11/11 : CMD ["serve", "/etc/docker/registry/config.yml"]
 ---> Running in 6ee0b0742744
Removing intermediate container 6ee0b0742744
 ---> 296d69b73f2a
Successfully built 296d69b73f2a
Successfully tagged myregistry:latest
```

7. Run the following command to confirm that the **myregistry** Docker image is available: **docker images**

Example 6-9 shows output that shows successful creation of the **myregistry** image.

Example 6-9 Output of docker images command

```
admin@fcc1a71d8f61:~/myregistry$ docker images
REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
myregistry              latest              082be17251ee        4 seconds ago       441MB
ibm_zcx_zos_cli_image   latest              00f61e376fd0        47 hours ago        363MB
ubuntu                  latest              f8835575bd80        3 weeks ago         64.8MB
s390x/ubuntu            16.04               fcae39a6d474        6 weeks ago         118MB
```

Optionally, you can save **myregistry** image to a tar file and load it in any other zCX instance. Although the steps to build an image are not complex, you can skip this work by saving and loading the image in any other zCX instance. Remember to use **scp** Linux command to send this file remotely, and make a backup copy of it. See the details in Example 6-10.

Example 6-10 Saving the myregistry image

```
# Saving the myregistry docker image to a file called myregistry.tar
admin@fcc1a71d8f61:~/myregistry$ docker save -o myregistry.tar myregistry
```

```
# Listing directory to confirm file has content
admin@fcc1a71d8f61:~/myregistry$ ls -la
total 442368
drwxrwxr-x 1 admin admin        48 Sep  7 20:44 .
drwxr-xr-x 1 admin admin       186 Sep  7 15:46 ..
-rw-rw-r-- 1 admin admin      2490 Sep  7 15:46 Dockerfile
-rw------- 1 admin admin 452978176 Sep  7 20:44 myregistry.tar

# Loading the myregistry into Docker
admin@fcc1a71d8f61:~/myregistry$ docker load <  myregistry.tar
Loaded image: myregistry:latest
```

6.4 Running a local private registry

After you build the private registry image, you start it and make it available for users who must deploy Docker containers. This section shows how to run an insecure private registry. For more information about secure private registries, see section 6.5, “Deploying a secure private registry” on page 118.

Complete the following steps for the zCX instance where you built the insecure registry image.

1. Ensure that you are connected to the zCX instance with SSH.
2. Run the following command to confirm that the private registry image is available:
   docker images

Example 6-11 shows typical output that indicates that the **myregistry** image is available.

Example 6-11 Output of docker images command

```
admin@fcc1a71d8f61:~/myregistry$ docker images
REPOSITORY              TAG         IMAGE ID            CREATED             SIZE
myregistry              latest      082be17251ee        4 hours ago         441MB
ibm_zcx_zos_cli_image   latest      00f61e376fd0        2 days ago          363MB
ubuntu                  latest      f8835575bd80        3 weeks ago         64.8MB
s390x/ubuntu            16.04       fcae39a6d474        6 weeks ago         118MB
```

3. Run the following command to start your private registry server to listen on port 5000.

   docker run -d -p 5000:5000 --restart=always --name registry myregistry:latest

Table 6-2 describes each parameter for the command.

Table 6-2 Parameters information

Parameter	Description
-d	Run container in background and print container ID
-p 5000:5000	Publish a container's port(s) to the host. The private registry now listens on port 5000.
--restart=always	Start this container during system startup
--name registry	The name of the container
myregistry:latest	Image name

4. Run the following command to confirm that the registry container is started and listening on 5000 port: **`docker ps`**

Example 6-12 shows typical output for this command.

Example 6-12 Output of docker run command showing that the registry is up and listening on 5000 port

```
CONTAINER ID   IMAGE                    COMMAND                   CREATED
admin@fcc1a71d8f61:~/myregistry$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS          PORTS
NAMES
ec7844f14e0c   myregistry:latest      "registry serve /etc…"   15 minutes ago  Up 15 minutes   0.0.0.0:5000->5000/tcp
registry
fcc1a71d8f61  ibm_zcx_zos_cli_image "sudo /usr/sbin/sshd…"  2 days ago     Up 29 hours    8022/tcp, 0.0.0.0:8022->22/tcp
ibm_zcx_zos_cli
```

Note: The private registry that you have deployed in this procedure is insecure.

6.4.1 Stopping a private registry

Although it is not typically necessary to stop a private registry, you run the following command to do so: **`docker container stop registry`**

Note: You can also use **`docker stop registry`** command without the word **container** to take down a container. Both commands can be used to accomplish this goal.

6.4.2 Removing a private registry

To remove the private registry, use the following commands:

```
docker container stop registry
docker container rm -v registry
```

6.5 Deploying a secure private registry

This section describes these aspects of deployment for a secure private registry:

- How to configure your zCX appliance to run and use a secure private registry. The steps are very similar to setting up an insecure registry. However, additional configuration steps are required to create the TLS certificates that encrypt the connections between Docker client and the registry.
- How to pull an image from Docker Hub and push it into a secure private registry.
- How to load Docker images into your zCX registry, even when it is running behind a firewall and does not have direct internet access to Docker Hub.

The insecure mode is the easiest way to deploy a private registry and make images available for your application developers. However, it is only recommended that an insecure Docker registry be used for performing isolated tests. Check your security policies to confirm whether you can run an insecure registry.

Note: If your zCX is running behind a firewall and you have Docker clients in a different network of your private registry, ensure that you open TCP port 5000 in your firewall.

By default, Docker uses HTTPS protocol instead of HTTP to connect to Docker registry.

The zCX appliance supports both secure (using TLS) and insecure modes.

The secure private registry is highly recommended configuration for local registries and it should be the preferred choice when deploying images for production environments. This mode uses TLS certificates for communication between Docker client and Docker registry.

To build a secure private registry, you must apply specific settings in the zCX instance.

- If you have deployed the zCX appliance and provided the required Docker settings during deployment phase, you do not to perform any update now.
- Otherwise, you must reconfigure the zCX instance by using IBM z/OS Management Facility to ensure that the Docker settings are properly defined. The following changes for the zCx instance are available during reconfiguration workflow.
  - CPU number
  - Memory size
  - zCX Hostname
  - zCX IP address
  - DNS
  - Docker settings
  - Docker Admin user
  - LDAP settings
  - Docker Admin SSH key

This section provides information about the updating of only Docker settings.

To run the reconfiguration workflow, you must create a new workflow in z/OSMF as described in Chapter 4, “Provisioning and managing your first z/OS Container” on page 47. The file path of the workflow definition file for reconfiguration of a zCX instance is as follows: **`/usr/lpp/zcx_zos/workflows/reconfiguration.xml`**.

You can start the reconfiguration workflow without having to stop the instance in advance. But as last steps of the workflow, you are required to stop and restart the instance so that the new settings are applied.

Here is the general process of the workflow: Step 1 of the workflow retrieves the instance information. Then, Step 2 gathers appliance instance properties that are currently applied and opens the settings for your update.

To configure the Docker configuration settings for your zCX, complete the following procedure.

1. Click the *zCX Docker Configuration* option as detailed Figure 6-3.

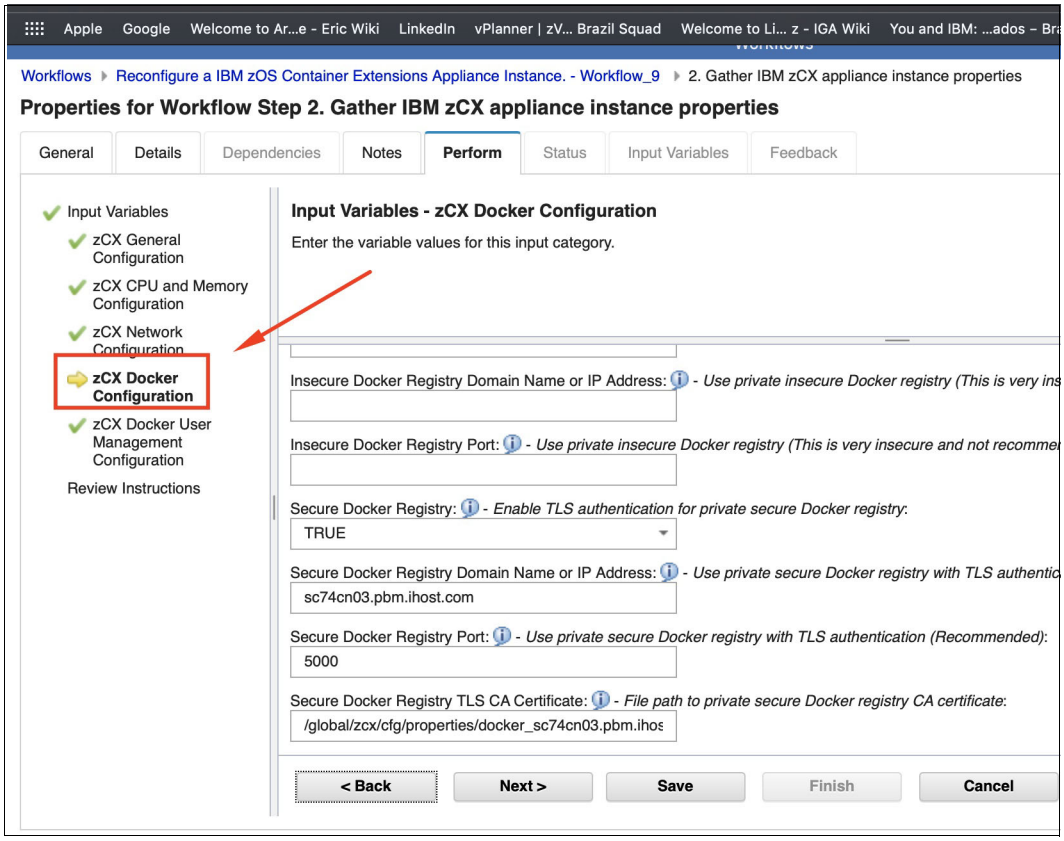


Figure 6-3 zCX Docker Configuration options

2. Update the settings.

   a. For an *insecure* private registry, update the following fields:

Table 6-3 Settings for an insecure private registry

Parameter	Value
Insecure Docker Registry Domain Name or IP address	Put the zCX IP name or IP Address (in other words, **`129.40.23.70`**)
Insecure Docker Registry Port	Put the Docker Registry port (in other words, **`5000`**)
Secure Docker Registry	Select FALSE

   b. For a *secure* private registry

Table 6-4 Settings for a secure private registry

Parameter	Value
Secure Docker Registry	Set it to YES
Secure Docker Registry Domain or IP address	Put the zCX IP name or IP Address (in other words, **`sc74cn03.pbm.ihost.com`**)
Secure Docker Registry Port	Put the Docker Registry port. (in other words, **`5000`**)
Secure Docker Registry TLS CA certificate	Put the file name for the TLS CA certificate. **Sample:** `/global/zcx/cfg/properties/docker_sc74cn03.pbm.ihost.com.crt`

Note: The **`docker tag`** command accepts either zCX IP name or an IP address. Important: If you use IP address in your zCX Docker configuration, ensure you also use IP address when use **`docker tag`** to manage your images. The same requirement applies to the use of the zCX IP name.

Example: If you use `sc74cn03.pbm.ihost.com` in Docker settings in your zCX, use this address every time that you use **`docker tag`**, **`docker pull,`** and **`docker push`** commands.

3. Click **Next** until you reach the windows where you review instructions.
4. Click **Finish**.
5. Select Step 3 of the workflow (**Start IBM zCX appliance reconfiguration**) and select to **Actions > Perform**. When requested, choose to run the rest of the workflow in automated mode.

The workflow halts as it reaches the steps where you stop and restart the zCX instance. You must manually stop and restart the instance to finish the workflow.

When the instance comes up again, connect through SSH and complete the steps in the next section.

After you confirm that the zCX settings for a secure registry are correct applied, complete the steps in the next section. Notice that you need a Windows or Linux system with internet access to be able to download Docker images from Docker Hub. In our case, we installed a SuSE Linux Enterprise Server 12, and attached it to a network that had internet access.

6.5.1 When zCX appliance is behind a firewall

In some organizations, the zCX instances might not have direct internet access. Therefore, the Docker client that runs on the zCX cannot pull Docker images from Docker Hub.

In such cases, you have options to download these images by using a laptop or any other system that has access to internet. In fact, you download the images from a host that has access to internet and then copy these images to a zCX instance.

These are the requirements to deploy a secure private registry behind a firewall.

- One system (container or stand-alone) with internet access
- Docker package installed (Docker client pulls images from Docker Hub)

Later in this section, we show how to download images by using a Linux instance and load them onto the zCX appliance. However, the procedure can be used on a computer that runs Windows or Macintosh if you have Docker client installed there.

This publication does not cover Docker installation for Windows, Macintosh, and other Linux distributions. Installation for those platforms is described at the following web pages:

- For Windows: `https://docs.docker.com/v17.12/docker-for-windows/install/`
- For Linux: `https://docs.docker.com/v17.12/install/#time-based-release-schedule`
- For Ubuntu: `https://docs.docker.com/v17.12/install/linux/docker-ce/ubuntu/`
- For Macintosh: `https://docs.docker.com/v17.12/docker-for-mac/install/`

The following steps describe deployment of a private secure registry in SuSE Linux. The procedure is very similar for other operating systems.

For SuSE Linux:

1. The Docker package is available in SuSE Linux packages repository. We installed Docker by using the **zypper install -y docker** command.

Example 6-13 Installing Docker on our SuSE Linux Server 12

```
mylinux:~ # zypper install -y docker
Retrieving repository 'SLE-Module-Containers-Pool' metadata
.....................................................................[done]
Building repository 'SLE-Module-Containers-Pool' cache ...................[done]
Retrieving repository 'SLE-Module-Containers-Updates' metadata ............[done]
Building repository 'SLE-Module-Containers-Updates' cache .................[done]
Retrieving repository 'SLE-Module-Legacy12-Pool' metadata ..................[done]
Building repository 'SLE-Module-Legacy12-Pool' cache ......................[done]
Retrieving repository 'SLE-Module-Legacy12-Updates' metadata ..............[done]
Building repository 'SLE-Module-Legacy12-Updates' cache ...................[done]
Retrieving repository 'SLE-Module-Public-Cloud12-Pool' metadata ...........[done]
Building repository 'SLE-Module-Public-Cloud12-Pool' cache .................[done]
Retrieving repository 'SLE-Module-Public-Cloud12-Updates' metadata..........[done]
Building repository 'SLE-Module-Public-Cloud12-Updates' cache ............[done]
Retrieving repository 'SLE-Module-Web-Scripting12-Pool' metadata ...........[done]
Building repository 'SLE-Module-Web-Scripting12-Pool' cache ..............[done]
Retrieving repository 'SLE-Module-Web-Scripting12-Updates' metadata ......[done]
Building repository 'SLE-Module-Web-Scripting12-Updates' cache ............[done]
Retrieving repository 'SLE-SDK12-SP4-Product' metadata ....................[done]
Building repository 'SLE-SDK12-SP4-Product' cache .........................[done]
Retrieving repository 'SLE-SDK12-SP4-Updates' metadata .....................[done]
Building repository 'SLE-SDK12-SP4-Updates' cache .......................[done]
Retrieving repository 'SLES12-SP4-Pool' metadata ..........................[done]
Building repository 'SLES12-SP4-Pool' cache ...............................[done]
Retrieving repository 'SLES12-SP4-Updates' metadata ....................[done]
Building repository 'SLES12-SP4-Updates' cache..............................[done]
Loading repository data...
Reading installed packages...
Resolving package dependencies...

The following NEW package is going to be installed:
  docker

1 new package to install.
Overall download size: 23.7 MiB. Already cached: 0 B. After the operation,
additional 135.5 MiB will be used.
Continue? [y/n/...? shows all options] (y): y
Retrieving package docker-19.03.1_ce-98.46.1.s390x
(1/1),  23.7 MiB (135.5 MiB unpacked)
Retrieving: docker-19.03.1_ce-98.46.1.s390x.rpm ............................[done]
Checking for file conflicts: ..............................................[done]
(1/1) Installing: docker-19.03.1_ce-98.46.1.s390x ..........................[done]
Additional rpm output:
Updating /etc/sysconfig/docker...
```

2. Start and enable Docker service to run during boot time by using these commands:

   systemctl start docker
   systemctl enable docker

3. Pull images from Docker Hub. Example 6-14 shows the commands and output of each.

```
docker pull postgres
docker pull ubuntu
docker pull nginx
docker pull ibmcom/registry-s390x:2.6.2.3
```

Example 6-14 Pulling Images from Docker Hub

```
# Pulling postgres image from Docker Hub
mylinux:~ # docker pull postgres
Using default tag: latest
latest: Pulling from library/postgres
054f83bb2543: Pull complete
e6fba16700a1: Pull complete
771747acead7: Pull complete
81e217460edc: Pull complete
f5045830171f: Pull complete
fa5edc5d2fdc: Pull complete
3916e82241bb: Pull complete
2fafafc32d20: Pull complete
3295a1ae7581: Pull complete
2125461a7eaf: Pull complete
72ecc4928948: Pull complete
59138b91491e: Pull complete
45e6b2e69d1a: Pull complete
12da3b1ea881: Pull complete
Digest: sha256:f766d03bb49c7dd16fe32c1eb2645d13cb315adc029557cd1e4a0a9c094933d5
Status: Downloaded newer image for postgres:latest
docker.io/library/postgres:latest

# Pulling ubuntu image from Docker Hub
mylinux:~ # docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
fa28e4de4daa: Pull complete
796cdcc842c0: Pull complete
1441a5fa9377: Pull complete
6ed3cedf561a: Pull complete
Digest: sha256:d1d454df0f579c6be4d8161d227462d69e163a8ff9d20a847533989cf0c94d90
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest

# Pulling nginx image from Docker Hub
mylinux:~ # docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
d3a78a1911c6: Pull complete
f7c23a53129b: Pull complete
f1887ddbfe67: Pull complete
Digest: sha256:53ddb41e46de3d63376579acf46f9a41a8d7de33645db47a486de9769201fec9
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest

mylinux:~ # docker pull ibmcom/registry-s390x:2.6.2.3
2.6.2.3: Pulling from ibmcom/registry-s390x
c5e4ba148bd1: Pull complete
8a4ff7bf8595: Pull complete
7b44230264d3: Pull complete
fbdb6d7676c0: Pull complete
b3e9a1a55ee2: Pull complete
db8b17a3eddc: Pull complete
```

```
Digest: sha256:96c5c0696f9df3ac582d5b9eb1fae11130d591af776a857f6ccc2672bb2a5f65
Status: Downloaded newer image for ibmcom/registry-s390x:2.6.2.3
docker.io/ibmcom/registry-s390x:2.6.2.3
```

4. Save the images to a tar archive file by using these commands

```
mkdir /tmp/docker_images
cd /tmp/docker_images/
docker save -o postgres.tar postgres
docker save -o ubuntu.tar ubuntu
docker save -o nginx.tar nginx
docker save -o registry.tar ibmcom/registry-s390x:2.6.2.3
```

5. Copy your SSH private key — which was generated during zCX installation — and save it in the **user** directory folder of your Linux server. In our lab environment, we save the SSH private key in a **/tmp/docker_images** directory.

```
mylinux:/tmp/docker_images # pwd
/tmp/docker_images

mylinux:/tmp/docker_images # ls -la
total 732792
drwx------  2 root root      4096 Sep 14 22:09 .
drwxrwxrwt 26 root root      4096 Sep 14 22:09 ..
-rw-------  1 root root       951 Sep  9 13:17 id_rsa
-rw-------  1 root root 128789504 Sep  9 13:59 nginx.tar
-rw-------  1 root root 248592896 Sep  9 13:59 postgres.tar
-rw-------  1 root root 305129984 Sep  9 13:59 registry.tar
-rw-------  1 root root  67096064 Sep  9 13:59 ubuntu.tar
```

6. Upload the image files to your zCX instance. Example 6-15 shows the commands to use.

Note: The zCX appliance is using secure SSH keys for authentication. To enable copying of files, you must copy your private SSH key (usually **id_rsa**) to your Linux server.

If you followed earlier instructions to access your zCX instance, the private and public SSH keys are located under your home directory in USS (UNIX Systems Services), similar to this example path:
/u/*<zos_userid>*/.ssh/
where *<zos_userid>* is your z/OS user ID.

Quick instructions:

1. Connect to z/OS instance through SSH.
2. Go to the **/u/*<zos_userid>*/.ssh/** folder
3. Issue **cat id_rsa** command to get content.
4. Go to the Linux Server and create a **id_rsa** file with the same content that you got in Step 3.

Remember to use the **-i /*some_folder*/id_rsa** parameter when you use **scp**.

Example 6-15 Uploading files to zCX instance

```
# Creating /home/admin/docker_images directory in your zCX instance
mylinux:/tmp/docker_images # ssh -p 8022 -i /tmp/docker_images/id_rsa admin@129.40.23.70
mkdir -p /home/admin/docker_images/

# Uploading images to the zCX instance
mylinux:/tmp/docker_images # scp -P 8022 -i /tmp/docker_images/id_rsa
/tmp/docker_images/*.tar admin@129.40.23.70:/home/admin/docker_images/
The authenticity of host '[129.40.23.70]:8022 ([129.40.23.70]:8022)' can't be established.
ECDSA key fingerprint is SHA256:IA5Sw4oNK7wODhFGkH4ayyJNvT73m1uvQprHjqOC41M.
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added '[129.40.23.70]:8022' (ECDSA) to the list of known hosts.
nginx.tar
100%  123MB   3.6MB/s   00:34
postgres.tar
100%  237MB   3.7MB/s   01:04
registry.tar
100%  291MB   3.8MB/s   01:17
ubuntu.tar
100%   64MB   3.8MB/s   00:17
```

5. Connect to the zCX instance through SSH (here is a sample command: **ssh -i id_rsa admin@129.40.23.70**), and load the Docker images.

Example 6-16 Loading Docker images

```
#Loading nginx image
admin@fcc1a71d8f61:~/docker_images$ docker load < nginx.tar
767040e8a7cf: Loading layer [==================================================>]
70.47MB/70.47MB
53fafafd1309: Loading layer [==================================================>]
58.29MB/58.29MB
5dd6f04549c3: Loading layer [==================================================>]
3.584kB/3.584kB
Loaded image: nginx:latest

# Loading Postgres image
admin@fcc1a71d8f61:~/docker_images$ docker load < postgres.tar
60b345ec3d8b: Loading layer [==================================================>]
62.25MB/62.25MB
86c65972e0f4: Loading layer [==================================================>]
11.42MB/11.42MB
6d418647cfe8: Loading layer [==================================================>]
343.6kB/343.6kB
ec29759a67e8: Loading layer [==================================================>]
4.215MB/4.215MB
b2bfc6205025: Loading layer [==================================================>]
17.1MB/17.1MB
88ba72943a90: Loading layer [==================================================>]
1.107MB/1.107MB
283f83842df4: Loading layer [==================================================>]
1.536kB/1.536kB
baf4d43e035e: Loading layer [==================================================>]
8.704kB/8.704kB
e2caf389d4ad: Loading layer [==================================================>]
152MB/152MB
9c6d4a33c1e8: Loading layer [==================================================>]
56.32kB/56.32kB
32302d217f8f: Loading layer [==================================================>]
2.048kB/2.048kB
80176049e5f6: Loading layer [==================================================>]
3.072kB/3.072kB
04e292c08f4b: Loading layer [==================================================>]
8.704kB/8.704kB
c4aef49e1726: Loading layer [==================================================>]
1.536kB/1.536kB
Loaded image: postgres:latest

# Loading Ubuntu Image
admin@fcc1a71d8f61:~/docker_images$ docker load < ubuntu.tar
5d5ce045d79f: Loading layer [==================================================>]
66.06MB/66.06MB
```

```
3aa2670921c9: Loading layer [==================================================>]
991.7kB/991.7kB
7f7250289cd1: Loading layer [==================================================>]
15.87kB/15.87kB
0e07ae918a65: Loading layer [==================================================>]
3.072kB/3.072kB
Loaded image: ubuntu:latestt

# Loading ibmcom/registry-s390x Image
admin@570e9473805e:~/docker_images$ docker load < registry.tar
Loaded image: ibmcom/registry-s390x:2.6.2.3
```

6. Tag the images with information about your private registry.

```
docker tag nginx sc74cn03.pbm.ihost.com:5000/nginx
docker tag postgres sc74cn03.pbm.ihost.com:5000/postgres
docker tag ubuntu sc74cn03.pbm.ihost.com:5000/ubuntu
docker tag ibmcom/registry-s390x:2.6.2.3
sc74cn03.pbm.ihost.com:5000/registry-s390x:2.6.2.3
```

7. Run the **docker images** command to check images after tagging. Example 6-17 show typical output of this command.

Example 6-17 Typical output of the docker images command

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ibm_zcx_zos_cli_image	latest	38c3c78a03a3	5 days ago	363MB
websphere-liberty	latest	161e444fb996	5 days ago	504MB
myregistry	latest	082be17251ee	7 days ago	441MB
nginx	latest	a7b54eb09cb6	4 weeks ago	125MB
sc74cn03.pbm.ihost.com:5000/nginx	latest	a7b54eb09cb6	4 weeks ago	125MB
ubuntu	latest	f8835575bd80	4 weeks ago	64.8MB
sc74cn03.pbm.ihost.com:5000/ubuntu	1604	f8835575bd80	4 weeks ago	64.8MB
sc74cn03.pbm.ihost.com:5000/ubuntu	latest	f8835575bd80	4 weeks ago	64.8MB
postgres	latest	abeb59ce1e39	4 weeks ago	241MB
sc74cn03.pbm.ihost.com:5000/postgres	latest	abeb59ce1e39	4 weeks ago	241MB
ubuntu	16.04	fcae39a6d474	7 weeks ago	118MB
s390x/ubuntu	16.04	fcae39a6d474	7 weeks ago	118MB
ibmcom/registry-s390x	2.6.2.3	7fb3fb9dd9f0	8 months ago	293MB
sc74cn03.pbm.ihost.com:5000/registry-s390x	2.6.2.3	7fb3fb9dd9f0	8 months ago	293MB
store/ibmcorp/mqadvanced-server-dev	9.1.0.0	7c66e184a2a7	10 months ago	871MB

8. Run the following commands while you are connected to a Linux Server.
   a. To generate your own certificate, use the following commands:

```
mkdir -p /tmp/certs
openssl req \
   -newkey rsa:4096 -nodes -sha256 \
   -keyout /tmp/certs/docker_domain.key \
   -x509 -days 365 -out /tmp/certs/docker_domain.crt
```

Note: Be sure to use the zCX hostname as Common Name.

For information about creating TLS certificates on Windows and Macintosh with the Docker client, see Section 6.6, “Creating TLS certificates on Windows or Macintosh” on page 130.

Example 6-18 shows the information that we entered for our system in bold (in the lower section of the example). You must enter the specific information that corresponds to your system and location.

Example 6-18 Using openssl to create the TLS certificate

```
mylinux:~ # openssl req \
> -newkey rsa:4096 -nodes -sha256 \
> -keyout /tmp/certs/docker_domain.key \
> -x509 -days 365 -out /tmp/certs/docker_domain.crt
Generating a 4096 bit RSA private key
.............................................................++++
.............................................................++++
writing new private key to '/tmp/certs/docker_domain.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:NY
Locality Name (eg, city) []:Poughkeepsie
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IBM
Organizational Unit Name (eg, section) []:Redbooks
Common Name (e.g. server FQDN or YOUR name) []:sc74cn03.pbm.ihost.com
Email Address []:emarins@br.ibm.com
```

9. If you have more than one ZCX instance, it is recommended that you rename the new created files to avoid conflicts with other instances, for example:
   - Rename **docker_domain.crt** file to **docker_sc74cn03.pbm.ihost.com.crt**
   - Rename **docker_domain.key** file to **docker_sc74cn03.pbm.ihost.com.key**

> **Note:** If you are managing several repositories, use the DNS name to help to easily identify the registry owner for these files.

Example 6-19 Renaming the certificates

```
# Listing directory before renaming
mylinux:/tmp/certs # ls -la
total 16
drwx------  2 root root 4096 Sep  9 20:58 .
drwxrwxrwt 26 root root 4096 Sep  9 21:11 ..
-rw-------  1 root root 2143 Sep  9 20:58 docker_domain.crt
-rw-------  1 root root 3268 Sep  9 20:58 docker_domain.key


# Renaming files
mylinux:/tmp/certs # mv docker_domain.crt docker_sc74cn03.pbm.ihost.com.crt
mylinux:/tmp/certs # mv docker_domain.key docker_sc74cn03.pbm.ihost.com.key

# Listing directory after renaming
mylinux:/tmp/certs # ls -la
total 16
drwx------  2 root root 4096 Sep  9 21:13 .
drwxrwxrwt 26 root root 4096 Sep  9 21:13 ..
-rw-------  1 root root 2143 Sep  9 20:58 docker_sc74cn03.pbm.ihost.com.crt
-rw-------  1 root root 3268 Sep  9 20:58 docker_sc74cn03.pbm.ihost.com.key
```

10. Send the certificate files to your z/OS instance via SSH by using the command in Example 6-20.

Note: Consider this background information before you complete this step:

- Our z/OS instance is **wtsc74.pbm.ihost.com** (not the zCX instance).
- Our target folder under UNIX System Services (USS) is **/global/zcx/cfg/properties/**
- Our z/OS privileged user is **zcxprv2**, which was defined earlier in this publication.

Example 6-20 Sending certificate files to our z/OS instance

```
mylinux:/tmp/certs # scp docker_sc74cn03.pbm.ihost.com.*
zcxprv2@wtsc74.pbm.ihost.com:/global/zcx/cfg/properties/
The authenticity of host 'wtsc74.pbm.ihost.com (129.40.23.1)' can't be
established.
RSA key fingerprint is SHA256:v27XurOE1iVOhnVzzsL4a3X97F9bHOMcumDxFpE3Ogc.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'wtsc74.pbm.ihost.com,129.40.23.1' (RSA) to the list of
known hosts.
zcxprv2@wtsc74.pbm.ihost.com's password:
docker_sc74cn03.pbm.ihost.com.crt
100% 2143     2.1KB/s   00:00
docker_sc74cn03.pbm.ihost.com.key
100% 3268     3.2KB/s   00:00
```

11. As a best practice for a private registry, we run the following commands to create a persistent storage for **/var/lib/registry** and **/certs** folders:

```
docker volume create certs_volume
docker volume create registry_volume
```

Note: Chapter 10, “Persistent data” on page 231 gives an overview of concepts regarding persistent data.

12. The next step is to upload the certificate files from Linux server to the zCX instance.

Note: Remember to use the private SSH key to access your zCX instance while using admin user ID. We use the **id_rsa** key file that was created in Step 6, which is located in **/tmp/docker_images**.

Example 6-21 Sending certificate files to our zCX instance

```
# Creating /home/admin/certs directory in your zCX instance
mylinux:/tmp/certs # ssh -p 8022 -i /tmp/docker_images/id_rsa admin@129.40.23.70
mkdir -p /home/admin/certs/

# Uploading images to the zCX instance
mylinux:/tmp/certs # scp -P 8022 -i /tmp/docker_images/id_rsa
/tmp/certs/docker_sc74cn03.pbm.ihost.com.* admin@129.40.23.70:/home/admin/certs/
docker_sc74cn03.pbm.ihost.com.crt 100% 2143     2.1KB/s   00:00
docker_sc74cn03.pbm.ihost.com.key 100% 3268     3.2KB/s   00:00
```

13. Run the following command to start your private registry server to listen on port 5000.

```
docker run -d \
   --restart=always \
   --name registry \
   -v certs_volume:/certs \
   -v registry_volume:/var/lib/registry \
   -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/docker_sc74cn03.pbm.ihost.com.crt \
   -e REGISTRY_HTTP_TLS_KEY=/certs/docker_sc74cn03.pbm.ihost.com.key \
   -p 5000:5000 \
   ibmcom/registry-s390x:2.6.2.3
```

Example 6-22 shows output of the command, which is the new Docker image ID.

Example 6-22 Output of docker run command

```
2559a944f4aa1720b1ab062107f18706b4e0063da88904e688721ca7743d432d
```

14. Follow these instructions to copy the TLS certificates into the new **container.commands** registry.

   a. First take down the Docker registry container:
      docker stop registry

   b. Copy local folder called cert into registry container:
      docker cp /home/admin/certs/ registry:./

   c. Now start Docker registry container:
      docker start registry

   d. Confirm that container is up and is using HTTPS by running the command that is listed in Example 6-23.

Example 6-23 Confirming images are available in your secure private registry

```
admin@570e9473805e:~$ curl -k https://sc74cn03.pbm.ihost.com:5000/v2/_catalog
{{"repositories":[]}
```

15. Now, run the following commands to push images to your private registry.

```
docker push sc74cn03.pbm.ihost.com:5000/nginx
docker push sc74cn03.pbm.ihost.com:5000/ubuntu
docker push sc74cn03.pbm.ihost.com:5000/postgres
docker push sc74cn03.pbm.ihost.com:5000/registry-s390x:2.6.2.3
```

Example 6-24 shows typical output for these commands.

Example 6-24 Pushing images to your private registry

```
admin@570e9473805e:~/docker_images$ docker push sc74cn03.pbm.ihost.com:5000/nginx
The push refers to repository [sc74cn03.pbm.ihost.com:5000/nginx]
5dd6f04549c3: Layer already exists
53fafafd1309: Layer already exists
767040e8a7cf: Layer already exists
latest: digest: sha256:69e12814dcae07a6a1095b2d26d4bac5b2b09966d230cf6a0b972843116a8347
size: 948
admin@570e9473805e:~/docker_images$ docker push sc74cn03.pbm.ihost.com:5000/ubuntu
The push refers to repository [sc74cn03.pbm.ihost.com:5000/ubuntu]
0e07ae918a65: Layer already exists
7f7250289cd1: Layer already exists
3aa2670921c9: Layer already exists
5d5ce045d79f: Layer already exists
```

```
1604: digest: sha256:0d66f428f847be647731237ae59b41b7d9734d4b43f37c03fffde8c10bc0134a size:
1152
0e07ae918a65: Layer already exists
7f7250289cd1: Layer already exists
3aa2670921c9: Layer already exists
5d5ce045d79f: Layer already exists
latest: digest: sha256:0d66f428f847be647731237ae59b41b7d9734d4b43f37c03fffde8c10bc0134a
size: 1152

admin@570e9473805e:~/docker_images$ docker push sc74cn03.pbm.ihost.com:5000/postgres
The push refers to repository [sc74cn03.pbm.ihost.com:5000/postgres]
c4aef49e1726: Layer already exists
04e292c08f4b: Layer already exists
80176049e5f6: Layer already exists
32302d217f8f: Layer already exists
9c6d4a33c1e8: Layer already exists
e2caf389d4ad: Layer already exists
baf4d43e035e: Layer already exists
283f83842df4: Layer already exists
88ba72943a90: Layer already exists
b2bfc6205025: Layer already exists
ec29759a67e8: Layer already exists
6d418647cfe8: Layer already exists
86c65972e0f4: Layer already exists
60b345ec3d8b: Layer already exists
latest: digest: sha256:d9b8b762e52d885335cfd8f9c84b0b9cf8df64da27a77ebaf17a875f5e87a894
size: 3245

admin@570e9473805e:~/docker_images$ docker push
sc74cn03.pbm.ihost.com:5000/registry-s390x:2.6.2.3
The push refers to repository [sc74cn03.pbm.ihost.com:5000/registry-s390x]
0b92cf7f1c78: Pushed
3848866aa24b: Pushed
05d5226eda27: Pushed
3c1abf982c00: Pushed
223899d9061b: Pushed
2660d69c0ffd: Pushed
2.6.2.3: digest: sha256:96c5c0696f9df3ac582d5b9eb1fae11130d591af776a857f6ccc2672bb2a5f65
size: 1579
```

16. Confirm what images exist in your private registry by running the following command:

    curl -k https://sc74cn03.pbm.ihost.com:5000/v2/_catalog

Example 6-25 shows typical output for this command:

Example 6-25 Output of curl command

```
{"repositories":["nginx","postgres","registry-s390x","ubuntu"]}
```

Alternatively, you can use your web browser to get this information. Open your preferred internet browser and access **https://sc74cn03.pbm.ihost.com:5000/v2/_catalog** link.

6.6 Creating TLS certificates on Windows or Macintosh

You might not have a Linux system where you can run the **openssl** command to create TLS certificates. In that case, you can follow the instructions in this section to generate the TLS

certificates on Windows or Macintosh. We use Docker to create a container, and then issue some commands to achieve this goal. Docker must be installed before you do the following steps.

1. In the Docker installation on your Windows or Macintosh, run the following command in the command-line interface:
   docker run -it ubuntu /bin/bash
2. Run the **apt-get update** command to update the package list as shown in Example 6-26.

Example 6-26 Updating package list by using apt-get update command

```
root@38d485c33e24:/# apt-get update
Get:1 http://ports.ubuntu.com/ubuntu-ports bionic InRelease [242 kB]
Get:2 http://ports.ubuntu.com/ubuntu-ports bionic-updates InRelease [88.7 kB]
Get:3 http://ports.ubuntu.com/ubuntu-ports bionic-backports InRelease [74.6 kB]
Get:4 http://ports.ubuntu.com/ubuntu-ports bionic-security InRelease [88.7 kB]
Get:5 http://ports.ubuntu.com/ubuntu-ports bionic/universe s390x Packages [10.8 MB]
Get:6 http://ports.ubuntu.com/ubuntu-ports bionic/multiverse s390x Packages [151 kB]
Get:7 http://ports.ubuntu.com/ubuntu-ports bionic/restricted s390x Packages [572 B]
Get:8 http://ports.ubuntu.com/ubuntu-ports bionic/main s390x Packages [1244 kB]
Get:9 http://ports.ubuntu.com/ubuntu-ports bionic-updates/main s390x Packages [662 kB]
Get:10 http://ports.ubuntu.com/ubuntu-ports bionic-updates/restricted s390x Packages [906 B]
Get:11 http://ports.ubuntu.com/ubuntu-ports bionic-updates/universe s390x Packages [1034 kB]
Get:12 http://ports.ubuntu.com/ubuntu-ports bionic-updates/multiverse s390x Packages [2465 B]
Get:13 http://ports.ubuntu.com/ubuntu-ports bionic-backports/universe s390x Packages [4227 B]
Get:14 http://ports.ubuntu.com/ubuntu-ports bionic-backports/main s390x Packages [2494 B]
Get:15 http://ports.ubuntu.com/ubuntu-ports bionic-security/multiverse s390x Packages [1674 B]
Get:16 http://ports.ubuntu.com/ubuntu-ports bionic-security/main s390x Packages [393 kB]
Get:17 http://ports.ubuntu.com/ubuntu-ports bionic-security/restricted s390x Packages [581 B]
Get:18 http://ports.ubuntu.com/ubuntu-ports bionic-security/universe s390x Packages [593 kB]
Fetched 15.4 MB in 9s (1638 kB/s)

Reading package lists... Done
```

3. Run the **apt-get install openssl** command to install the **openssl** package, as shown in Example 6-27.

Example 6-27 Installing openssl using apt-get install command

```
root@38d485c33e24:/# apt-get install openssl
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libssl1.1
Suggested packages:
  ca-certificates
The following NEW packages will be installed:
  libssl1.1 openssl
0 upgraded, 2 newly installed, 0 to remove and 17 not upgraded.
Need to get 1568 kB of archives.
After this operation, 4935 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://ports.ubuntu.com/ubuntu-ports bionic-updates/main s390x libssl1.1 s390x
1.1.1-1ubuntu2.1~18.04.4 [969 kB]
Get:2 http://ports.ubuntu.com/ubuntu-ports bionic-updates/main s390x openssl s390x
1.1.1-1ubuntu2.1~18.04.4 [599 kB]
Fetched 1568 kB in 1s (1632 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package libssl1.1:s390x.
(Reading database ... 4041 files and directories currently installed.)
```

```
Preparing to unpack .../libssl1.1_1.1.1-1ubuntu2.1~18.04.4_s390x.deb ...
Unpacking libssl1.1:s390x (1.1.1-1ubuntu2.1~18.04.4) ...
Selecting previously unselected package openssl.
Preparing to unpack .../openssl_1.1.1-1ubuntu2.1~18.04.4_s390x.deb ...
Unpacking openssl (1.1.1-1ubuntu2.1~18.04.4) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Setting up libssl1.1:s390x (1.1.1-1ubuntu2.1~18.04.4) ...
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based frontend cannot
be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 76.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (Can't locate Term/ReadLine.pm in @INC (you may need to install the Term::ReadLine
module) (@INC contains: /etc/perl /usr/local/lib/s390x-linux-gnu/perl/5.26.1
/usr/local/share/perl/5.26.1 /usr/lib/s390x-linux-gnu/perl5/5.26 /usr/share/perl5
/usr/lib/s390x-linux-gnu/perl/5.26 /usr/share/perl/5.26 /usr/local/lib/site_perl
/usr/lib/s390x-linux-gnu/perl-base) at /usr/share/perl5/Debconf/FrontEnd/Readline.pm line
7.)
debconf: falling back to frontend: Teletype
Setting up openssl (1.1.1-1ubuntu2.1~18.04.4) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
```

4. Run the following commands connected to create the TLS certificates.
   a. To generate your own certificate, run the following commands:

```
mkdir -p /tmp/certs
openssl req \
   -newkey rsa:4096 -nodes -sha256 \
   -keyout /tmp/certs/docker_domain.key \
   -x509 -days 365 -out /tmp/certs/docker_domain.crt
```

Note: Be sure to use the zCX hostname as Common Name.

In Example 6-18, ensure that you respond to the questions with information that is accurate for your system and location. We highlighted in bold the responses that were valid for our environment.

Example 6-28 Using openssl to create the TLS certificate

```
root@38d485c33e24:/# openssl req \
> -newkey rsa:4096 -nodes -sha256 \
> -keyout /tmp/certs/docker_domain.key \
> -x509 -days 365 -out /tmp/certs/docker_domain.crt
Can't load /root/.rnd into RNG
4396277008160:error:2406F079:random number generator:RAND_load_file:Cannot open
file:../crypto/rand/randfile.c:88:Filename=/root/.rnd
Generating a RSA private key
.............................................................++++
.............................................................++++
writing new private key to '/tmp/certs/docker_domain.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
```

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:NY
Locality Name (eg, city) []:Poughkeepsie
Organization Name (eg, company) [Internet Widgits Pty Ltd]:IBM
Organizational Unit Name (eg, section) []:Redbooks
Common Name (e.g. server FQDN or YOUR name) []:sc74cn03.pbm.ihost.com
Email Address []:emarins@br.ibm.com
```

5. If you have more than one ZCX instance, it is recommended that you rename the newly created files to avoid conflicts with other instances, for example:
   - Rename **docker_domain.crt** file to **docker_sc74cn03.pbm.ihost.com.crt**
   - Rename **docker_domain.key** file to **docker_sc74cn03.pbm.ihost.com.key**

> **Note:** If you are managing several repositories, use the DNS name to help to easily identify the registry owner for these files.

Example 6-29 Renaming the certificates

```
root@38d485c33e24:/# cd /tmp/certs

root@38d485c33e24:/tmp/certs# ls -la
total 8
drwxr-xr-x 1 root root   68 Sep 15 00:33 .
drwxrwxrwt 1 root root   10 Sep 15 00:32 ..
-rw-r--r-- 1 root root 2163 Sep 15 00:33 docker_domain.crt
-rw------- 1 root root 3272 Sep 15 00:32 docker_domain.key

root@38d485c33e24:/tmp/certs# mv docker_domain.crt
docker_sc74cn03.pbm.ihost.com.crt
root@38d485c33e24:/tmp/certs# mv docker_domain.key
docker_sc74cn03.pbm.ihost.com.key

root@38d485c33e24:/tmp/certs# ls -la
total 8
drwxr-xr-x 1 root root  132 Sep 15 00:35 .
drwxrwxrwt 1 root root   10 Sep 15 00:32 ..
-rw-r--r-- 1 root root 2163 Sep 15 00:33 docker_sc74cn03.pbm.ihost.com.crt
-rw------- 1 root root 3272 Sep 15 00:32 docker_sc74cn03.pbm.ihost.com.key
root@38d485c33e24:/tmp/certs#
```

6. Run the **apt-get -y install openssh-client** command to install the **scp** utility.
7. Now, send the certificate files to your z/OS instance by using the **scp** command that is shown in Example 6-20.

> **Note:** Consider this background information before you complete this step:
> - Our z/OS instance is **wtsc74.pbm.ihost.com** (not the zCX instance).
> - Our target folder under UNIX System Services (USS) is **/global/zcx/cfg/properties/**
> - Our z/OS privileged user is **zcxprv2**, which was defined earlier in this publication.

Example 6-30 Sending certificate files to our z/OS instance

```
root@38d485c33e24:/tmp/certs# scp docker_sc74cn03.pbm.ihost.com.*
zcxprv2@wtsc74.pbm.ihost.com:/global/zcx/cfg/properties/
The authenticity of host 'wtsc74.pbm.ihost.com (129.40.23.1)' can't be
established.
RSA key fingerprint is SHA256:v27XurOE1iVOhnVzzsL4a3X97F9bHOMcumDxFpE3Ogc.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'wtsc74.pbm.ihost.com,129.40.23.1' (RSA) to the list of
known hosts.
zcxprv2@wtsc74.pbm.ihost.com's password:
docker_sc74cn03.pbm.ihost.com.crt
100% 2163     3.5MB/s   00:00
docker_sc74cn03.pbm.ihost.com.key
100% 3272     4.7MB/s   00:00
```

8. The final step is to upload the certificate files from your container to the zCX instance.
9.

Note: Remember to use the private SSH key to access your zCX instance while using admin user ID. We use the **id_rsa** key file that was created in Step 6, which is located in **/tmp/docker_images**.

Note: Remember to use the private SSH key to access your zCX instance while using admin user ID. You must copy or create an **id_rsa** key file. If you need to know the **vi** command that is required to create this file, here it is: **apt-get install vim**

Remember to run the **chmod 700 id_rsa** command to ensure that your private key has appropriate file permissions.

Example 6-31 Sending certificate files to our zCX instance

```
# Creating /home/admin/certs directory in your zCX instance
root@38d485c33e24:/tmp/certs# ssh -p 8022 -i id_rsa admin@129.40.23.70 mkdir -p
/home/admin/certs/

# Uploading images to the zCX instance
root@38d485c33e24:/tmp/certs# scp -P 8022 -i id_rsa
/tmp/certs/docker_sc74cn03.pbm.ihost.com.* admin@129.40.23.70:/home/admin/certs2/
docker_sc74cn03.pbm.ihost.com.crt
100% 2163     1.2MB/s   00:00
docker_sc74cn03.pbm.ihost.com.key
100% 3272     2.5MB/s   00:00
```

6.7 Working with tags

This section provides some information and instructions about tagging in the Docker world.

The importance and impact of tagging an image is such that application developers need a better understanding of them. That way, they can effectively manage the Docker image repository (registry).

In fact, tagging is a critical component in a Docker environment, which might have thousands of Docker images and a high number of releases. Your ability to interpret and use tags helps you to successfully manage and deploy new application workloads in your zCX infrastructure. Consider the components of the following example Docker command:

docker pull sc74cn03.pbm.ihost.com:5000/ubuntu:16.04

Table 6-5 Information about a command that references tags

Parameter	Description
sc74cn03.pbm.ihost.com:5000	First part of the tag is a hostname (or IP) and port. Docker uses this information as the location of the registry.
ubuntu	Image name.
16.04	Image version (if not informed, Docker uses the default = latest).

The preceding example pulls an image from private registry (**sc74cn03.pbm.ihost.com**) on port **5000** and pushes it to your local registry. It is important to note the following points when you load a new container instance with the **Docker run** command by using this image name and version (**ubuntu:16.04**):

- If image is located, Docker uses it.
- Otherwise, Docker connects to the registry and downloads it.

The steps to tag an image are as follows:

1. Pull image from repository with the **docker pull** command.
2. Tag the image through use of the **docker tag** command.

Now, we describe how to download a WebSphere Liberty image from Docker Hub and tag it for our private registry.

1. Run the **docker pull websphere-liberty** command, as shown in Example 6-32.

Example 6-32 Pulling websphere-liberty from Docker Hub

```
# docker pull websphere-liberty
Using default tag: latest
latest: Pulling from library/websphere-liberty
45848351f60c: Pull complete
9d06bd1d66a7: Pull complete
46992b2c26a8: Pull complete
cee1faa53d30: Pull complete
0b9b57a4b4f2: Pull complete
1f41269b381c: Pull complete
55ddf29068d1: Pull complete
a4f182145137: Pull complete
6548610a91f0: Pull complete
ab297c326c1c: Pull complete
db3e4f1da7ea: Pull complete
4a6c54712062: Pull complete
b3568c94c253: Pull complete
15f8216147a2: Pull complete
Digest: sha256:55d2a383c232d29917272c286bece1947b316b723bc21803a8d29b1ee1d043ea
Status: Downloaded newer image for websphere-liberty:latest
docker.io/library/websphere-liberty:latest
```

2. Now, run the following command to tag this image for use in the **sc74cn03.pbm.ihost.com:5000** registry:

   docker tag websphere-liberty sc74cn03.pbm.ihost.com:5000/websphere-liberty:javaee8

3. To confirm that the tag was added, run the **docker images** command. Example 6-33 shows typical output.

Example 6-33 Authentication to Docker Hub

```
# docker images
REPOSITORY                                      TAG                 IMAGE ID            CREATED             SIZE
ibm_zcx_zos_cli_image                           latest              38c3c78a03a3        20 hours ago        363MB
websphere-liberty                               latest              161e444fb996        21 hours ago        504MB
sc74cn03.pbm.ihost.com:5000/websphere-liberty   javaee8             161e444fb996        21 hours ago        504MB
myregistry                                      latest              082be17251ee        3 days ago          441MB
sc74cn03.pbm.ihost.com:5000/nginx               latest              a7b54eb09cb6        3 weeks ago         125MB
nginx                                           latest              a7b54eb09cb6        3 weeks ago         125MB
ubuntu                                          latest              f8835575bd80        3 weeks ago         64.8MB
sc74cn03.pbm.ihost.com:5000/ubuntu              latest              f8835575bd80        3 weeks ago         64.8MB
postgres                                        latest              abeb59ce1e39        3 weeks ago         241MB
sc74cn03.pbm.ihost.com:5000/postgres            latest              abeb59ce1e39        3 weeks ago         241MB
ubuntu                                          16.04               fcae39a6d474        7 weeks ago         118MB
s390x/ubuntu                                    16.04               fcae39a6d474        7 weeks ago         118MB
store/ibmcorp/mqadvanced-server-dev             9.1.0.0             7c66e184a2a7        9 months ago        871MB$
```

Here is another example command that references tags: **docker pull** ubuntu**:16.04**

Table 6-6 Tagging Information

Parameter	Description
Not Specified	The first part of the tag is a hostname (or IP address) and port. Docker uses this information as the location of the registry.
ubuntu	Image Name
16.04	Image Version (if not otherwise informed, Docker uses the **default = latest image**)

This example tells the Docker client to download version 16.04 of the Ubuntu image from the default repository, which is Docker Hub, but only if image is not locally cached. After the download is complete, it remains in cache until you remove it or update it through use of the **docker pull** command.

> **Note:** When you use tags with a zCX private registry, ensure that the name of the repository that you use is the same as you specified it in zCX Docker configuration during zCX deployments.

Removing a tag

To remove a tag, see the example command in Example 6-21.

Example 6-34 Removing a tag

```
$ docker rmi sc74cn03.pbm.ihost.com:5000/websphere-liberty:javaee8
Untagged: sc74cn03.pbm.ihost.com:5000/websphere-liberty:javaee8
```

Additional information about tags can be found at
https://docs.docker.com/engine/reference/commandline/tag/

6.8 Deleting an image from a private Docker registry

This section provides instructions for removal of images. Deleting an image from private registry is not a trivial operation. If you must delete images from your private registry — for example, to recover disk space — use Docker registry APIs. Currently, the Docker client does not support this task.

For demonstration purposes, we clean up the Ubuntu image from our Registry repository.

1. First, we run the commands in Example 6-35 to push images to the private registry.

Example 6-35 Pushing images to the private registry

```
admin@570e9473805e:~$ docker push sc74cn03.pbm.ihost.com:5000/ubuntu:latest
The push refers to repository [sc74cn03.pbm.ihost.com:5000/ubuntu]
0e07ae918a65: Layer already exists
7f7250289cd1: Layer already exists
3aa2670921c9: Layer already exists
5d5ce045d79f: Layer already exists
latest: digest:
sha256:0d66f428f847be647731237ae59b41b7d9734d4b43f37c03fffde8c10bc0134a size: 1152

admin@570e9473805e:~$ docker push sc74cn03.pbm.ihost.com:5000/ubuntu:1604
The push refers to repository [sc74cn03.pbm.ihost.com:5000/ubuntu]
0e07ae918a65: Layer already exists
7f7250289cd1: Layer already exists
3aa2670921c9: Layer already exists
5d5ce045d79f: Layer already exists
1604: digest:
sha256:0d66f428f847be647731237ae59b41b7d9734d4b43f37c03fffde8c10bc0134a size: 1152
```

2. Now, run the following command to confirm that the expected images are available in our private registry.

Example 6-36 Listing images available

```
admin@570e9473805e:~$ curl -k -X GET
https://sc74cn03.pbm.ihost.com:5000/v2/_catalog
{"repositories":["nginx","postgres","ubuntu"]}
```

3. For the Ubuntu image, list available tags. Use command in Example 6-37.

Example 6-37 Listing tags for ubuntu image

```
admin@570e9473805e:~$ curl -k -X GET
https://sc74cn03.pbm.ihost.com:5000/v2/ubuntu/tags/list
{"name":"ubuntu","tags":["1604","latest"]}
```

4. Now, for the Ubuntu image, get the number that is generated by the sha256 algorithm.

Be aware that you must specify any tag available for the image. The command listed in Example 6-38 shows an example for latest tag.

Example 6-38 Getting the algorithm number for ubuntu image

```
curl -X GET -k -H "Accept: application/vnd.docker.distribution.manifest.v2+json" -sI
"https://sc74cn03.pbm.ihost.com:5000/v2/ubuntu/manifests/latest" |grep sha256 |grep -i
"docker-content-digest" | awk '{print $2}'
sha256:0d66f428f847be647731237ae59b41b7d9734d4b43f37c03fffde8c10bc0134a
```

The output of the preceding command is the information that is required to delete the image. It is a unique number that identifies the image in the registry repository.

5. Now, delete the image by using command listed in Example 6-39.

The API syntax for this command is as follows:
DELETE /v2/<*image_name*>/manifests/<*reference_number*>
where:
<*image_name*> = The image name.
<*reference_number*> = The number from the sha256 algorithm.

Example 6-39 Deleting an image

```
admin@570e9473805e:~$ curl -k -sk -X DELETE
https://sc74cn03.pbm.ihost.com:5000/v2/ubuntu/manifests/sha256:0d66f428f847be64773
1237ae59b41b7d9734d4b43f37c03fffde8c10bc0134a
admin@570e9473805e:~$
```

No output (the absence of output) confirms the success of the DELETE command as shown in Example 6-39.

6. Now, run the registry garbage collector command:

   docker exec registry registry garbage-collect /etc/docker/registry/config.yml

The output of the garbage-collect command is long. However, try to find information about blobs that are eligible for deletion as in Example 6-40.

Example 6-40 Garbage-collect output

```
27 blobs marked, 6 blobs and 0 manifests eligible for deletion
```

7. Restart the Docker registry.

Example 6-41 Restarting registry

```
admin@570e9473805e:~$ docker restart registry
registry
```

8. Verify repository directories.

Example 6-42 Verifying repository directories

```
admin@570e9473805e:~$ docker exec -it registry ls
/var/lib/registry/docker/registry/v2/repositories/
my-ubuntu  nginx  postgres  ubuntu
```

Each image has a directory in the registry under **/var/lib/registry/docker/registry/v2/repositories**. The next step shows how to remove it.

9. Issue the following command to delete the image directory from repository folder.

   docker exec -it registry rm -rf /var/lib/registry/docker/registry/v2/repositories/ubuntu

10. Now, confirm that the Ubuntu image was successfully removed.

Example 6-43 Confirming registry catalog

```
admin@570e9473805e:~$ curl -k -X GET
https://sc74cn03.pbm.ihost.com:5000/v2/_catalog
{"repositories":["nginx","postgres"]}
```

6.9 Using the private registry to create containers

Before application developers start to create new containers, you must instruct them about the private registry and how they can use it. Remember that images are available in your registry only if users upload images through the **docker push** command.

Users can get information about the repository through this URL:

http://*<your_private_registry>*/v2/*_catalog address*

where *<your_private_registry>* is the private registry address, as in this sample:

https://sc74cn03.pbm.ihost.com:5000/v2/_catalog

Consider a scenario where the **HTTPd** image was pushed into the private registry with address **sc74cn03.pbm.ihost.com:5000**. In this case, users can run the following command to create a web server container that uses the **HTTPd** image.

docker run -dit --name my-apache-app -p 8080:80 sc74cn03.pbm.ihost.com:5000/httpd

Run the **docker ps** command to confirm that **my-apache-app** is running as shown in Example 6-44.

Example 6-44 Output of docker ps command

```
admin@570e9473805e:~/myregistry$ docker ps
CONTAINER ID    IMAGE                              COMMAND                  CREATED             STATUS
PORTS                           NAMES
be87df66fc65    sc74cn03.pbm.ihost.com:5000/httpd  "httpd-foreground"       11 seconds ago      Up 4 seconds
0.0.0.0:8080->80/tcp            my-apache-app
844a59b71115    ibmcom/registry-s390x:2.6.2.3      "registry serve /etc…"   About an hour ago   Up About an hour
0.0.0.0:5000->5000/tcp          registry
570e9473805e    ibm_zcx_zos_cli_image              "sudo /usr/sbin/sshd…"   5 days ago          Up 3 days
8022/tcp, 0.0.0.0:8022->22/tcp   ibm_zcx_zos_cli
```

The container is running and accessible to application users through port 80.

This example showed how to deploy containers for your private registry. If you need to deploy containers with persistent data, see in Chapter 10, “Persistent data” on page 231.

7

Operation

This chapter describes the basic tasks for running and maintaining zCX. It includes the following sections:

- 7.1, “Software maintenance” on page 142
- 7.2, “Automation” on page 149
- 7.3, “Backup and recovery” on page 151
- 7.4, “Diagnosis” on page 156
- 7.5, “Monitoring with RMF on zCX instance level” on page 160
- 7.6, “Configuring Grafana to monitor zCX containers” on page 166
- 7.7, “Monitoring with Grafana on container level” on page 177

© Copyright IBM Corp. 2019. All rights reserved.

7.1 Software maintenance

zCX itself is a component of z/OS, but the Docker images that are used within zCX are not part of z/OS. As a result, different methods are required to update the software levels of zCX versus updating the application software that runs within the containers.

7.1.1 Maintenance for zCX

The application of service to zCX is a two-step process. In the first step, you install the maintenance into the z/OS system. In the second step, the installed service has to be picked up by the individual zCX instances.

Install zCX service into the system

zCX is a component of z/OS. So, fixes to zCX are delivered as PTFs and installed through SMP/E, which is the component in z/OS that is most commonly used to install products and fixes. You can and should order and install maintenance to zCX as part of your normal maintenance procedures for z/OS as a whole.

For more information on SMP/E, see the following this link in the IBM Knowledge Center: https://z-esrv01h.servicedpaor.de/zos/knowledgecenter/SSLTBW_2.4.0/com.ibm.zos.v2r4.gim/gim.htm

Activate Service for zCX instances

The installation of PTFs with SMP/E does not activate this service for any zCX instance that is provisioned. If you want to pick up the new service, it is not sufficient to restart an instance. Instead, to activate the service you must run the upgrade workflow in z/OSMF for each zCX instance separately.

To run the upgrade workflow, you must create a new workflow in z/OSMF as described in Chapter 4.8.2, “Select the zCX provisioning workflow” on page 61. The file path for the workflow definition file that upgrades a zCX instance is as follows:

/usr/lpp/zcx_zos/workflows/upgrade.xml

You can start the upgrade workflow without having to stop the instance in advance. But as last steps of the workflow, you are required to stop and restart the instance.

Step 1 of the workflow retrieves the instance information. Then, Step 2 checks for the upgrade version and shows you the currently active software level and also the level that is activated after this workflow runs, as shown in Figure 7-1 on page 143.

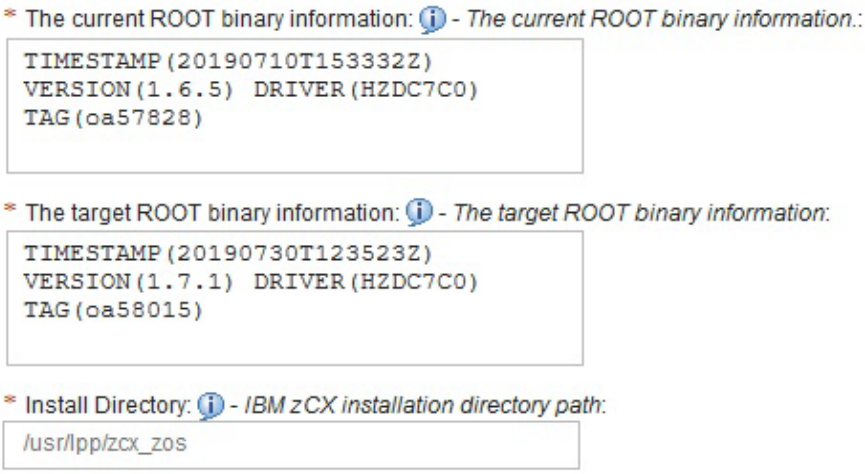


Figure 7-1 Upgrade workflow ROOT binary information

After Step 2, you can run the rest of the workflow with automation. The workflow halts as it reaches the steps to stop and restart the zCX instance. You must manually stop and restart the instance to finish the workflow.

When the instance comes up again, it displays messages about formatting of the root file system.

Example 7-1 zCX Startup messages after upgrade

```
GLZV002I zCX ZCXVL01 is formatting available space for use by
DSN=ZCX.REDB.ZCXVL01.ROOT2
GLZV003I zCX ZCXVL01 formatting complete for
DSN=ZCX.REDB.ZCXVL01.ROOT2
```

You can then issue a MODIFY command DISPLAY,DISKVER to verify that the current root version is the expected one that was outlined as target ROOT binary information in the workflow:

Example 7-2 zCX modify command DISPLAY,DISKVER

```
F ZCXVL01,DISPLAY,DISKVER
GLZC008I Disk Version information for zCX instance ZCXVL01
DevNo  Data Set Name                                  Version
   1   ZCX.REDB.ZCXVL01.ROOT2                         20190730T123523Z
         3.5.1           1.7.1           HZDC7C0         oa58015
   2   ZCX.REDB.ZCXVL01.CONF                          20190905T184238Z
   3   ZCX.REDB.ZCXVL01.SWAP1                         20190905T184152Z
   4   ZCX.REDB.ZCXVL01.DATA1                         20190905T184156Z
   5   ZCX.REDB.ZCXVL01.DLOG1                         20190905T184159Z
Total number of disks: 5
```

Rollback service

There is one more step at the end of the upgrade workflow, named **`On error run this step to restore backed up files`**. *This step is not meant to be run under normal circumstances*.

You would use this step to switch back to the backup version of the root file system in the following immediate circumstances only:

- The upgrade workflow had failed at some point. OR
- The instance does not come up with the new level.

In contrast, if problems arise later (after the workflow has already been finished) you can run the rollback workflow to restore the previous service level. The file path for the definition file of this workflow is as follows:

`/usr/lpp/zcx_zos/workflows/rollback.xml`

7.1.2 Maintenance for containers

Software that runs within the containers of zCX must be serviced separately from zCX itself. There are many sources for these containers, starting with images pulled from Docker Hub or the proprietary registries of any vendor and continuing up to containers built by your organization. For this reason, the procedures to install and activate service to the software of a container might vary.

Maintenance without Docker swarm

Applying maintenance to a container instance normally replaces the image that is used to start this container with a newer one.

The examples in this section show the upgrade from Grafana5.2.0-f2 to 5.2.0-f3. Grafana is a graphical data visualizer that we use to monitor our zCX instance in Chapter 7.6, “Configuring Grafana to monitor zCX containers” on page 166.

Notice that no persistent data is used for Grafana in Example 7-3 on page 144. This approach keeps the example simple and focused on the maintenance and not on setting up Grafana. If you have persistent data that is defined for your Grafana container, the **`docker run`** command reflects that. (For information on the use of persistent data, see Chapter 10, “Persistent data” on page 231.)

Example 7-3 Starting a Grafana container

```
VL1:/home/admin>docker run -d -p 3000:3000 --name grafana
ibmcom/grafana-s390x:5.2.0-f2
d5bd946914840f7d8618c594f8676540b6e96b2ffacdd9598c9f0a6980897b0f
```

To update this container to a newer level, you must pull the newer version of the Docker image from your registry, by using the tag that points to the correct version. In Example 7-4 on page 144, we upgrade our Grafana from the previously installed version 5.2.0-f2 to the new version 5.2.0-f3.

Example 7-4 Pull a new version of Grafana from the registry

```
VL1:/home/admin>docker pull ibmcom/grafana-s390x:5.2.0-f3
5.2.0-f3: Pulling from ibmcom/grafana-s390x
b766debbc269: Pull complete
6a13c128fd9c: Pull complete
d6662734a61f: Pull complete
b6c6781ca03e: Pull complete
638b1f753742: Pull complete
Digest: sha256:752c2a42cec21c88c1821cd7afa9a6e593a5646ad035893adb0e03da350d2a0d
Status: Downloaded newer image for ibmcom/grafana-s390x:5.2.0-f3
```

Having done this, you must stop the running container, remove it, and run a container with the newer image version as shown in Example 7-5 on page 145.

Example 7-5 Stopping Grafana and restarting with new image level

```
VL1:/home/admin>docker stop grafana
grafana

VL1:/home/admin>docker rm grafana
grafana

VL1:/home/admin>docker run -d -p 3000:3000 --name grafana
ibmcom/grafana-s390x:5.2.0-f3
08acc3fcd0f898f6b758072cdadcbcbdc0f364ddd91fb98a667c470b3caab2da
```

Any changes that have been done to the container are lost through the replacement. The changes must be done again to the new image.

Therefore, you should avoid applying customizations to a container within files of that container. If it is not possible or not desired to specify the needed configuration through parameters to the **docker run** command, consider the following approach:
Mount a Docker volume to that container to hold the configurations.
For more information on this approach, see Chapter 10, “Persistent data” on page 231.

Maintenance through the use of Docker swarm

If you run your Docker container as a Docker swarm service, you can use the swarm capabilities to upgrade your container to a new software level. The preceding point about changed files within the container also applies to maintenance, when you use Docker swarm.

For more details about setting up Docker swarm and a Docker swarm service see Chapter 11, “Swarm on zCX” on page 239. Example 7-6 uses the **docker service create** command to create a service that runs Grafana.

Example 7-6 Starting a Docker swarm service

```
VL1:/home/admin>docker service create --name grafana --replicas 1 -p 3000:3000
ibmcom/grafana-s390x:5.2.0-f2
xj4hbcpgcwp6hmoosirbdyrk5
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged

VL1:/home/admin>docker service ps grafana
ID              NAME             IMAGE                       NODE
DESIRED STATE       CURRENT STATE            ERROR               PORTS
mqqiyxmgt8sr        grafana.1           ibmcom/grafana-s390x:5.2.0-f2
sc74cn15.pbm.ihost.com   Running             Running 51 seconds ago
```

To update the level of a swarm service, you must first pull the image with the new level to the zCX instances where the service will be deployed as shown in the preceding section in Example 7-4 on page 144.

Then, you issue the **docker service update** command with the **--image** parameter. And you point to the new image level as shown in Example 7-7 on page 146.

Example 7-7 Update Docker service to new image level

```
VL1:/home/admin>docker service update --image ibmcom/grafana-s390x:5.2.0-f3
grafana
grafana
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged

VL1:/home/admin>docker service ps grafana
ID              NAME             IMAGE                      NODE
DESIRED STATE       CURRENT STATE                 ERROR               PORTS
8sqvw3k0mwt2        grafana.1           ibmcom/grafana-s390x:5.2.0-f3
sc74cn15.pbm.ihost.com   Running             Running about a minute ago
zy5tf3y49goz         \_ grafana.1       ibmcom/grafana-s390x:5.2.0-f2
sc74cn15.pbm.ihost.com   Shutdown            Shutdown about a minute ago
nqfqtgij1a2v         \_ grafana.1       ibmcom/grafana-s390x:5.2.0-f2
sc74cn15.pbm.ihost.com   Shutdown            Shutdown 2 minutes ago
mqqiyxmgt8sr         \_ grafana.1       ibmcom/grafana-s390x:5.2.0-f2
sc74cn15.pbm.ihost.com   Shutdown            Shutdown 6 minutes ago
```

If your swarm service consists of multiple instances, you have options in the **`docker service update`** command to control the sequence, parallelism, and delay for the update of the single instances:

Table 7-1 Options to control docker service update

option	meaning
`--update-delay duration`	Delay between updates (ns|us|ms|s|m|h)
`--update-failure-action string`	Action on update failure ("pause"|"continue"|"rollback")
`--update-max-failure-ratio float`	Failure rate to tolerate during an update
`--update-monitor duration`	Duration after each task update to monitor for failure (ns|us|ms|s|m|h)
`--update-order string`	Update order ("start-first"|"stop-first")
`--update-parallelism uint`	Maximum number of tasks updated simultaneously (0 to update all at once)

These options help you to ensure that only one instance is affected if an error occurs in the update process.

7.1.3 Building and maintaining your own image

In the prior examples, the Grafana image from Docker Hub was used directly. If you want to customize your Grafana image, you can easily build your own image based on the public image, and then add some configurations.

Creating the Dockerfile

The first step in building your own image is to create the Dockerfile within a separate directory. This directory must contain only the files that are used for the Docker image that you want to build.

Example 7-8 Creating a directory to build your own image

```
VL1:/home/admin>mkdir mygrafana
VL1:/home/admin>cd mygrafana
VL1:/home/admin/mygrafana>touch Dockerfile
VL1:/home/admin/mygrafana>vi Dockerfile
```

The last command of Example 7-8 starts the commonly used **vi** editor. For a first example, you can just put some environment variables into the file to control operation of Grafana. To enhance this example, we add a **read.me** file, which demonstrates how you can add file changes to a custom image. Example 7-9 shows an example of a Dockerfile to build a Grafana image, including an additional **read.me** file.

Example 7-9 Contents of Dockerfile for a first custom Grafana image

```
#Base Image is grafana
FROM ibmcom/grafana-s390x:5.2.0-f3
#Define Environment Variables
ENV GF_SERVER_ROOT_URL http://grafana.server.name
ENV GF_INSTALL_PLUGINS grafana-clock-panel,grafana-simple-json-datasource
#Insert a read.me file
COPY read.me /tmp/read.me
```

For the **docker build** command to succeed, the **read.me** file must exist in the **mygrafana** directory. Example 7-10 shows how to build the image.

Example 7-10 Building the custom Grafana image

```
VL1:/home/admin/mygrafana>ls -l
total 8
-rw-rw-r-- 1 admin admin 301 Sep 10 15:44 Dockerfile
-rw-rw-r-- 1 admin admin  96 Sep 10 15:53 read.me

VL1:/home/admin/mygrafana>docker build -t redb_grafana:1.1a .
Sending build context to Docker daemon  3.072kB
Step 1/4 : FROM ibmcom/grafana-s390x:5.2.0-f3
 ---> 2a9beae25fc2
Step 2/4 : ENV GF_SERVER_ROOT_URL http://grafana.server.name
 ---> Running in 345d40c0f526
Removing intermediate container 345d40c0f526
 ---> 8c1eef1e2f99
Step 3/4 : ENV GF_INSTALL_PLUGINS grafana-clock-panel,grafana-simple-json-datasource
 ---> Running in ec16249ded79
Removing intermediate container ec16249ded79
 ---> 873b60d2c00c
Step 4/4 : COPY read.me /tmp/read.me
 ---> 453eb921eb28
Successfully built 453eb921eb28
Successfully tagged redb_grafana:1.1a

VL1:/home/admin/mygrafana>docker images
REPOSITORY              TAG            IMAGE ID            CREATED              SIZE
redb_grafana            1.1a           f21a2eb2caf5        2 hours ago          942MB
ibm_zcx_zos_cli_image   latest         4f886cf032e4        4 days ago           363MB
```

To build an image with a newer level of Grafana, you must change the FROM statement of the Dockerfile to reflect the new version that you want to use. The same is true for changed

environment variables or changes in the included files. You can then do a new **docker build**, creating a new tag, such as **redb_grafana:1.2a**.

Distributing and deploying the image

After building a new image, you can start your container by using the new image as shown in Example 7-11.

Example 7-11 Starting a container with a new image

```
VL1:/home/admin>docker run -d -p 3000:3000 --name grafana redb_grafana:1.1a
065501676a8e7cdbad7468cfab8b746774a6d9828e7841c9d9e573b320a47414
```

Likewise, you can use this image to create a Docker service as described before.

To have this image available for different zCX instances, it must be tagged and sent to your registry as shown in Example 7-12. Each connected zCX can pull the image from that registry. (For information on setting up a private registry, see Chapter 6, “Private registry implementation” on page 107.)

Example 7-12 Pushing your image to the registry

```
VL1:/home/admin/mygrafana>docker tag redb_grafana:1.1a
sc74cn15.pbm.ihost.com:5000/redb_grafana:1.1a

VL1:/home/admin/mygrafana>docker push
sc74cn15.pbm.ihost.com:5000/redb_grafana:1.1a
The push refers to repository [sc74cn15.pbm.ihost.com:5000/redb_grafana]
7d8ef5f54238: Pushed
d4f91fc67624: Pushed
a467a22dc7e9: Pushed
a56b05807b81: Pushed
ce7cd584af89: Pushed
cd2bab4d3640: Pushed
1.1a: digest:
sha256:3ea5907c8a5f6ea8e78f6d51124a3813f8ece2e3bfc66996692fbdb1f5875ec7 size: 1790
```

If you have not set up a registry to which to push the image, you can still make the image available in a different zCX. To do so, you save the image to a tar file as shown in Example 7-13, transfer it to the target zCX, and load it there.

Example 7-13 Save your image to a tar file and send to a different zCX instance

```
VL1:/home/admin>docker save -o /tmp/rbg11a.tar redb_grafana:1.1a

VL1:/home/admin>scp -P 8022 /tmp/rbg11a.tar
volkmar@sc74cn16.pbm.ihost.com:/tmp/rbg11a.tar
volkmar@sc74cn16.pbm.ihost.com's password:
rbg11a.tar
```

To use **scp** in this way, you first create a user on the receiving zCX by using the **adduser** command.

On the target zCX instance, you can then load the image by using the **docker load** command as shown in Example 7-14 on page 149.

Example 7-14 Load your image from a tar file

```
VL2:/home/admin>docker load -i /tmp/rbg11a.tar
cd2bab4d3640: Loading layer [==================================================>]  240.3MB/240.3MB
ce7cd584af89: Loading layer [==================================================>]  114.6MB/114.6MB
a56b05807b81: Loading layer [==================================================>]  602.1MB/602.1MB
a467a22dc7e9: Loading layer [==================================================>]  4.608kB/4.608kB
d4f91fc67624: Loading layer [==================================================>]  37.04MB/37.04MB
a16efe2c6d33: Loading layer [==================================================>]   2.56kB/2.56kB
Loaded image: redb_grafana:1.1a
```

7.2 Automation

This section shows the required information for automating the zCX started task through some z/OS automation product like System Automation for z/OS.

Also, this section shows methods to automate the container workload that runs within a zCX. This topic lies outside the scope of z/OS automation.

7.2.1 Automating zCX instances

If you want to automate the operation of zCX, you must define the dependencies of zCX. You must also define dependencies for start and stop commands and the messages that indicate a successful start or stop to your automation product.

Starting zCX

zCX needs to read the **start.json** from the zFS file system, and it must connect to the TCPIP stack. So, these dependencies must be defined for starting zCX. OMVS and ZFS must be up and running, as does TCPIP.

To start the zCX instance, you use the start command that is provided by the provisioning workflow and which looks similar to Example 7-15:

Example 7-15 Starting a zCX instance

```
S GLZ,JOBNAME=ZCXVL01,CONF='/global/zcx/instances/ZCXVL01/start.json'
```

The instance has finished startup and is ready to work when the following message appears:

GLZB001I zCX instance ZCXVL01 initialization is complete. Code date 11/08/19.

Stopping zCX

To stop a zCX instance, you use the stop command that is provided by the provisioning workflow. Example 7-16 show a typical stop command.

Example 7-16 Stopping a zCX instance

```
P ZCXVL01
```

When zCX has successfully stopped, it issues the following message:

GLZB002I zCX instance ZCXVL01 has ended.

The shutdown of a zCX might take some minutes to complete. Therefore, automation should wait at least 3 minutes before it takes more aggressive actions to stop the instance.

For a shutdown, zCX signals the containers that are running inside zCX to also stop and gives them 10 seconds to stop, before they get the SIGTERM signal.

If zCX does not gracefully come to an end, you have the option to force it using the command as shown in Example 7-17 on page 150:

Example 7-17 Forcing a zCX instance

```
FORCE ZCXVL01,ARM
```

You know that the zCX instance is down in this case, when the following message appears:

```
IEF450I ZCXVL01 ZCXVL01 - ABEND=SA22 U0000 REASON=00000000 TIME=16.59.02
```

Moving zCX to a different system

Instances of zCX are not bound to a specific system. You can start them on any system that has these characteristics:

- Runs within the same sysplex.
- Fulfills the prerequisites to run zCX.
- Has access to the VSAM linear data sets of the zCX instance.
- Has access to the folder in the UNIX file system that contains the **start.json** for this instance.

When you move zCX instances from one system to another, be sure that this system has enough free memory to handle the additional fixed memory requirements for the zCX instance.

7.2.2 Automating containers

Using the --restart option of docker run

Docker gives you the ability to get containers started automatically after zCX completes the startup and after a failure. To achieve this, you use the **`--restart`** option of the **`docker run`** command as shown in Example 7-18 on page 150.

Example 7-18 Using the --restart option on docker run command

```
VL1:/home/admin>docker run -d -p 3000:3000 --name grafana --restart always
redb_grafana:1.1a
418c8df46ec61c58b6137f5fda381b1822b9457ac259748ee92620a1a398bbe5
```

The keywords for the **`--restart`** option are shown in Table 7-2.

Table 7-2 Keywords for --restart option of docker run

Keyword	Meaning
no	No automated start of the container.
on-failure[:tries]	Restart the container after a failure. Try this for a maximum of the specified amount of tries or infinite, if no tries are specified.

Keyword	Meaning
unless-stopped	Restart the container after a failure and in addition at startup of zCX, if it was not in a stopped state when zCX was stopped for the last time.
always	Restart the container after a failure and in addition at startup of zCX, regardless of the container‘s state when zCX stopped for the last time.

Docker delays the restart of a failing container by 100ms. And Docker doubles this delay for each time the container fails. The delay is reset to 100ms when the container stays up for at least 10 seconds after a restart.

Using Docker swarm

With Docker swarm, you define a service that is running your container. Multiple instances of the container can run in parallel. Also, you can have multiple instances of zCX — even spread over different LPARs — to run your container instances.

To use Docker swarm, you first must set up the swarm, as described in Chapter 11, “Swarm on zCX” on page 239. When the swarm is set up, you can run a service in that swarm with the **docker service create** command as shown in Example 7-19.

Example 7-19 Create a docker service

```
VL1:/home/admin>docker service create -p 3000:3000 --name grafana redb_grafana:1.1a
image redb_grafana:1.1a could not be accessed on a registry to record
its digest. Each node will access redb_grafana:1.1a independently,
possibly leading to different nodes running different
versions of the image.

eg8w5ppyt2d1mmqj046w1bpu5
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged
```

The preceding sample used a Docker swarm that is spread over two zCX instances with no private registry setup. For this reason, swarm issues a message that warns us that separate images are accessed on the different nodes of the swarm.

If your container uses Docker volumes that are mounted to it, spreading your service over multiple zCX instances does not work properly, because volumes cannot be shared across multiple zCX instances.

7.3 Backup and recovery

Table 7-3 describes the two approaches for backing up the data for your zCX environment.

Table 7-3 Options for backing up zCX data

Option	Details
Back up at the instance level.	You easily manage backup and recovery for a whole instance, but cannot recover single containers within that instance.

Option	Details
Back up at the container level.	This is a more complex way to back up the containers and their data: ► You still must do the instance-level backup, in case you must recover zCX itself. ► But to recover single containers without affecting the other containers that run in this zCX instance, you must have additional backups.

7.3.1 Backup and recovery on zCX instance level

The instance-level backup is a very convenient way to back up and recover a full zCX instance with all containers that run inside it. It is especially suitable to handle a zCX instance that matches one of these descriptions:

- Only one container runs in it. OR
- All containers that run in it belong to a single application.

Data belonging to a zCX instance

All data that makes up a zCX instance lives within the VSAM linear data sets that have been defined by the provisioning or the add-volume workflows. These VSAM linear data sets are the ones whose name starts with the high-level qualifier for this zCX. This qualifier is set during the provisioning workflow. Example 7-20 shows a list of these data sets.

Example 7-20 VSAM linear data sets that belong to a zCX instance

```
DSLIST - Data Sets Matching ZCX.REDB.ZCXVL01
Command ===>

Command - Enter "/" to select action
-------------------------------------------------------
         ZCX.REDB.ZCXVL01.CONF
         ZCX.REDB.ZCXVL01.CONF.DATA
         ZCX.REDB.ZCXVL01.DATA1
         ZCX.REDB.ZCXVL01.DATA1.DATA
         ZCX.REDB.ZCXVL01.DLOG1
         ZCX.REDB.ZCXVL01.DLOG1.DATA
         ZCX.REDB.ZCXVL01.ROOT
         ZCX.REDB.ZCXVL01.ROOT.DATA
         ZCX.REDB.ZCXVL01.SWAP1
         ZCX.REDB.ZCXVL01.SWAP1.DATA
         ZCX.REDB.ZCXVL01.ZFS
         ZCX.REDB.ZCXVL01.ZFS.DATA
***************************** End of Data Set list
```

In addition, the zCX instance must know the directory locations of the following components:

- The UNIX file system where the **start.json** for this instance is stored.
- The properties file of the instance.

Except for the **start.json** file that is needed for the start of zCX, these files are not used at run time but rather for the z/OSMF workflows to maintain the zCX instance. Example 7-21 shows a list of these files.

Example 7-21 UNIX files and directories belonging to a zCX instance

```
Pathname . : /global/zcx/instances/ZCXVL01
EUID . . . : 311
Command  Filename
-------------------------------------------
         .
         ..
         config
```

```
         config.bkup
         start.json
         start.json.bkup
         tmp
         FFDC
         ZCX-appliance-version
         ZCX-ZCXVL01-user.properties
         ZCX-ZCXVL01-user.properties.bkup

Pathname . : /global/zcx/cfg/properties
 EUID . . . : 311
 Command  Filename
 ---------------------------------------------
          ZCXVL01-user.properties
```

Be aware that there are techniques to attach external data to an application, for example, by using NFS or a database that is located outside of this zCX instance. This external data is not covered by the backup and recovery processes that are described here.

Backing up the data of a zCX instance

DFSMShsm is not able to back up all of the VSAM linear data sets that belong to your zCX instance while the instance is running, because the data sets are opened by zCX.

You can stop the instance to do the HSM backups, and then restart it. If this is not an option, you can backup the VSAM linear data set by using DFSMSdss as is shown in Example 7-22.

Example 7-22 Using DSS to manually backup the VSAM linear data sets for a zCX instance

```
//ZCXPRV8D JOB (ACCNT),LANGER,CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1),
//             REGION=32M
//BACKUP  EXEC PGM=ADRDSSU
//BACKUP   DD DISP=(,CATLG),DSN=ZCX.REDB.BACKUP.ZCXVL01,
//             SPACE=(CYL,(5000,5000)),DSNTYPE=LARGE
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
 DUMP DS(INCLUDE(ZCX.REDB.ZCXVL01.*)) -
      OUTDDNAME(BACKUP) TOL(ENQF) CC(PREFERRED) ZCOMPRESS(PREFERRED)
```

You can include the zFS data set that belongs to this zCX instance in this backup, too. But for this zFS, you also have the option of doing the backup by using your normal DFSMShsm procedures.

Be aware that the properties file of the instance might be located in a directory outside of the zFS that you are backing up. In this case, you must backup the properties file separately.

As of z/OS 2.4, you can use DFSMShsm to backup the UNIX files. The UNIX files don‘t contain data for the containers. Instead, they contain the configuration data for the z/OSMF workflows and the start script for zCX. There are no changes to these files through the zCX instance itself, except for the FFDC directory, where first failure capture data is written by zCX for specific error events. Example 7-23 demonstrates the use of DFSMShsm to back up the UNIX files.

Example 7-23 Using DFSMShsm to back up the UNIX files for a zCX instance

```
//ZCXPRV8L JOB (ACCNT),LANGER,CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//USS     EXEC PGM=BPXBATCH
//STDOUT    DD SYSOUT=*
//STDERR    DD SYSOUT=*
```

```
//STDIN     DD DUMMY
//STDPARM   DD *
SH
hbackup '/global/zcx/instances/ZCXVL01/*';
hbackup '/global/zcx/cfg/properties/ZCXVL01-user.properties';
```

Recover a zCX instance

To recover a zCX instance, you must recover all the VSAM linear data sets of that instance. The instance must be stopped for a recovery.

You then recover the VSAM linear data sets from a backup that was created with DFSMSdss as shown in Example 7-24 on page 154. The backup of the VSAM linear data sets also included the zFS that contains the configuration information of the zCX instance. You must unmount this zFS before doing the recovery, if it is still mounted. Also, after recovery, you must mount the zFS again.

If the zFS of the instance is not part of the backup, you can skip the UNMOUNT and REMOUNT steps.

Example 7-24 Using DFSMSdss to recover the VSAM linear data sets of a zCX instance

```
//ZCXPRV8R JOB (ACCNT),LANGER,CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1),
//             REGION=32M
//UNMOUNT EXEC PGM=IKJEFT01
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD *
 UNMOUNT FILESYSTEM('ZCX.REDB.ZCXVL01.ZFS')
//RECOVER EXEC PGM=ADRDSSU
//BACKUP    DD DISP=SHR,DSN=ZCX.REDB.BACKUP.ZCXVL01
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD *
 RESTORE DS(INCLUDE(**)) INDD(BACKUP)
//REMOUNT EXEC PGM=IKJEFT01
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD *
 MOUNT FILESYSTEM('ZCX.REDB.ZCXVL01.ZFS') TYPE(ZFS) MODE(RDWR) -
       MOUNTPOINT('/global/zcx/instances/ZCXVL01') -
       AUTOMOVE
```

Attention: Be aware that the backups were done on open VSAM linear data sets. When recovering the instance from these backups, there might be additional actions necessary to recover some of the containers data inside of the instance to a consistent point in time.

If the instance‘s zFS was not part of the restored data sets, you must recover those files separately. Again, be aware that the properties file of the instance might be located outside of the zFS that was backed up for this instance.

As of z/OS 2.4, you can use DFSMShsm to recover UNIX files on a file level as shown in Example 7-25.

Example 7-25 Using DFSMShsm to recover the UNIX files of a zCX instance

```
//ZCXPRV8R JOB (ACCNT),LANGER,CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//RECOVER EXEC PGM=IKJEFT01
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD *
```

```
HRECOVER '/global/zcx/instances/ZCXVL01/*' REPLACE
HRECOVER '/global/zcx/cfg/properties/ZCXVL01-user.properties' REPLACE
```

After this recovery has finished, you can start the zCX instance again.

7.3.2 Backup and recovery on container level

Backup and recovery at the container level is a more complex approach, and it might vary depending on the type of application and the type of data that is used by the application.

You also must make sure that the image of the container itself is backed up. Normally, the image should also exist in your registry. If for some reason, you have images that are not pushed to a registry, you can save the image to an external file by using the **docker save** command as shown in Example 7-13 on page 148.

If the container application provides its own methods for backing up the application‘s data, it is preferable that you use these methods for backup and recovery.

In other cases, you have various options for backing up container data depending on where it resides.

Backing up application data that resides in the container file system

You can use the **docker cp** command to copy data between the container file system and the zCX file system. In this way, you can back up application data into a backup directory. Then, in a second step, you tar these backed-up files. Example 7-26 shows how to do this kind of backup.

*Example 7-26 Backup container file system data by using **docker cp** and **tar***

```
VL1:/home/admin>mkdir smallapp.backup
VL1:/home/admin>docker cp smallapp:/var/appdata ./smallapp.backup/appdata
VL1:/home/admin>cd smallapp.backup
VL1:/home/admin/smallapp.backup>tar -cf backup.tar appdata
VL1:/home/admin/smallapp.backup>tar -tvf backup.tar
drwxr-xr-x admin/admin       0 2019-09-11 15:01 appdata/
-rw-r--r-- admin/admin   35330 2019-09-11 15:01 appdata/more.data
-rw-r--r-- admin/admin  184214 2019-09-11 15:01 appdata/smallapp.log
-rw-r--r-- admin/admin       9 2019-09-11 14:59 appdata/smallapp.pid
-rw-r--r-- admin/admin   10550 2019-09-11 15:00 appdata/userdata.db
```

When you must restore this data, you unpack the tar file and use the **docker cp** command to copy the data back into the container again as shown in Example 7-27.

Example 7-27 Recover container file system data by using **tar** and **docker cp**

```
VL1:/home/admin/smallapp.backup>tar -xf backup.tar
VL1:/home/admin/smallapp.backup>docker cp appdata smallapp:/var
```

Backing up application data that resides in a Docker volume

Instead of having the data inside your container, you can define a Docker volume and mount that to the containers directory that is to contain the data, as shown in Example 7-28.

Example 7-28 Using docker volume as persistent storage

```
VL1:/home/admin>docker volume create --name smallapp-appdata
smallapp-appdata
VL1:/home/admin>docker run -it --name smallapp -v smallapp-appdata:/var/appdata ubuntu
```

The advantage of using a Docker volume is the persistency of the data:

- **Without a Docker volume:** If for any reason the container is stopped and removed, data that you store in the file system of the container itself is lost.
- **With a Docker volume:** If you use a volume, the data resides outside the container and survives the stopping and removal of the container.

Backing up and recovering the data can be done in the same way as described in the preceding section.

7.4 Diagnosis

This section describes ways to discover problems in the scope of zCX and to identify the component that might be the source of the problem. The second part of this section describes the procedures to get problem determination data.

7.4.1 Ways to check healthiness of zCX

If you suspect something might be wrong with your zCX instance, there are some places to look first for possible causes.

SSH into zCX

Check whether you are still able to log on to the zCX by using ssh. If you are not able to ssh into the zCX, you must search for symptoms from the outside. In this case, you can jump to “Checking the zCX joblog” on page 158.

Checking TOP command

The UNIX **top** command shows some memory and cpu statistics and usage information per process. Figure 7-2 on page 156 shows a sample output of this command.

```
top - 17:52:37 up 20:45,  1 user,  load average: 4.00, 3.63, 2.13
Tasks:   8 total,   1 running,   5 sleeping,   0 stopped,   2 zombie
%Cpu(s): 98.3 us,  1.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.3 st
KiB Mem :  1756844 total,    12244 free,  1510156 used,   234444 buff/cache
KiB Swap:  2041852 total,  1486332 free,   555520 used.   211984 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
    1 root      20   0   20804    156    156 S  0.0  0.0   0:00.01 sudo
    6 root      20   0    9904    164    164 S  0.0  0.0   0:00.00 sshd
   13 admin     20   0       0      0      0 Z  0.0  0.0   0:00.00 bash
  154 root      20   0   22948    196    196 S  0.0  0.0   0:00.02 sshd
  159 admin     20   0   22948    492    356 S  0.0  0.0   0:00.37 sshd
  160 admin     20   0    4092   1776   1532 S  0.0  0.1   0:00.12 bash
  209 admin     20   0       0      0      0 Z  0.0  0.0   0:00.01 ssh
  716 admin     20   0    5312   2944   2488 R  0.0  0.2   0:00.19 top
```

Figure 7-2 output of top command

The top lines about %Cpu, Memory, and Swap show values for the whole zCX instance, not for the specific container that you are in. In contrast, the process list shows the processes of the container that you are currently working in.

> **Note:** You are already within the **ibm_zcx_zos_cli** container when you log in to zCX. The **ibm_zcx_zos_cli** container provides the shell functions for zCX.

The **top** command updates its display periodically. To stop the display, press **control-C**.

In the ssh CLI where you are logged on, you can enter the **top** command to get a first view of the zCX instance. You can then enter the **top** command for the different containers by using the **`docker exec`** command as shown in Example 7-29. Most containers should be able to run the command. There might be special containers for which the command is not implemented.

Example 7-29 Execute top command within a container

```
docker exec -it registry top
```

Eventually, you can identify a misbehaving process on this layer. Give special attention to the systems memory, and also swap usage as to the %CPU usage on the instance level. And of course, you must monitor the %CPU and %MEM values of the processes within the different containers.

Checking Docker stats

If your checks in the **top** display do not give any hints about a problem situation, you can enter a similar command for the Docker containers that are running. This is the command **`docker stats`**, which shows output that is similar to what you see in Example 30.

Example 30 Docker stats output

```
CONTAINER ID   NAME            CPU %   MEM USAGE / LIMIT     MEM %   NET I/O           BLOCK I/O         PIDS
52869f058a54   grafana         0.01%   20.93MiB / 1.675GiB   1.22%   4.34MB / 4.35MB   67.3MB / 200MB    17
6134d99615c6   prometheus      0.71%   23.45MiB / 1.675GiB   1.37%   2.44MB / 1.15MB   5.98MB / 8.19kB   16
2475f664e9aa   cadvisor        9.79%   59.66MiB / 1.675GiB   3.48%   98.4MB / 1.71GB   104MB / 84MB      20
ddabf91531ac   nodeexporter    0.00%   3.492MiB / 1.675GiB   0.20%   1.01kB / 0B       12MB / 0B         6
a0fb146c85ae   registry        0.00%   5.504MiB / 1.675GiB   0.32%   3.74kB / 0B       111MB / 0B        13
cae0df807ac6   ibm_zcx_zos_cli 0.76%   27.42MiB / 1.675GiB   1.60%   534kB / 761kB     528MB / 4.1MB     19
```

This command shows you how the different containers behave in terms of cpu, memory, network, and disk activity. If you see unexpected numbers, you might dive deeper into the respective container activities.

Checking Docker logs

The **`docker logs`** command allows you to view the log of a container, as shown in Example 31 on page 158.

Example 31 Using docker logs command

```
VL2:/home/admin>docker logs --tail 10 grafana
t=2019-09-11T19:41:47+0000 lvl=info msg="Initializing InternalMetricsService" logger=server
t=2019-09-11T19:41:47+0000 lvl=info msg="Initializing AlertingService" logger=server
t=2019-09-11T19:41:47+0000 lvl=info msg="Initializing HTTPServer" logger=server
t=2019-09-11T19:41:47+0000 lvl=info msg="Initializing CleanUpService" logger=server
t=2019-09-11T19:41:47+0000 lvl=info msg="Initializing NotificationService" logger=server
t=2019-09-11T19:41:47+0000 lvl=info msg="Initializing ProvisioningService" logger=server
t=2019-09-11T19:41:47+0000 lvl=info msg="Initializing RenderingService" logger=server
t=2019-09-11T19:41:47+0000 lvl=info msg="Initializing TracingService" logger=server
t=2019-09-11T19:41:47+0000 lvl=info msg="Initializing Stream Manager"
t=2019-09-11T19:41:47+0000 lvl=info msg="HTTP Server Listen" logger=http.server address=0.0.0.0:3000
protocol=http subUrl= socket=
```

This information might reveal more details about problems that the container might have experienced. You can limit the output of the command by using the options in Table 7-4 on page 158.

Table 7-4 Options for the docker logs command

Option	Meaning
--tail amount	Shows the given number of log lines from the end of the log. Default is **all**.
--since when	Starts showing the log from a given point in time, either in the format yyyy-mm-ddThh:mm:ss or in relative format like 30m for 30 minutes.
--until when	Stops showing the log at the given point in time.

Checking the zCX joblog

The zCX started task writes its log information to SYSPRINT. For example, if there is an error in network communication, you might find it in this log. You might also see messages that show memory constraints here. You can find the SYSPRINT log by viewing the output of your zCX task while you use a spool display application like SDSF. You should also review the joblog for the zCX started task.

Checking CPU consumption of zCX

You can use monitoring tools like RMF to look at the CPU consumption of zCX on a started-task level. It might indicate whether the task is affected by other work within the system. For more information on using RMF, see Section 7.5, “Monitoring with RMF on zCX instance level” on page 160.

Checking system health

In RMF, you can also check for memory constraints in the system. At this time, checking the memory consumption of your zCX task itself has little troubleshooting value, because zCX allocates and fixes all its memory at startup time. Nonetheless, checking the system’s overall memory usage might be helpful. For example, if your system uses most of its memory before zCX gets into the system, starting a zCX instance might constrain the system further. In turn, that affects the performance of zCX.

Besides memory, you can check other metrics to confirm system health, such as CPU usage, zIIP usage, DASD response times, and network response times.

7.4.2 Gathering problem data for zCX

In case you must gather diagnosis data for deeper problem analysis, the following steps might help you to collect the needed diagnosis.

MODIFY instance,DISPLAY,DISKVER

Issue this system console command to get information about the currently active root version of the zCX instance.

Example 7-32 MODIFY instance,DISPLAY,DISKVER

```
F ZCXVL01,DISPLAY,DISKVER
GLZC008I Disk Version information for zCX instance ZCXVL01
DevNo  Data Set Name                                  Version
   1   ZCX.REDB.ZCXVL01.ROOT                          20190730T123523Z
         3.5.1          1.7.1          HZDC7C0          oa58015
   2   ZCX.REDB.ZCXVL01.CONF                          20190911T181812Z
   3   ZCX.REDB.ZCXVL01.SWAP1                         20190905T174439Z
   4   ZCX.REDB.ZCXVL01.DATA1                         20190905T174442Z
   5   ZCX.REDB.ZCXVL01.DLOG1                         20190905T174446Z
Total number of disks: 5
```

MODIFY instance,DISPLAY,CONFIG

This system console command shows some information about the allocated memory and CPU, and also the location of the **start.json** for this zCX instance.

Example 7-33 MODIFY instance,DISPLAY,CONFIG

```
F ZCXVL01,DISPLAY,CONFIG
GLZC003I Configuration information for zCX instance ZCXVL01
File Path: /global/zcx/instances/ZCXVL01/start.json
FFDC Path: /global/zcx/instances/ZCXVL01/FFDC
Dump Path: /global/zcx/instances/ZCXVL01/FFDC/zcx-guest.dmp
Memory size:           2GB
Number of CPUs:        4
Number of Disks:       5
Number of Networks:    1
CTRACE Parmlib Member: CTIGLZ00
```

MODIFY instance,DISPLAY,DISK

This system console command shows the sizes of all file systems (backed by VSAM linear data sets) that are allocated to the zCX instance.

Example 7-34 MODIFY instance,DISPLAY,DISK

```
F ZCXVL01,DISPLAY,DISK
GLZC004I Disk information for zCX instance ZCXVL01
DevNo     Size  Encrypted? Data set Name
   1       4GB     No      ZCX.REDB.ZCXVL01.ROOT
   2       3MB     No      ZCX.REDB.ZCXVL01.CONF
   3       2GB     No      ZCX.REDB.ZCXVL01.SWAP1
   4      20GB     No      ZCX.REDB.ZCXVL01.DATA1
   5    1001MB     No      ZCX.REDB.ZCXVL01.DLOG1
Total number of disks: 5
```

MODIFY instance,DISPLAY,NET

This system console command shows the configured IP address and MTU size for the zCX instance.

Example 7-35 MODIFY instance,DISPLAY,NET

```
F ZCXVL01,DISPLAY,NET
GLZC005I Network information for zCX instance ZCXVL01
```

```
DevNo  Stack     MTU    IP Address
   0   TCPIP     1492   129.40.23.82
Total number of networks: 1
```

MODIFY instance,DUMP,GUEST

This system console command writes a dump of the Linux guest of zCX to the instances directory in the z/OS UNIX file system.

Example 7-36 MODIFY instance,DUMP,GUEST

```
F ZCXVL01,DUMP,GUEST
GLZC010I A Dump Guest command for zCX instance ZCXVL01 has been accepted.
GLZC011I A complete dump of the guest memory for zCX
instance ZCXVL01 has been written to file
/global/zcx/instances/ZCXVL01/FFDC/zcx-guest.dmp
```

SVC dump

If the zCX task suffered from an abend and has written an SVC dump, this might also be a valuable source for problem diagnosis. If there is no dump available and you need a dump of the started task, you can request one by running the following command:

Example 7-37 Requesting an SVC dump

```
DUMP COMM='meaningful title for dump'
020 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND

20,JOBNAME=ZCXVL01,END
IEE600I REPLY TO 020 IS;JOBNAME=ZCXVL01,END
IEA794I SVC DUMP HAS CAPTURED:
DUMPID=007 REQUESTED BY JOB (*MASTER*)
DUMP TITLE=meaningful title for dump
IEA611I COMPLETE DUMP ON DUMP.D190912.H19.SC74.#MASTER#.S00007
DUMPID=007 REQUESTED BY JOB (*MASTER*)
FOR ASIDS(0001,00B7)
INCIDENT TOKEN: PLEX75   SC74     09/12/2019 19:36:03
```

The IEA611I message tells you which data set your dump has been written to.

Joblog and Syslog

You should save the joblog of the zCX instance that you want to diagnose. Also, you should extract the syslog from the time that the problem occurred.

7.5 Monitoring with RMF on zCX instance level

Monitoring of the zCX environment consists of two areas. On the one hand, you can monitor zCX as a started task within z/OS, and on the other hand you can monitor the containers that run within that zCX instance.

The main metrics that you might want to monitor from the outside of zCX are the CPU consumption — CP and also zIIP — and the disk I/O rate and response times. The memory is not something that you would monitor for the zCX started task, because the whole memory of the task is fixed in real memory when zCX starts.

If RMF is available on your system, you can access it from the z/OS system programmer applications panel by using option 12 of the ISPF Primary Options Panel, which is shown in Example 7-38.

Example 7-38 ISPF Primary Options Panel

```
Menu  Utilities  Compilers  Options  Status  Help
--------------------------------------------------------------------------
                          ISPF Primary Option Menu
Option ===> 12

0  Settings      Terminal and user parameters          User ID . : ZCXPRV8
1  View          Display source data or listings       Time. . . : 04:18
2  Edit          Create or change source data          Terminal. : 3278
3  Utilities     Perform utility functions             Screen. . : 1
4  Foreground    Interactive language processing       Language. : ENGLISH
5  Batch         Submit job for language processing    Appl ID . : PDF
6  Command       Enter TSO or Workstation commands     TSO logon : IKJACCT
7  Dialog Test   Perform dialog testing                TSO prefix: ZCXPRV8
9  IBM Products  IBM program development products      System ID : SC74
10 SCLM          SW Configuration Library Manager      MVS acct. : ACCNT#
11 Workplace     ISPF Object/Action Workplace          Release . : ISPF 7.4
12 z/OS System   z/OS system programmer applications
13 z/OS User     z/OS user applications

     Enter X to Terminate using log/list defaults
```

After you navigate to the z/OS system programmer applications panel, select option 9, which is shown in Example 7-39 on page 161.

Example 7-39 ISPF z/OS System Programmer Primary Option Menu

```
z/OS System Programmer Primary Option Menu
Option ===>   9

1  GDDM PQM      GDDM Print Queue Manager
2  HCD           HCD I/O configuration
5  APPC Admin    APPC Administration Dialog
6  WLM           Work Load Manager
7  FFST          FFST dump formatting
8  Infoprint Srv Infoprint Server
9  RMF           RMF
10 SMP/E         SMP/E
11 TCP/IP NPF    TCP/IP NPF
```

Within RMF, we focus on the Monitor III, which can be selected with option 3 from within RMF. You can access complete documentation for the available reports within RMF through the following IBM Knowledge Center link:
https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.4.0/com.ibm.zos.v2r4.erbb500/abstract.htm

Example 7-40 RMF Primary Panel

```
RMF - Performance Management                    z/OS V2R4 RMF
Selection ===> 3

Enter selection number or command on selection line.


  1 Postprocessor   Postprocessor reports for Monitor I, II, and III    (PP)
```

```
 2 Monitor II      Snapshot reporting with Monitor II                  (M2)
 3 Monitor III     Interactive performance analysis with Monitor III   (M3)

 U USER            User-written applications (add your own ...)        (US)

 R RMF SR         Performance analysis with the Spreadsheet Reporter
 N News            What's new in z/OS V2R4 RMF

                           T TUTORIAL    X EXIT

 RMF Home Page:   http://www.ibm.com/systems/z/os/zos/features/rmf/

            5650-ZOS Copyright IBM Corp. 1994, 2019.
              Licensed Materials - Property of IBM
```

Example 7-41 shows the primary menu for RMF Monitor III.

Example 7-41 RMF Monitor III main menu

```
RMF Monitor III Primary Menu               z/OS V2R4 RMF
Selection ===>

Enter selection number or command on selection line.


 S SYSPLEX         Sysplex reports and Data Index                      (SP)
 1 OVERVIEW        WFEX, SYSINFO, and Detail reports                   (OV)
 2 JOBS            All information about job delays                    (JS)
 3 RESOURCE        Processor, Device, Enqueue, and Storage             (RS)
 4 SUBS            Subsystem information for HSM, JES, and XCF        (SUB)

 U USER            User-written reports (add your own ...)             (US)


                  O OPTIONS    T TUTORIAL    X EXIT

            5650-ZOS Copyright IBM Corp. 1986, 2019.
              Licensed Materials - Property of IBM
```

7.5.1 RMF overview display

First, select the **Overview** menu and then **Job Usage** as shown in the following examples.

Example 7-42 RMF Overview Report Selection Menu

```
RMF Overview Report Selection Menu
Selection ===>

Enter selection number or command for desired report.

 Basic Reports
         1 WFEX     Workflow/Exceptions                          (WE)
         2 SYSINFO  System information                            (SI)
         3 CPC      CPC capacity

 Detail Reports
         4 DELAY    Delays                                      (DLY)
        4A USAGE    Job Usage                                   (USG)
         5 GROUP    Group response time breakdown                (RT)
```

```
  6 ENCLAVE  Enclave resource consumption and delays    (ENCL)
  7 OPD      OMVS process data
 10 SPACEG   Storage space                               (SPG)
 11 SPACED   Disk space                                  (SPD)
 12 LOCKSP   Spin locks                                  (LSP)
 13 LOCKSU   Suspend locks                               (LSU)
```

Example 7-43 RMF Job Usage

```
RMF V2R4   Job Oriented Usage              Line 1 of 143
Command ===>                                              Scroll ===>
CSR

Samples: 100     System: SC74  Date: 09/15/19  Time: 20.15.00  Range: 100 Se

           Service  --- I/O ---  --- CPU ---  - Storage -  ----- QScan ----
Jobname  CX Class    Conn   EXCP  Total   TCB  Total Fixed  Total Resct Time

ZCXED01  SO SYSSTC   197.0  0.00  40.47 40.10   544K  527K      0 0.000 0
ZCXME02  SO SYSSTC   176.5  0.00  45.56 45.27   545K  527K      0 0.000 0
ZCXJN04  SO SYSSTC   169.9  0.00  45.32 44.97   544K  527K      0 0.000 0
ZCXZB01  SO SYSSTC   169.3  0.00  43.89 43.47   545K  527K      0 0.000 0
ZCXRJ02  SO SYSSTC   150.4  0.00  45.80 45.50   545K  527K      0 0.000 0
ZCXEM01  SO SYSSTC   135.0  0.07  46.44 46.06   544K  527K      0 0.000 0
ZCXZB02  SO SYSSTC   130.3  0.00  48.57 48.36   545K  527K      0 0.000 0
ZCXME01  SO SYSSTC   123.1  0.00  57.56 57.01  1072K 1053K      0 0.000 0
ZCXRJ01  SO SYSSTC   73.65  0.00  81.05 80.80   545K  527K      0 0.000 0
ZCXSM01  SO SYSSTC   62.87  0.04  44.92 44.53   545K  527K      0 0.000 0
XCFAS    S  SYSTEM   0.366  5.64   0.01  0.01   7822  2129      0 0.000 0
ZCXPRV8  T  TSO1     0.347  5.72   0.04  0.04    161    23      1 0.000 241
ZCXVL01  SO SYSSTC   0.209  0.19   2.84  2.84   544K  527K      0 0.000 0
RMFGAT   SO SYSSTC   0.124  0.08   0.16  0.16  29138   255      0 0.000 0
OMVS     S  SYSTEM   0.032  3.48   0.11  0.11   641K  5895      0 0.000 0
CATALOG  S  SYSTEM   0.021  0.60   0.01  0.01   4579   698      0 0.000 0
```

As you can see in Example 7-43, RMF shows information about disk I/O, CPU usage, memory usage, and GRS Queue Scans for each task in the system. You can use the F10 and F11 keys to scroll through the time intervals. This scrolling action is available in all of the RMF panels that show a time range in the heading.

The I/O column shows the cumulative time in seconds within the interval that the task was connected to a device. You can also see the average number of EXCPs (Execute Channel Programs) that were executed per second within the interval.

The CPU column shows the time in seconds that the task was using a processor in the interval. The first value is the sum of all processor usage. The second value shows only the processor time that was used by the problematic program itself. This display does not distinguish between zIIP and CP processors.

The storage column displays the number of memory frames (4K) that were allocated to the task and the amount of these frames that are fixed to main memory. For zCX, almost all memory is fixed, all the time. These values do not significantly vary over time.

The QScan column displays information about GRS queue scans in the interval that was done by the task. For zCX, this value is normally zero.

7.5.2 RMF CPC capacity

A second view that can you can choose from the RMF Overview Report Selection Menu is the CPC capacity. This report shows the processor usage of the different LPARs on the machine for the different processor types.

Example 7-44 RMF CPC Capacity

```
RMF V2R4   CPC Capacity                    Line 1 of 46
Command ===>                                                Scroll ===> CSR

Samples: 100     System: SC74  Date: 09/16/19  Time: 04.35.00  Range: 100   Se

Partition:   ARIES22    8561 Model 716
CPC Capacity:    2953   Weight % of Max: ****   4h Avg:  220   Group:   N/A
Image Capacity:  1477   WLM Capping %:    0.0   4h Max:  261   Limit:   N/A
MT Mode IIP:        2   Prod % IIP:      99.7   AbsMSUCap: N

Partition  --- MSU ---  Cap     Proc    Logical Util %   - Physical Util % -
             Def   Act  Def      Num    Effect   Total   LPAR  Effect  Total

*CP                             46.0                      0.8    62.6   63.4
ARIES02        0  1477  N N N    8.0       100     100    0.0    50.0   50.0
ARIES16        0     1  N N N    2.0       0.4     0.4    0.0     0.0    0.1
ARIES2B        0    98  N N N    8.0       6.7     6.9    0.1     3.3    3.5
ARIES21        0     2  N N N    2.0       0.6     0.6    0.0     0.1    0.1
ARIES22        0   250  N N N    4.0      33.8    34.0    0.0     8.5    8.5
ARIES23        0     7  N N N    4.0       0.9     1.0    0.0     0.2    0.3
ARIES24        0     4  N N N    4.0       0.5     0.5    0.0     0.1    0.1
ARIES25        0     4  N N N    4.0       0.5     0.6    0.0     0.1    0.1
ARIES26        0     2  N N N    2.0       0.5     0.6    0.0     0.1    0.1
ARIES27        0     2  N N N    2.0       0.7     0.7    0.0     0.1    0.1
ARIES28        0     1  N N N    4.0       0.1     0.2    0.0     0.0    0.0
ARIES29        0     1  N N N    2.0       0.2     0.3    0.0     0.0    0.0
PHYSICAL                                                   0.5            0.5

*IFL                            70.0                      0.0    13.9   13.9
ARIES1B                N N N    8.0       100     100    0.0    13.8   13.8
ARIES12                N N N    8.0       0.1     0.1    0.0     0.0    0.0
ARIES13                N N N   16.0       0.0     0.1    0.0     0.0    0.0
ARIES14                N N N    8.0       0.1     0.1    0.0     0.0    0.0
ARIES15                N N N   16.0       0.0     0.1    0.0     0.0    0.0
ARIES17                N N N    6.0       0.3     0.4    0.0     0.0    0.0
ARIES18                N N N    4.0       0.1     0.2    0.0     0.0    0.0
ARIES28                N N N    4.0       0.0     0.0    0.0     0.0    0.0
PHYSICAL                                                   0.0            0.0

*ICF                             8.0                      0.1     0.0    0.1
ARIES2C                N N N    2.0       0.0     0.0    0.0     0.0    0.0
ARIES2D                N N N    2.0       0.0     0.0    0.0     0.0    0.0
ARIES2E                N N N    2.0       0.0     0.0    0.0     0.0    0.0
ARIES2F                N N N    2.0       0.0     0.0    0.0     0.0    0.0
PHYSICAL                                                   0.1            0.1

*IIP                            33.0                      0.0    44.8   44.8
ARIES02                N N N    1.0       100     100    0.0     6.2    6.2
ARIES2B                N N N    4.0       4.1     4.1    0.0     1.0    1.0
ARIES21                N N N    2.0       0.3     0.3    0.0     0.0    0.0
ARIES22                N N N    4.0      99.8    99.8    0.0    24.9   24.9
ARIES23                N N N    4.0      49.8    49.8    0.0    12.5   12.5
ARIES24                N N N    4.0       0.1     0.1    0.0     0.0    0.0
ARIES25                N N N    4.0       0.2     0.2    0.0     0.1    0.1
ARIES26                N N N    2.0       0.0     0.0    0.0     0.0    0.0
ARIES27                N N N    2.0       0.0     0.0    0.0     0.0    0.0
ARIES28                N N N    4.0       0.0     0.0    0.0     0.0    0.0
```

```
ARIES29                N N N    2.0       0.0     0.0    0.0     0.0    0.0
PHYSICAL                                                0.0            0.0
```

Example 7-44 shows one table for each type of processor. Each table starts with a summary line for that processor type where the processor type is preceded with an asterisk, like ***CP**. This line shows the number of processors of that type that are physically available in this machine and also the overall utilization of this processor pool.

After this summary line, one line per LPAR follows, for all LPARs for which at least one processor of this type is defined. In these LPAR statistics lines, you see the number of logical processors of that type that are defined to the LPAR. You also view their utilization as seen by the LPAR itself (**logical utilization**); and the percentage of the overall physical utilization.

If a processor type on an LPAR has a very high logical percentage, it means that the system is using nearly all processor resources that are defined to this system. If the physical percentage of a processor pool sum shows a very high percentage, this means that the machine has used almost all installed capacity of that processor type.

Very high percentages can cause poor responsiveness for your zCX instance. The zCX itself is mainly running on zIIP processors. Nonetheless, it depends on other components of the z/OS system that are running on CP, such as TCPIP.

7.5.3 RMF job information

To check performance of the zCX instance, navigate back to the RMF III main menu and choose option 2 (JOBS). You must enter the name of your zCX instance in the Jobname field before you select any options.

Example 7-45 RMF Job Report Selection Menu

```
RMF Job Report Selection Menu
 Selection ===> 1

 Enter selection number or command and jobname for desired job report.

   Jobname ===> ZCXED01_

   1 DEVJ            Delay caused by devices                      (DVJ)
  1A DSNJ            .. Data set level                            (DSJ)
   2 ENQJ            Delay caused by ENQ                           (EJ)
   3 HSMJ            Delay caused by HSM                           (HJ)
   4 JESJ            Delay caused by JES                           (JJ)
   5 JOB             Delay caused by primary reason            (DELAYJ)
   6 MNTJ            Delay caused by volume mount                 (MTJ)
   7 MSGJ            Delay caused by operator reply               (MSJ)
   8 PROCJ           Delay caused by processor                     (PJ)
   9 QSCJ            Delay caused by QUIESCE via RESET command     (QJ)
  10 STORJ           Delay caused by storage                       (SJ)
  11 XCFJ            Delay caused by XCF                           (XJ)

 These reports can also be selected by placing the cursor on the
 corresponding delay reason column of the DELAY or JOB reports and
 pressing ENTER or by using the commands from any panel.
```

Selection 1 (DEVJ) shows the main device delay that is affecting the selected job. This information might be of greatest value for problem determination in zCX.

Example 7-46 RMF Job Delays (Device)

```
RMF V2R4   Job Delays                      Line 1 of 1
 Command ===>                                                Scroll ===> CSR

 Samples: 100     System: SC74  Date: 09/15/19  Time: 20.15.00  Range: 100   Sec

 Job: ZCXED01    Requested delay: Excessive pending time on volume BH5ZCD.

 Probable causes: 1) Contention with another system for use of the volume.
                  2) Overutilized channel, control unit or head of string.


 -------------------------- Volume BH5ZCD Device Data --------------------------
 Number:    09959       Active:      82%        Pending:   66%     Average Users
 Device:    33909       Connect:     16%        Delay DB:  58%        Delayed
 Shared:    Yes         Disconnect:   0%        Delay CM:   1%          0.3
 PAV:       6.6H
 --------------------------- Job Performance Summary ---------------------------
         Service       WFL -Using%- DLY IDL UKN ---- % Delayed for ---- Primary
 CX ASID Class    P Cr  %   PRC DEV  %   %   %  PRC DEV STR SUB OPR ENQ Reason
 SO 0175 SYSSTC   1     82   53  45  21   0  20  13   8   0   0   0   0 ZCXRJ02
```

If the task is delayed for I/O, this display shows the most active device for the task and suggests possible reasons for the delay.

Selection 8 (PROCJ) provides a similar display for processor delays.

Example 7-47 RMF Job Delays (CPU)

```
RMF V2R4   Job Delays                      Line 1 of 1
 Command ===>                                                Scroll ===> CSR

 Samples: 100     System: SC74  Date: 09/15/19  Time: 20.15.00  Range: 100   Sec

 Job: ZCXED01      Primary delay: Job is waiting to use the processor.

 Probable causes: 1) Higher priority work is using the system.
                  2) Improperly tuned dispatching priorities.


 ------------------------- Jobs Holding the Processor --------------------------
 Job:         ZCXRJ02         Job:         ZCXSM01         Job:         ZCXZB01
 Holding:         11%         Holding:         10%         Holding:         10%
 PROC Using:      67%         PROC Using:      62%         PROC Using:      69%
 DEV Using:       31%         DEV Using:       20%         DEV Using:       40%
 --------------------------- Job Performance Summary ---------------------------
         Service       WFL -Using%- DLY IDL UKN ---- % Delayed for ---- Primary
 CX ASID Class    P Cr  %   PRC DEV  %   %   %  PRC DEV STR SUB OPR ENQ Reason
 SO 0175 SYSSTC   1     82   53  45  21   0  20  13   8   0   0   0   0 ZCXRJ02
```

If the task is delayed by processors, you can see which jobs are contributing to the delay.

7.6 Configuring Grafana to monitor zCX containers

One of the most common monitoring tools for Docker container environments is Prometheus, with visualization by Grafana Both cAdvisor and Node-Exporter are needed as data providers

for Prometheus. Each of these four components runs in a separate container and play the following roles:

- Node-Exporter exposes metrics about the Linux operating system.
- cAdvisor exposes metrics about containers.
- Prometheus collects the data of the preceding components.
- Grafana visualizes data that it pulls from Prometheus.

The solution that is demonstrated in this section is designed for a single zCX instance. Therefore, you must repeat this installation for every zCX instance that you want to monitor. For ease of distribution to different zCX instances, consider pushing the images to a private registry. Detailed instructions on setup and use of a private registry can be found in Chapter 6, “Private registry implementation” on page 107.

See Section 5.2, “Get an image from Docker Hub” on page 96 for detailed instructions on how to get images into your zCX, such as the images that are mentioned in the following paragraphs.

7.6.1 Install Node-Exporter

To use Node-Exporter, you must build your own image, because there are different versions of Node-Exporter. The image **ibmcom/node-exporter-s390x:v0.16.0-f3** available on Docker Hub is not compatible for the purpose that is needed here.

To build the node-exporter image, you start by creating a directory named **nodeexporter** and create a file named **`Dockerfile`** within that directory. The contents that are needed for this file are shown in Example 7-48.

Example 7-48 Dockerfile for Node-Exporter

```
########## Dockerfile for Node Exporter #########
# To build this image, from the directory containing this Dockerfile
# (assuming that the file is named Dockerfile):
# docker build -t <image_name> .
# Base Image
FROM s390x/ubuntu:18.04
RUN apt-get update && apt-get install -y prometheus-node-exporter
EXPOSE 9100
# attention: The following command is a very long line and not multiple lines
# please be sure to eliminate any new-line characters that might exist
# from cutting and pasting it
CMD prometheus-node-exporter --path.procfs="/host/proc" --path.sysfs="/host/sys"
--collector.diskstats --collector.loadavg --collector.meminfo --collector.netdev
--collector.netstat --collector.stat --collector.time --collector.uname --collector.vmstat
--collector.filesystem.ignored-mount-points="^/(sys|proc|dev|host|etc)($$|/)"
# this is the first line after the CMD command
```

To build the image, navigate to the directory that contains the preceding Dockerfile, and run the following command. (Notice the final period, which is mandatory.):
`docker build -t nodeexporter:latest .`

Before you can start the Node-Exporter, you first must run the following command:
`docker network create monitoring`

You need to do this one time only. Subsequently, you can start Node-Exporter by using the following command:

```
docker run --name nodeexporter -v /proc:/host/proc:ro -v /sys:/host/sys:ro \
```

```
-v /media:/rootfs/media:ro -v /etc/hostname:/etc/host_hostname:ro -p 9100:9100
-d \
--network monitoring nodeexporter:latest
```

If Node-Exporter has started correctly, you should be able to navigate with a browser to your zCX-address port 9100 to see the page, as shown in Figure 7-3.

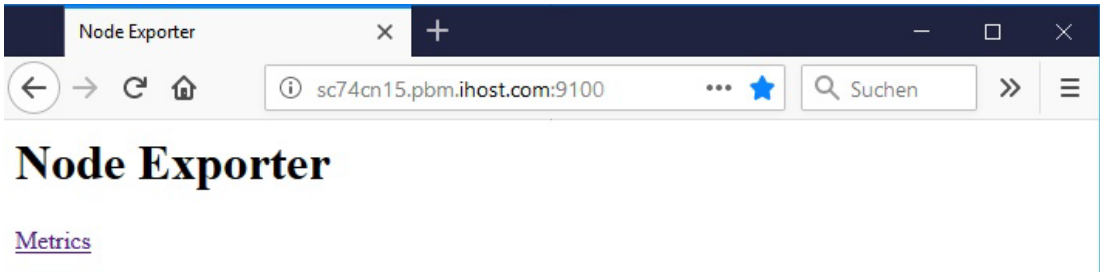


Figure 7-3 Node Exporter starting page

7.6.2 Install cAdvisor

To install cAdvisor, you can use the image **ibmcom/cadvisor-s390x:0.33.0**, which is available on Docker Hub. The command to start cAdvisor is as follows:

```
docker run -v /proc:/rootfs/proc:ro -v /media:/rootfs/media:ro \
-v /var/run/docker.sock:/var/run/docker.sock:ro -v /sys:/sys:ro \
-v /var/lib/docker/:/var/lib/docker:ro -v /dev/disk/:/dev/disk:ro \
-p 8080:8080 -d --network monitoring --name=cadvisor
ibmcom/cadvisor-s390x:0.33.0
```

You can verify that cAdvisor is running properly by navigating with your browser to the address of your zCX instance port 8080 to see the cAdvisor main page, as shown in Figure 7-4.

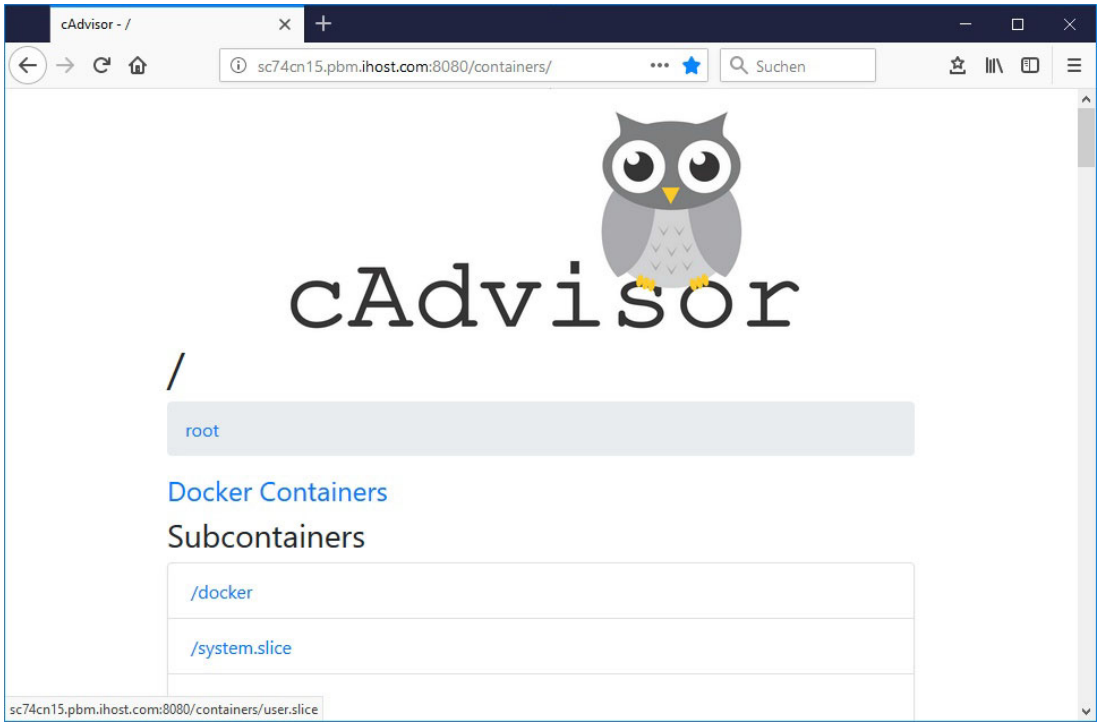


Figure 7-4 cAdvisor Entry Page

7.6.3 Install Prometheus

To install Prometheus, use the *ibmcom/prometheus-s390x:v2.8.0* image, which is available on Docker Hub.

Prometheus needs configuration file settings to enable it to use the data providers node-exporter and cadvisor that you started before. The easiest way to accomplish this is to build your own image.

To build your own image of the Prometheus image first create a directory, for example, **Prometheus**, and then navigate into that directory. Next, create the configuration file named **`prometheus.yml`** in this directory by using the command **`touch prometheus.yml`**. Then, edit the file by using **`vi`**. The contents of the file are visible in Example 7-49. You can also download the file as instructed in Appendix A, “Obtaining the additional material” on page 251.

Example 7-49 file prometheus.yml

```
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every
1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. Default is every 1 minute.
  # scrape_timeout is set to the global default (10s).
  external_labels:
     monitor: 'my-project'
# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093
```

```
# Load rules once and periodically evaluate them according to the global
'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this
config.
  - job_name: 'prometheus'
    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
            - targets: ['localhost:9090', 'cadvisor:8080', 'nodeexporter:9100']
```

Next, create a Dockerfile using the command **touch Dockerfile** and edit it using **vi**:

Example 7-50 Dockerfile for Prometheus

```
#Base Image is prometheus
FROM ibmcom/prometheus:v2.8.0
#Insert the config file
COPY prometheus.yml /etc/prometheus/prometheus.yml
```

Having done this, you can issue the following command to build the image:
docker build -t prometheus:v2.8.0 .
Be sure to include the dot at the end of the command.

Because Prometheus is the component that stores the monitored data, you should store this data to persistent storage by using a Docker volume. You can create a Docker volume by using the command **docker volume create prometheus-data**.

The command to start your Prometheus instance would then be the following:

docker run --name prometheus --network monitoring -v prometheus-data:/prometheus -p 9090:9090 -d prometheus:v2.8.0

To verify that Prometheus is up and running, point a browser to the zCX address through port 9090, as shown in Figure 7-5 on page 171.

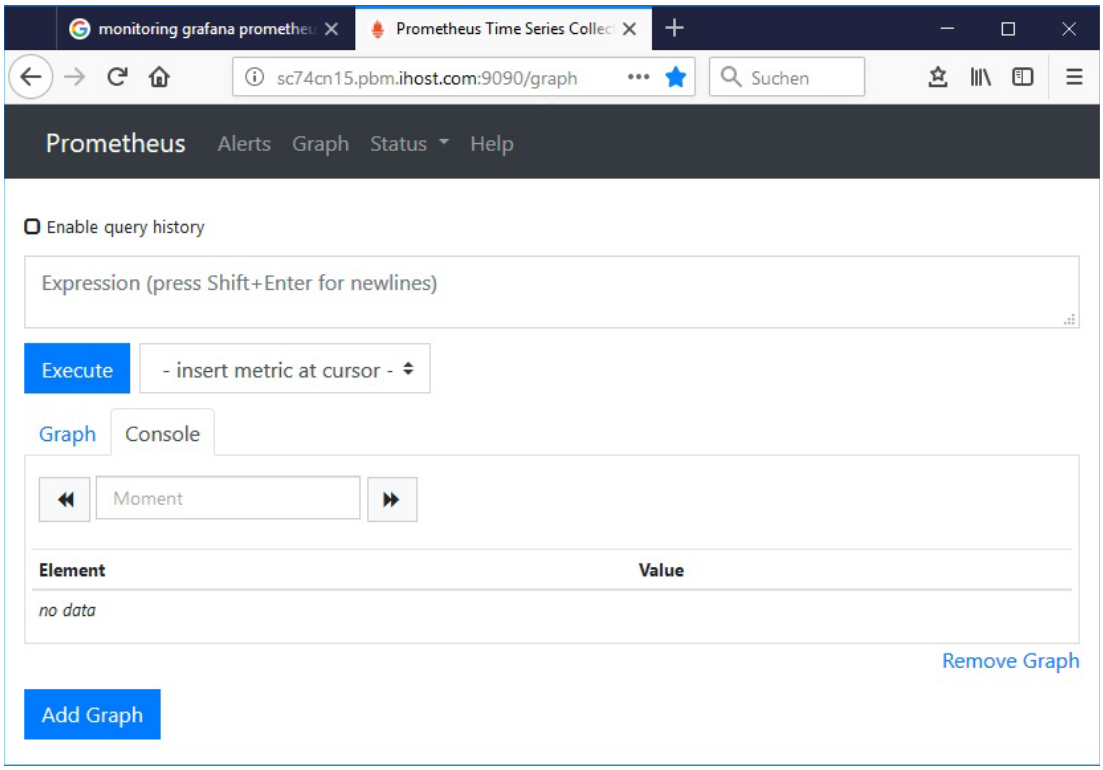


Figure 7-5 Initial view of Prometheus

7.6.4 Installation of Grafana

You can install Grafana by using the image *ibmcom/grafana:5.2.0-f3* available on Docker Hub.

Because Grafana stores several types of configuration data, consider using persistent storage to run the container. To achieve this, you create a Docker volume by using the following command:

```
docker volume create grafana-data
```

Mount this volume through the following run command for Grafana:

```
docker run --name grafana -d -p 3000:3000 -v grafana-data:/var/lib/grafana
ibmcom/grafana:5.2.0-f3
```

To verify that Grafana is up and running, connect your browser to the zCX address on port 3000. If you used the default Grafana setup, you can log on with userid “admin“ and password “admin“. Grafana prompts you to change the password.

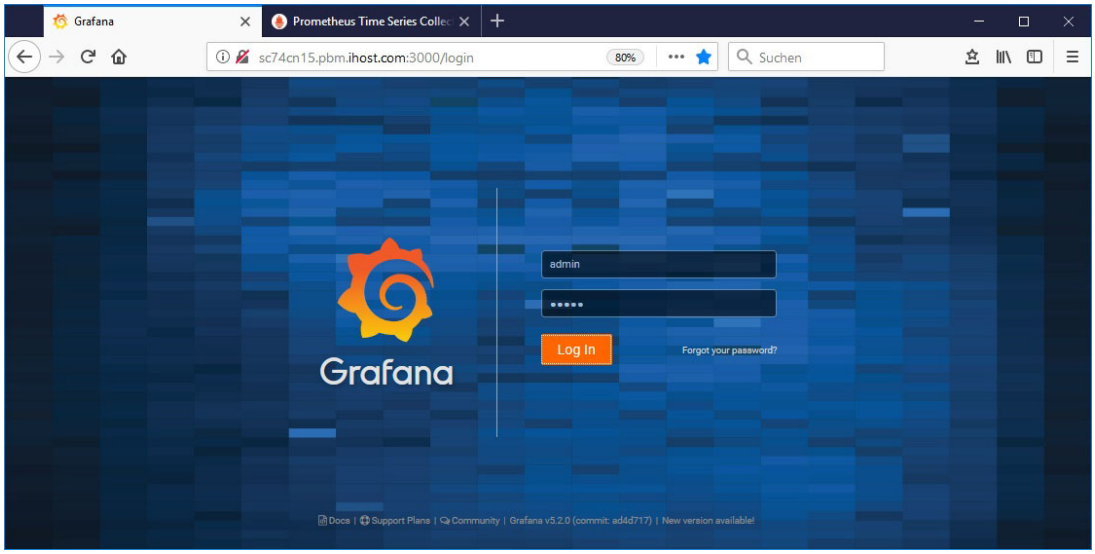


Figure 7-6 Grafana Logon Window

After you change the password, you are redirected to the home page of your Grafana server. You can set preferences for your user by hovering on the user button (see red arrow in Figure 7-7 on page 172) and selecting “Preferences“. From there, we changed the UI Theme from “dark“ to “light“ for better readability in the documentation.

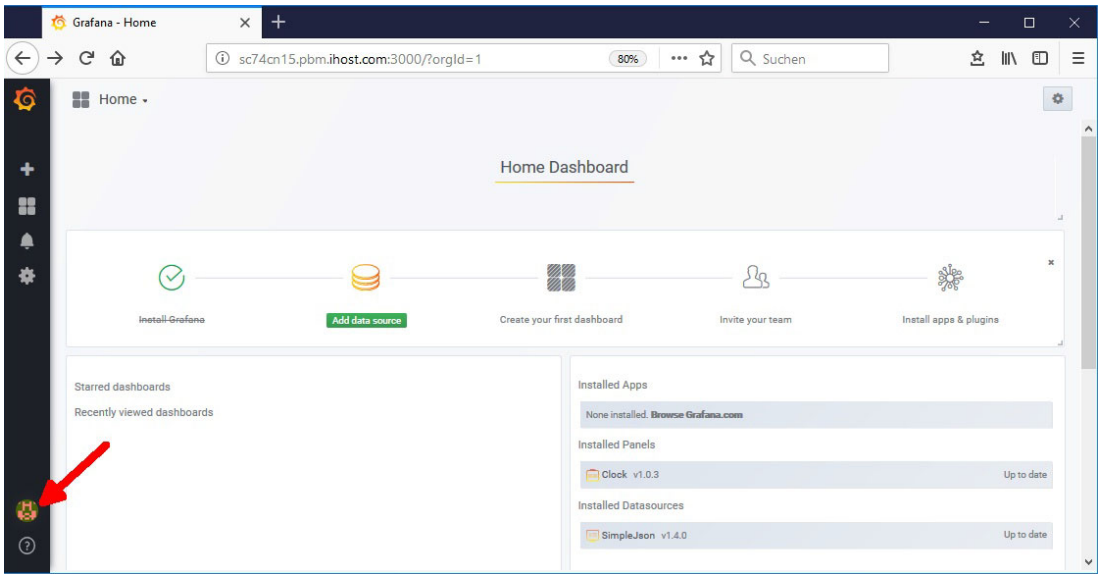


Figure 7-7 Grafana home page

7.6.5 Adding Prometheus as data source to Grafana

To use Grafana for visualization of data that is collected by Prometheus, you must add your Prometheus instance as a data source to your Grafana server.

You have two options to get to the add datasource panel as shown in Figure 7-8 on page 173.

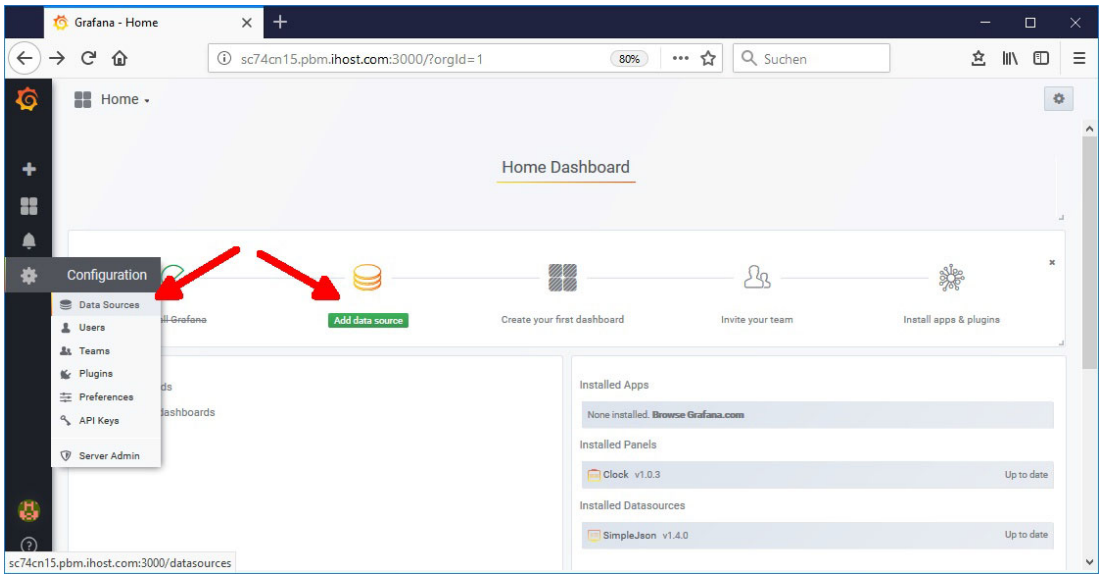


Figure 7-8 Grafana select adding a data source

In the panel for adding data sources, first enter the name that you want to give to your Prometheus data source. Then, select “Prometheus“ as type of the data source from the drop-down list as shown in Figure 7-9 on page 174. After you select the type **Prometheus**, you must enter the URL of your Prometheus instance, including the port number. You finish this panel by clicking **save&test**. You see a message telling that confirms that the data source is working.

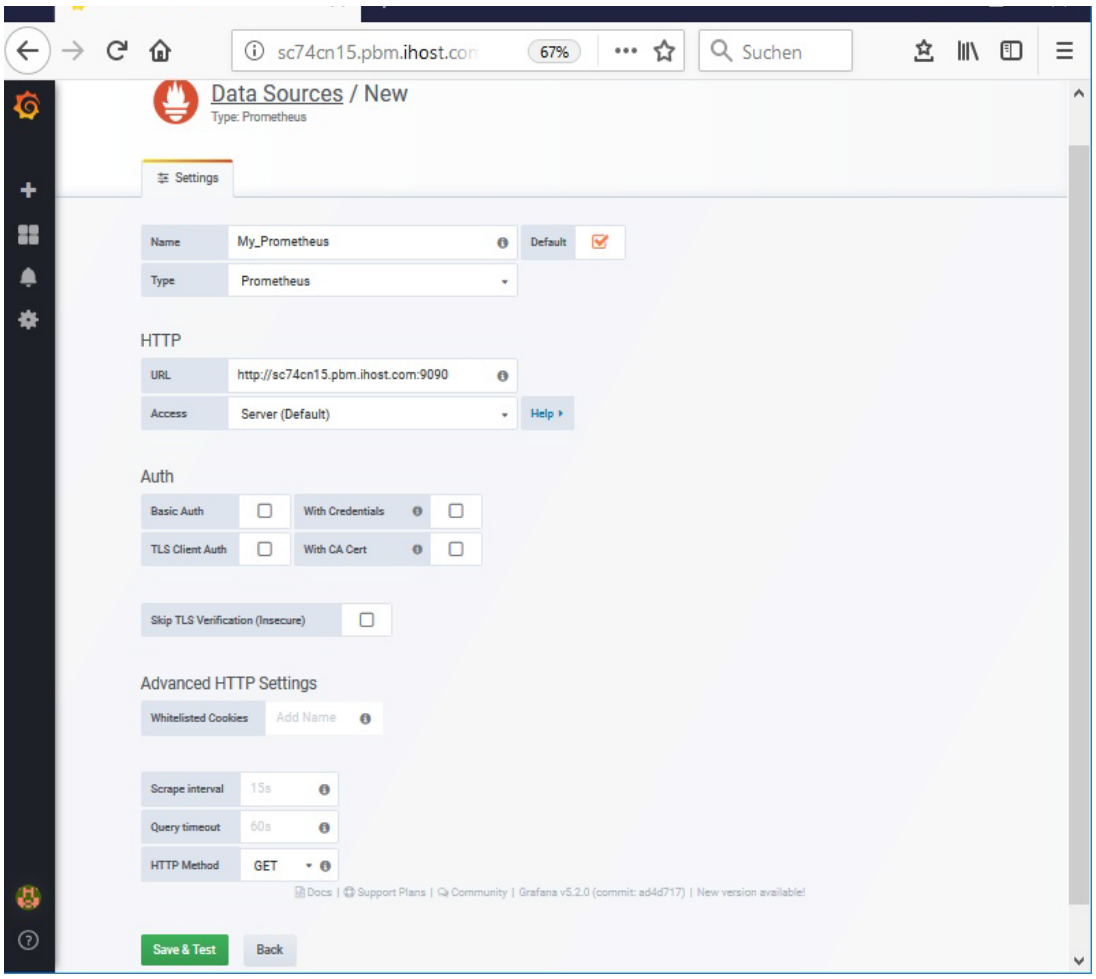


Figure 7-9 Adding Prometheus data source to Grafana

7.6.6 Creating a first dashboard

To add your first dashboard, import the sample that is provided by the IBM zCX team. The file is named *IBM zCX for z_OS Monitoring.json*. You download this file to your workstation as part of the material that is documented in Appendix A, “Obtaining the additional material” on page 251.

Next, select *Manage* from the dashboard side menu as shown in Figure 7-10 on page 175.

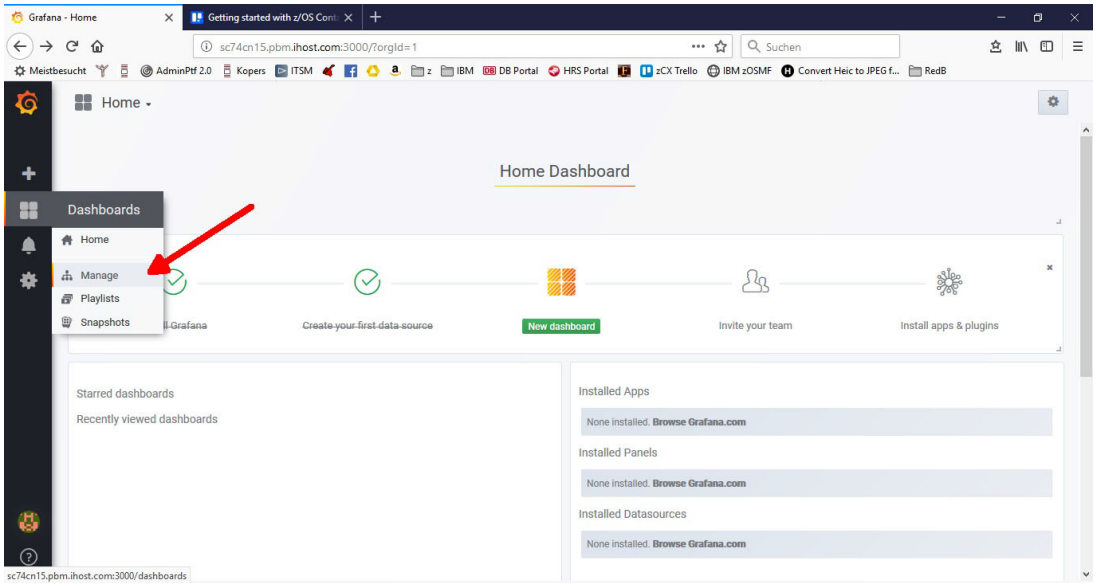


Figure 7-10 Managing dashboards

Now, click **Import** in the upper right of the window as shown in Figure 7-11.

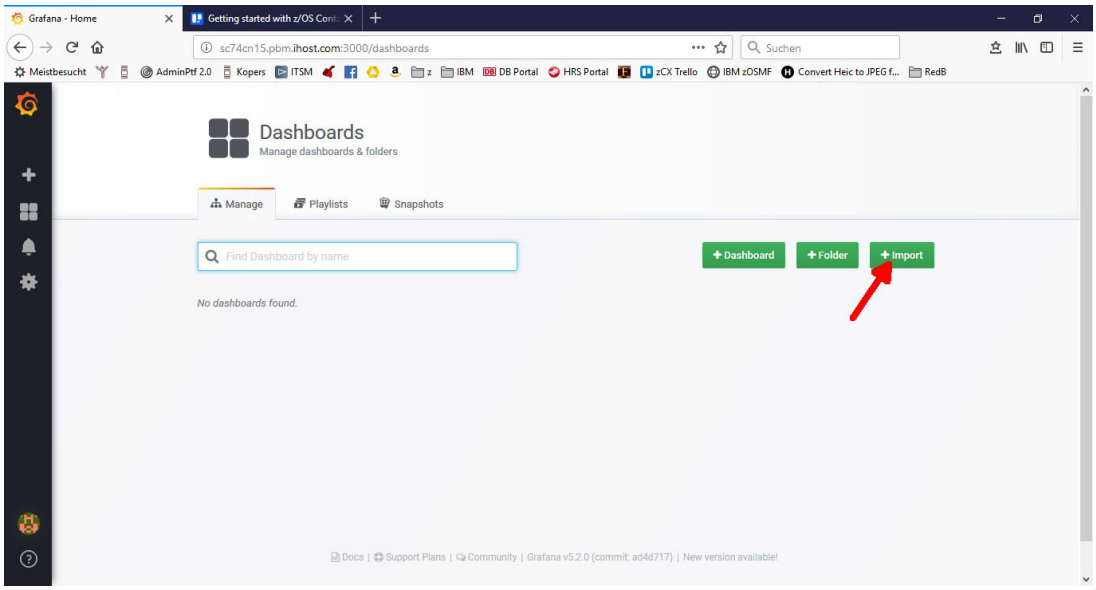


Figure 7-11 Select to import a dashboard

On the next screen, click **`Upload .json File`** as in Figure 7-12 on page 176. Then, select the downloaded JSON file on your workstation.

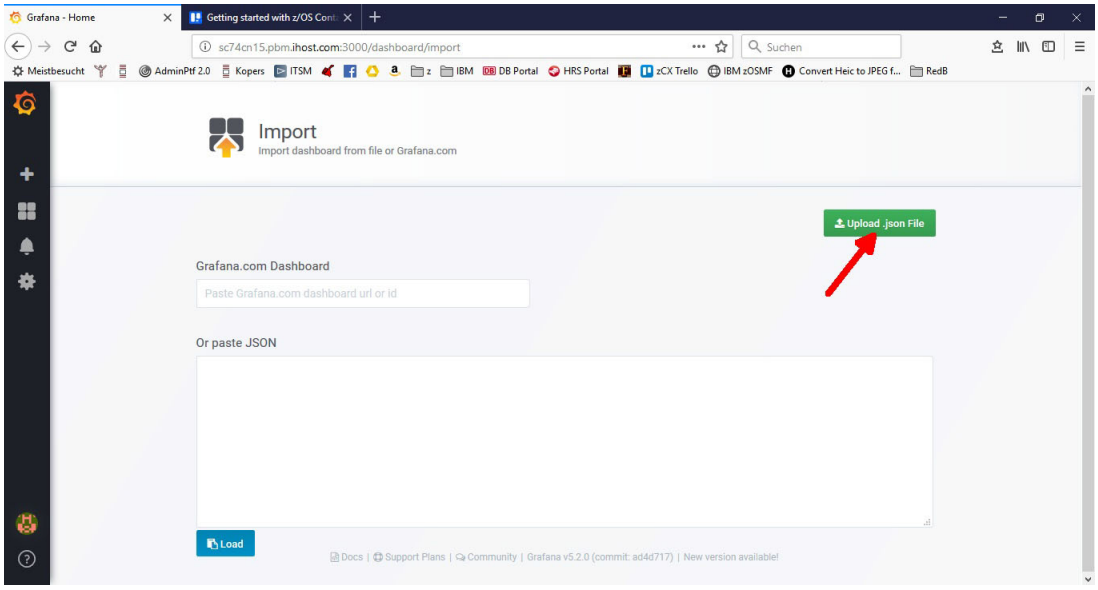


Figure 7-12 Upload .json File

The last step is to insert your Prometheus instance as data source and click **Import**.

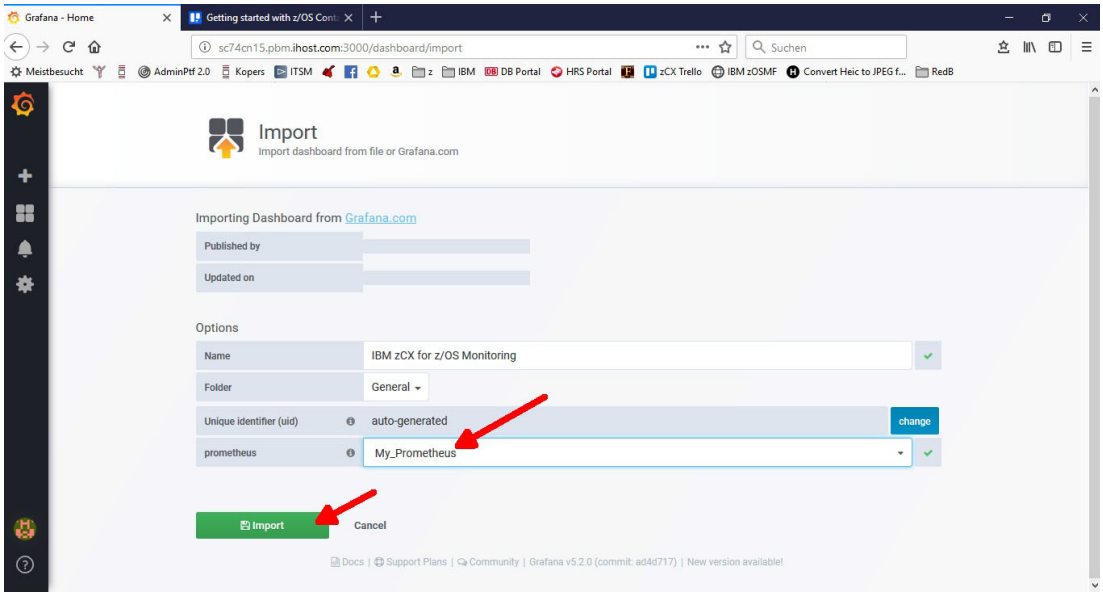


Figure 7-13 Import IBM zCX for z/OS Monitoring.json

Now, the IBM zCX for z/OS Monitoring dashboard is displayed as an option in the list of available dashboards on the dashboard home page as shown in Figure 7-14 on page 177.

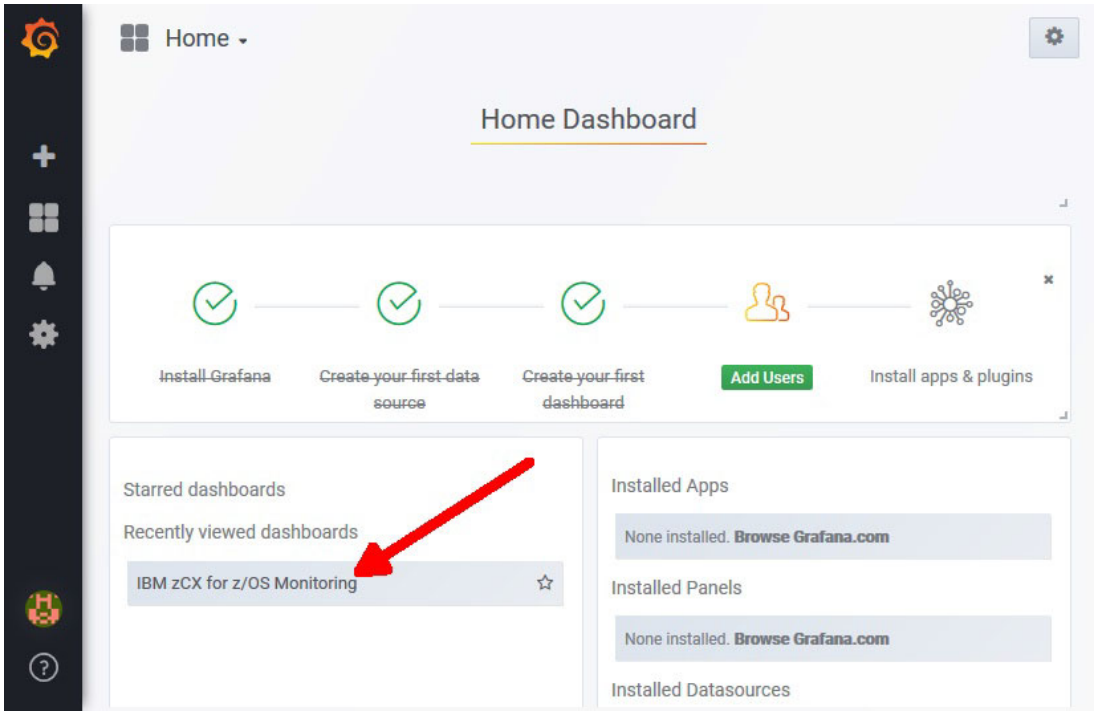


Figure 7-14 Grafana Dashboard Home Page

Click the dashboard title to open the dashboard.

7.7 Monitoring with Grafana on container level

Now that you have created your first dashboard, this section shows you how to adjust it to meet your needs and what the different graphs mean.

The actual dashboard that you see might differ from what is shown in this book, as there might be updates to the dashboard after the book has been published.

The following paragraphs are meant to give an impression of what this monitoring solution can do for you. At the upper left of each graph in the dashboard, you see a small info-icon. Hover on the icon to get a short description of what that graph is showing.

7.7.1 Adjusting the dashboard

When you first open the dashboard, it looks similar to what is seen in Figure 7-15 on page 178.

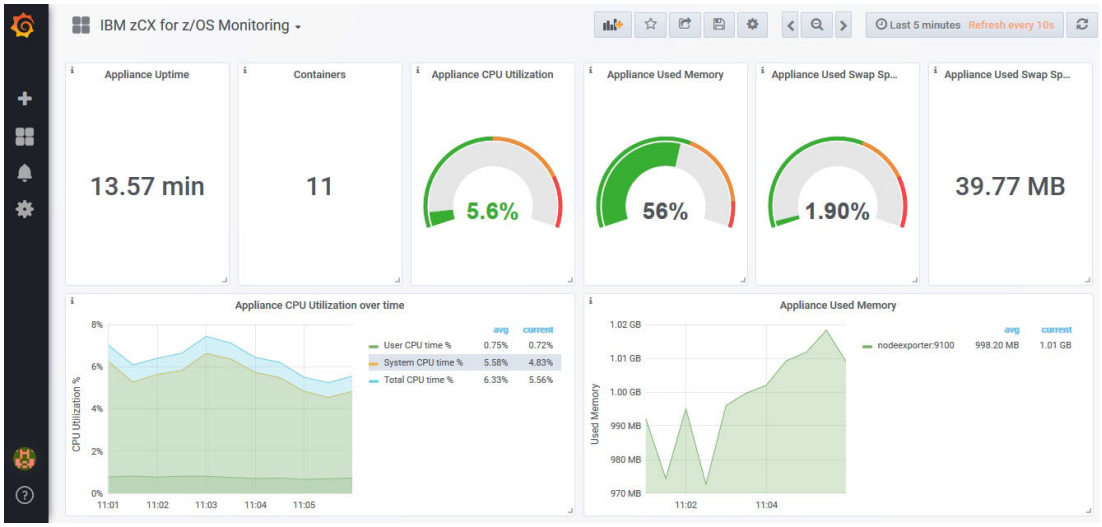


Figure 7-15 First view of IBM zCX for z/OS Monitoring dashboard

You can easily resize any tile in the dashboard by positioning your mouse over the small angle in the lower right and while pressing the left mouse button move your mouse. Accordingly, the lower left of the tile moves with the mouse pointer.

To move a tile on your dashboard, drag the tile by clicking and holding the title line of the tile. Then, drag the tile to the desired position. The other tiles of the dashboard reorganize themselves around the newly positioned tile.

When you are satisfied with the changes, you can save them by clicking the diskette icon at the top of the window.

Figure 7-16 on page 178 shows the points where you position the mouse to resize or move the tile.

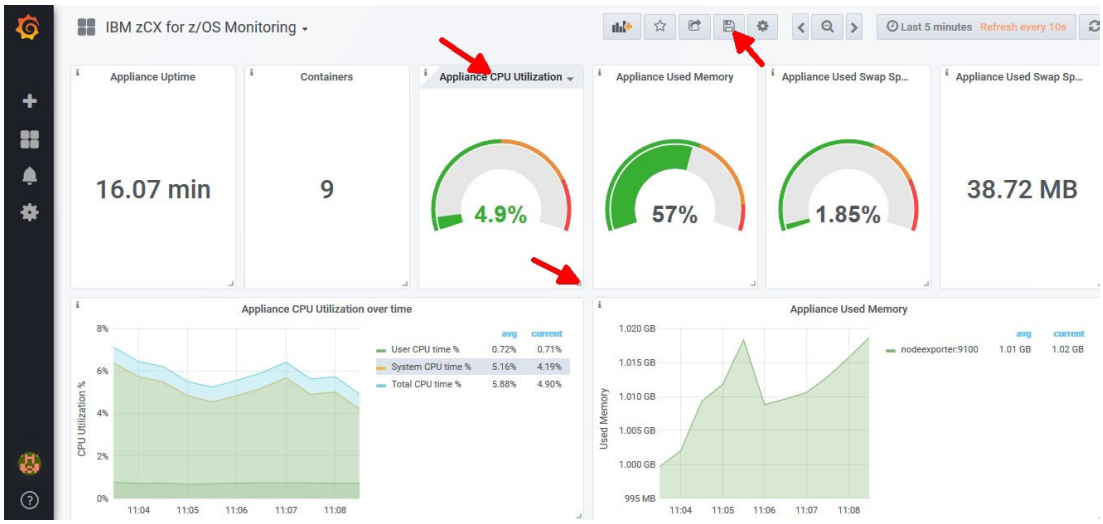


Figure 7-16 Spots to resize and rearrange tiles on the dashboard

7.7.2 Instance level data

The first tiles on the dashboard shown in Figure 7-16 on page 178 show information on the zCX instance level as seen from within the zCX.

Appliance Uptime

The uptime tile just tells since how long the zCX instance is up and running.

Containers

The containers tile shows how many containers are currently running within this zCX instance.

Appliance CPU Utilization gauge

This tile tells the CPU percent utilized for available CPU's, from the Linux perspective. To interpret this value, keep in mind that you have configured a number of virtual processors to the zCX instance. This does not mean that your zCX instance has exclusive access to these processors. It gets only a share of the processors that are available to z/OS, based on the decisions of the z/OS Workload Manager.

The share that z/OS gives to zCX is seen by zCX as 100% CPU and the percentage shown in the tile is relative to this share.

Appliance Used Memory gauge

This tile shows the amount of memory within the zCX instance that is currently in use for processes, as a percentage of all memory available to the zCX instance.

Appliance Used Swap Space gauge

This tile shows the percentage of swap memory that is in use.

Appliance Used Swap Space

This tile shows the absolute value for the swap space that is in use.

Appliance CPU Utilization over time

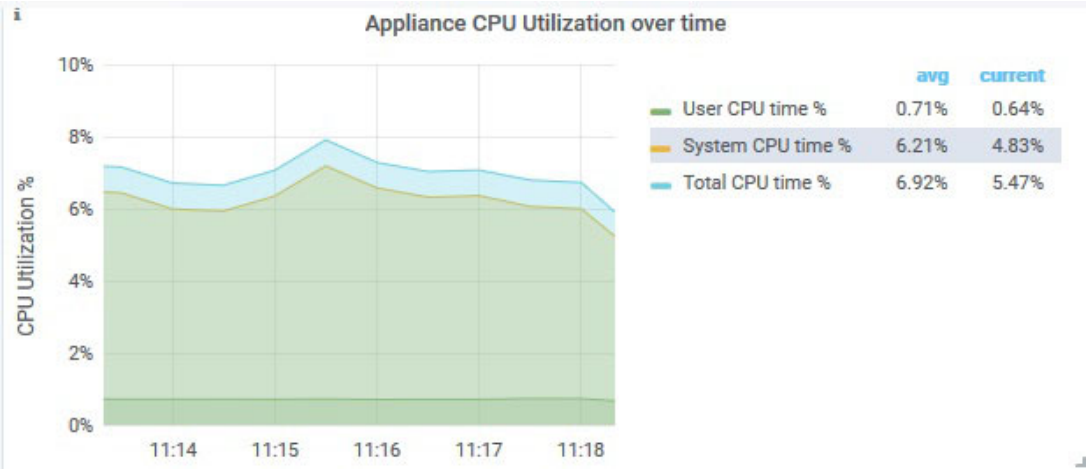


Figure 7-17 Appliance CPU Utilization over time

The graph for CPU utilization shows the CPU utilization over time distinguished between system and user usage.

Click the legend to the right of the graph on the words **system** or **user** to limit the display to that part of the CPU usage. Click the legend again, to show the full graph again.

Appliance Used Memory

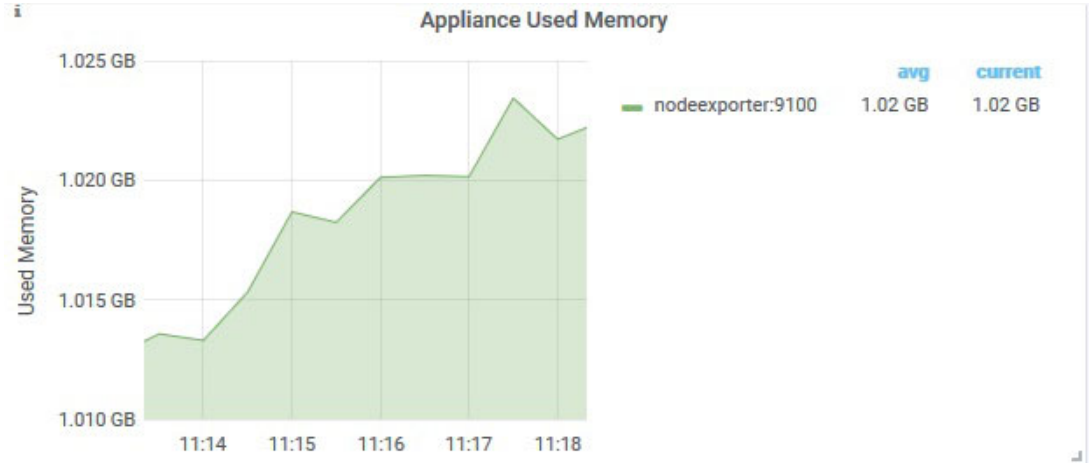


Figure 7-18 Appliance Used Memory

This graph shows the memory usage of the zCX instance.

Appliance Network Traffic

The Network Traffic tile shows the amount of data that is sent and received over the network by the zCX instance. If you hover over the graph, it shows the numeric values for the point in time where your mouse is located, as shown in Figure 7-19.

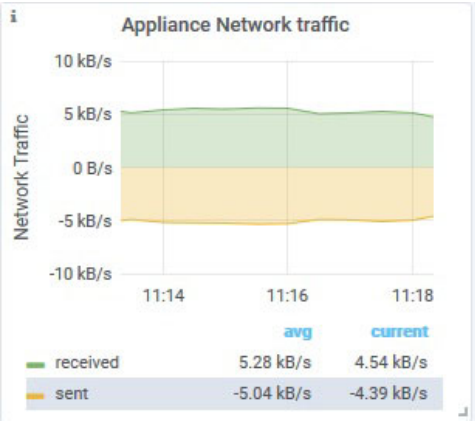


Figure 7-19 Appliance Network Traffic

Because of the way Grafana formats this graph, sent bytes are shown as negative values. Actually, they are positive values.

Appliance Disk IO

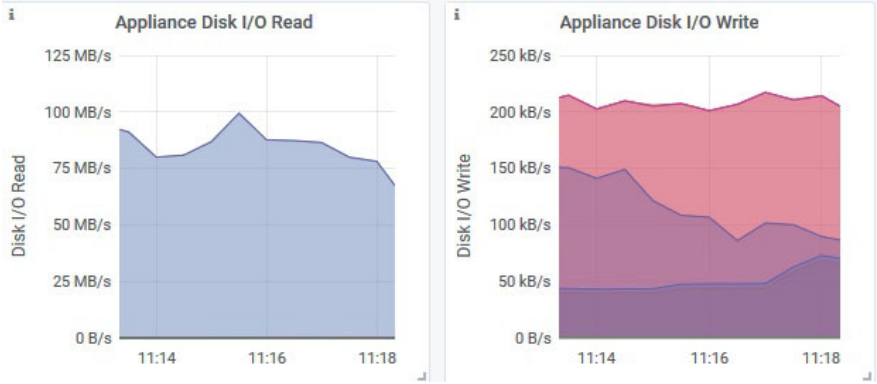


Figure 7-20 Appliance Disk IO

These tiles show the sum of all disk I/O that the zCX instance does. These amounts are divided into read and write I/O.

Appliance Load Averages

Appliance Load Averages

Load 1M	Load 5M	Load 15M
0.49	1.06	0.83
0.63	1.12	0.84

Figure 7-21 Appliance Load Averages

This tile shows the average number of processes that are currently active in the upper row and the average number of processes that are waiting to use the processor in the bottom line.

The three columns display these averages taken over 1, 5, and 15 minutes.

7.7.3 Container level data

Container CPU Utilization %

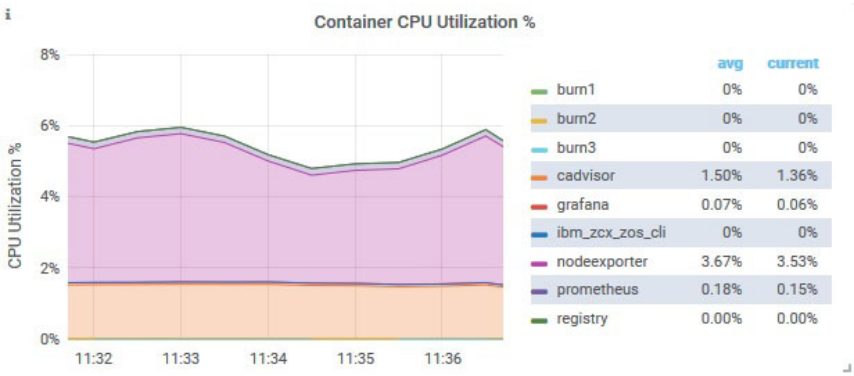


Figure 7-22 Container CPU Utilization %

This graph shows the percentage of the available CPU that is used by the different containers that run within this zCX. You can click a container in the legend of the graph to show only the graph for this single container.

Container Memory Usage

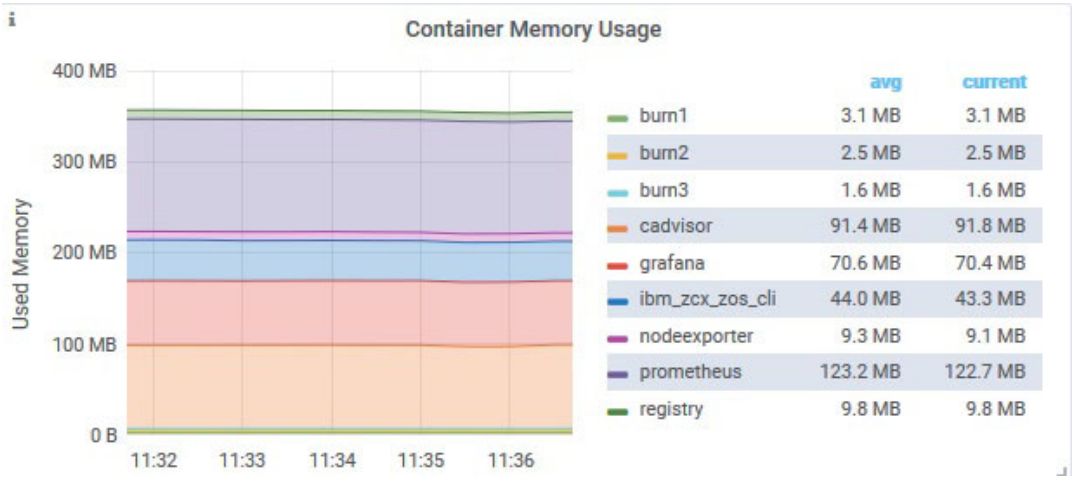


Figure 7-23 Container Memory Usage

This graph is showing the amount of memory that is in use for each container. It shows the real memory footprint, excluding swapped out memory.

Received Network Traffic per Container

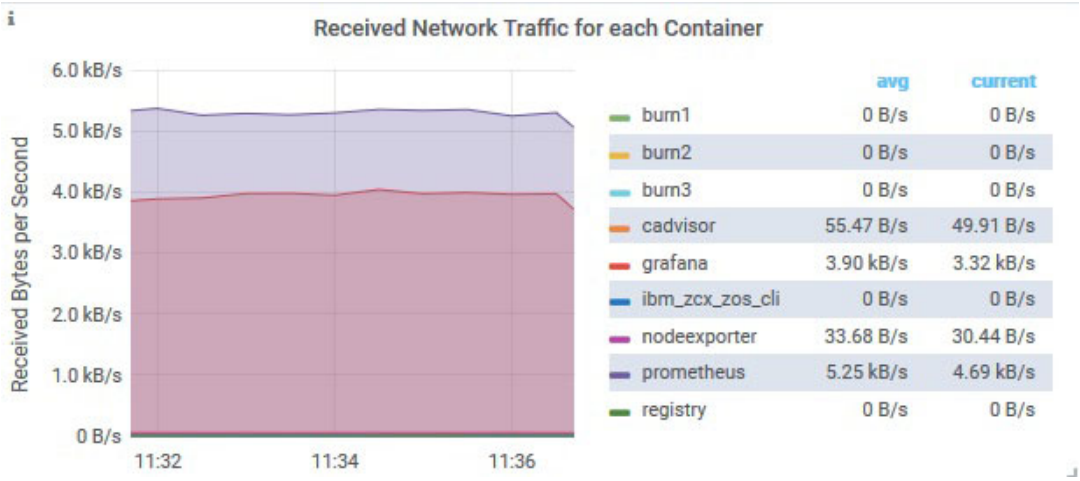


Figure 7-24 Received Network Traffic per Container

This graphic shows the amount of data received per second over the network by each container.

Sent Network Traffic per Container

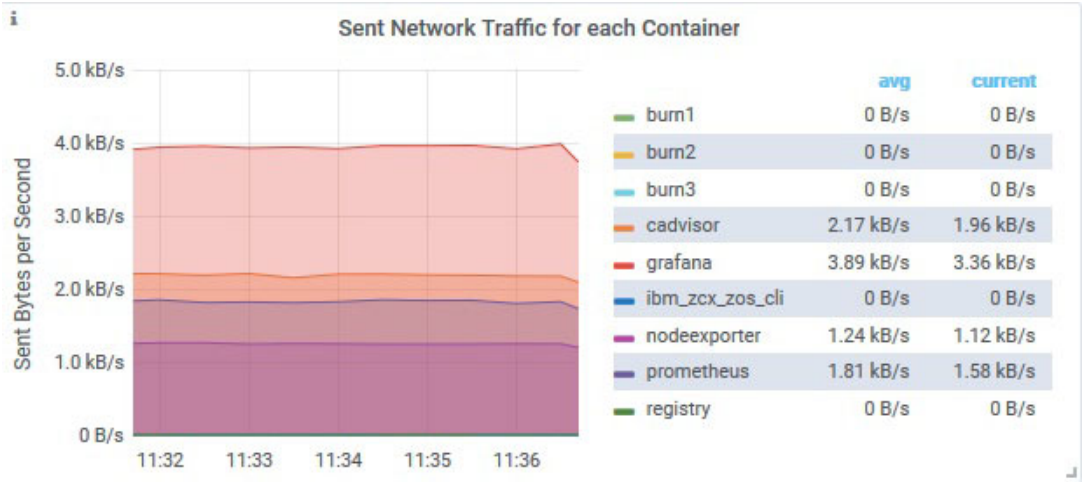


Figure 7-25 Sent Network Traffic per Container

This graphic shows you the amount of data that is sent per second over the network by each container.

Bytes read per second for each Container

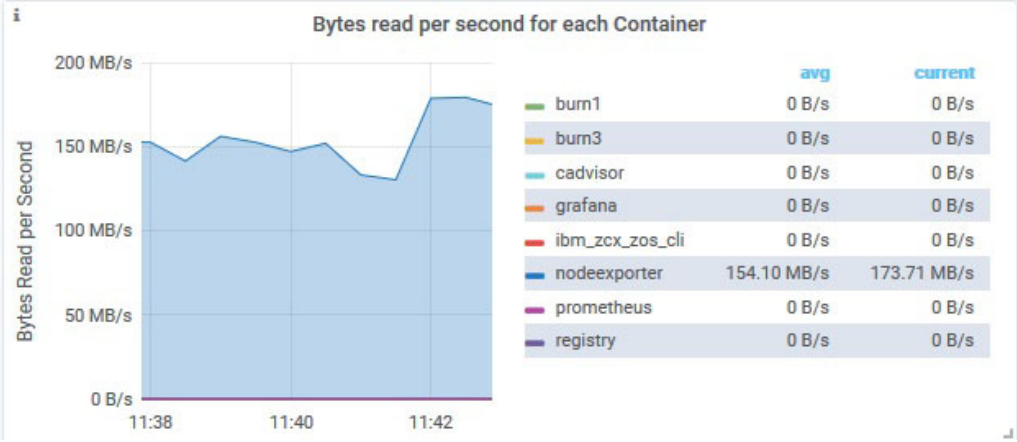


Figure 7-26 Bytes read per Second for each Container

This graph shows the amount of data that is read per second from the file system by each container.

Bytes written per Second by each Container

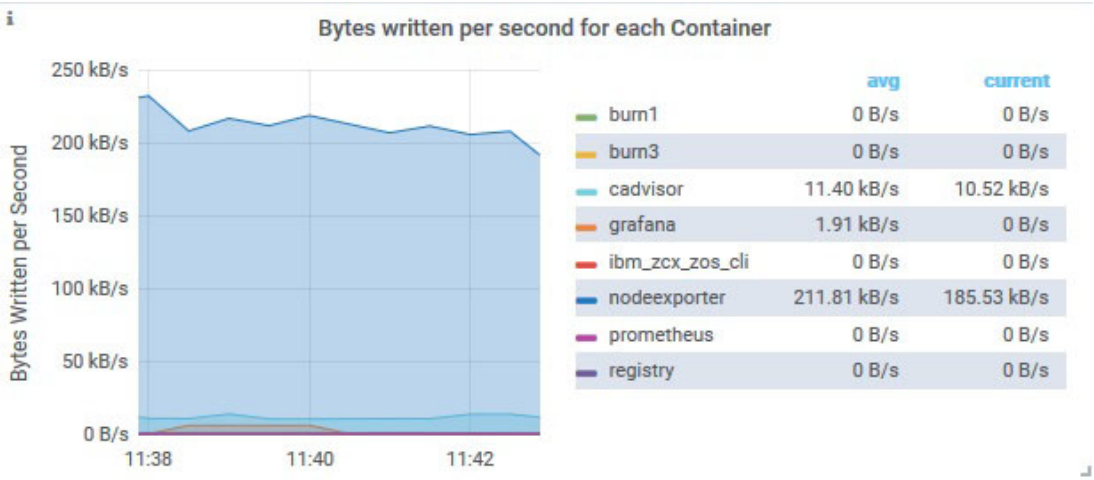


Figure 7-27 Bytes written per Second by each Container

This graph shows the bytes per second that are written to the file system by each container.

Used Memory for each Container

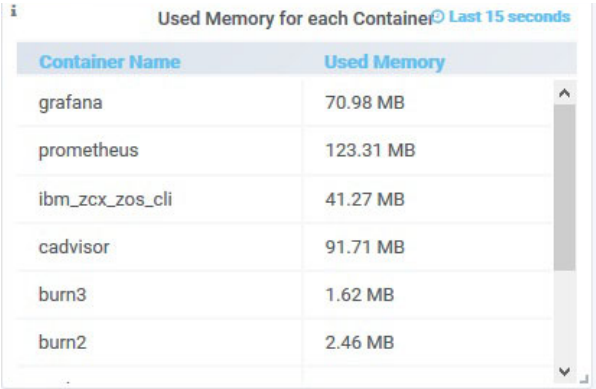


Figure 7-28 Used Memory for each Container

This table shows the current memory usage of each container as numeric values.

Memory Limit for each Container

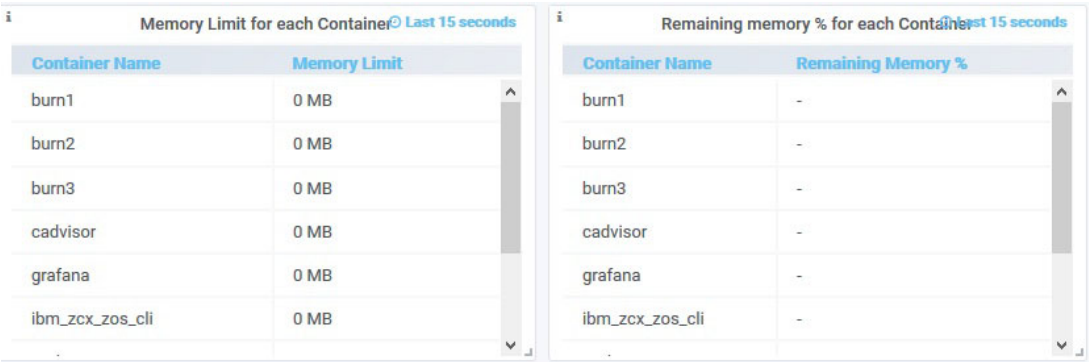


Figure 7-29 Memory Limit for each Container

These two tables show the memory limits for each container.

The table *Memory Limit for each Container* shows the amount of memory that a container is limited to (for example, by the **`--memory`** option of a Docker command). If no limits apply to that container, the table shows the value **0 MB** for this container.

The table *Remaining memory % for each Container* show the amount of memory that is still available to the container before it reaches the memory limit that is shown in the previously mentioned table. If no memory limit applies to the container, a hyphen (-) is shown in this table for the container.

8

Integrating container applications with other processes on z/OS

This chapter provides sample scenarios on how to architect, build, and deploy a hybrid solution that consists of two components on the same z/OS platform: 1) z/OS software and 2) Linux on Z Docker containers.

In this chapter, we demonstrate these three sample scenarios:

- 8.1, “Interconnecting IBM MQ on z/OS with IBM MQ in zCX” on page 188
- 8.2, “Accessing Db2 from within Docker container when you are using Jupyter Notebook” on page 199
- 8.3, “Accessing application in a zCX container from z/OS” on page 209

© Copyright IBM Corp. 2019. All rights reserved.

8.1 Interconnecting IBM MQ on z/OS with IBM MQ in zCX

This section looks at an IBM MQ integration scenario that is based on a messaging infrastructure that uses IBM MQ on z/OS and zCX. We review how to build a IBM MQ container on zCX and create the necessary objects for the integration scenario. A messaging-based solution establishes a shared integration layer. This approach enables the seamless flow of multiple types of data between a customer's heterogeneous systems.

IBM MQ is the market-leading messaging integration middleware product. Originally introduced in 1993 (under the IBM MQSeries® name), it provides an available, reliable, scalable, secure, and high-performance transport mechanism that addresses the integration needs between heterogeneous systems and applications. IBM MQ connects systems and applications, regardless of the platform or environment.

8.1.1 Objectives

Many customers have Systems of Engagement (SoE) applications that run as cloud-native applications inside a container, as microservices. The Systems of Record (SoR) applications run on z/OS as CICS/IMS/Db2 applications. IBM MQ can be used to connect these SoE applications that run inside the zCX with SoR applications that run on z/OS. The value of colocating these applications on z/OS has been explained in 2.1.5, "Planning for network connectivity" on page 31. IBM MQ can provide access to such SoR applications by using a request-reply pattern. This section explains how a Queue Manager that runs inside zCX can communicate with a Queue Manager that runs on z/OS.

Solution Requirements

Table 8-1 shows how the use of IBM MQ can meet the requirements of SoE and SoR integration.

Table 8-1 Project Requirements

Requirement	Solution
Ability to invoke CICS/IMS applications from zCX	IBM MQ provides channels that can be created and configured to send messages between applications by using MQ Queues.
High Availability	IBM MQ can be configured to use Queue Sharing Group (QSG) on z/OS. Because applications can connect to any queue manager in a QSG, and as all queue managers in a QSG can access shared queues, client applications are not dependent on the availability of a specific queue manager.
High Scalability	The use of a QSG enables a scalable solution that takes advantage of the resources across the IBM Parallel Sysplex®. New instances of CICS regions / IMS regions/ or queue managers can be easily introduced into the QSG as business growth dictates.
Workload Management	Both zCX and IBM MQ can be managed by WLM.

8.1.2 Architecture

A zCX instance and an IBM MQ Queue Manager run as started tasks on z/OS. CICS and IMS are also started tasks on z/OS. Communication Server on z/OS is used to establish communication between MQ that runs inside the container and MQ that runs on z/OS.

Figure 8-1 on page 189 shows how applications inside zCX communicate with other z/OS applications through the Communication Server.

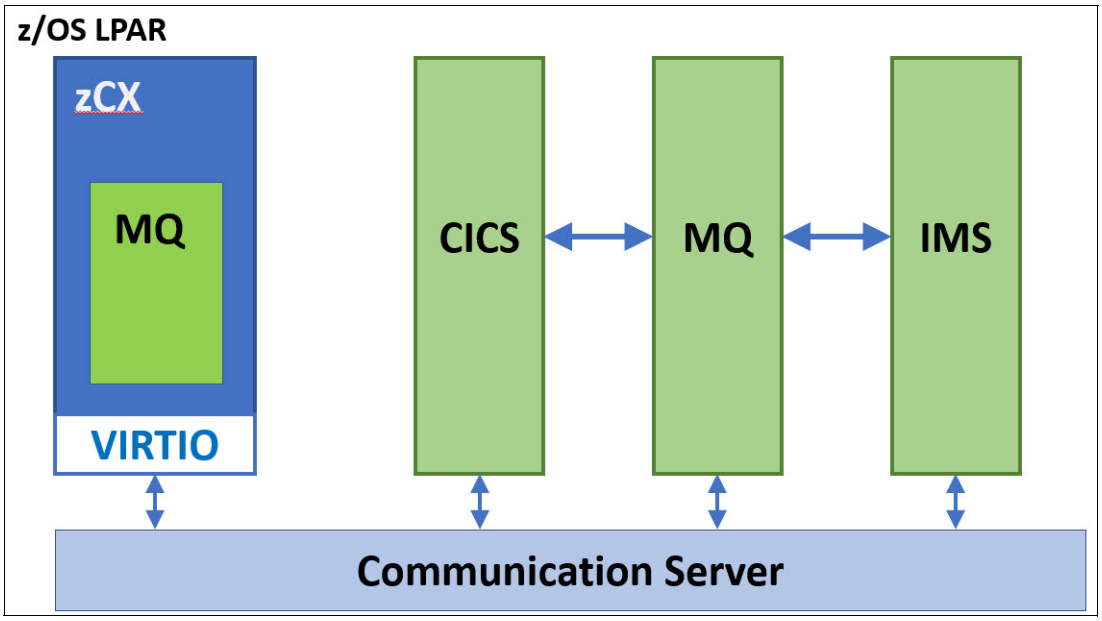


Figure 8-1 Communication between zCX containers and z/OS

8.1.3 Scenario

We discuss an MQ integration scenario where two Queue Managers communicate with each other through messages on a Queue. For simplicity, we do not build the producer and consumer applications that put and get messages from a queue. CICS and IMS applications can both be producers and consumers of messages. Cloud Native applications that run inside a zCX container can also act as either consumers and/or producers. For the scenario, we do not build the consumer and producer applications. We use the tools that MQ provides to put and get messages from MQ Queues. Figure 8-2 on page 190 shows the infrastructure that supports communication between the Queue Managers that we build.

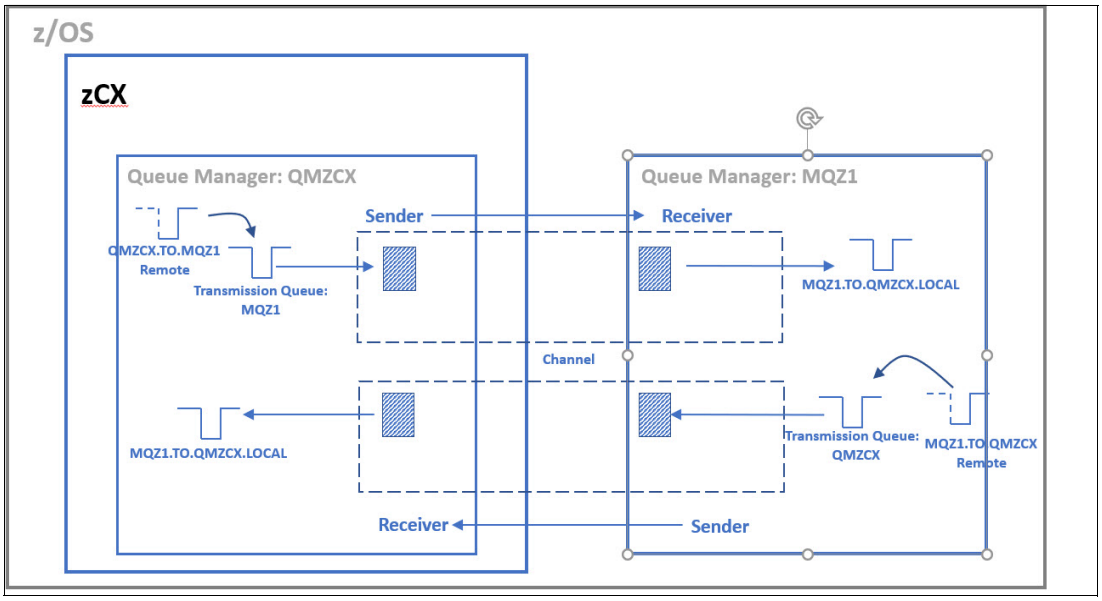


Figure 8-2 Scenario infrastructure

8.1.4 Implementation

In this section, we discuss what needs to be built and tested for the scenario in Figure 8-2.

- Download an IBM MQ image from dockerhub inside zCX.
- Create a IBM MQ Queue Manager container on zCX.
- Create MQ Channels for communication between the z/OS Queue Manager and zCX Queue Manager.
- Test the communication between zCX Queue Manager and z/OS Queue Manager.

Download IBM MQ Advanced image

We download the IBM MQ Advanced image from the dockerhub repository. The following steps were performed:

- Navigate to dockerhub IBM MQ Advanced image with the following link: https://hub.docker.com/_/ibm-mq-advanced

  The display looks similar to Figure 8-3 on page 191:

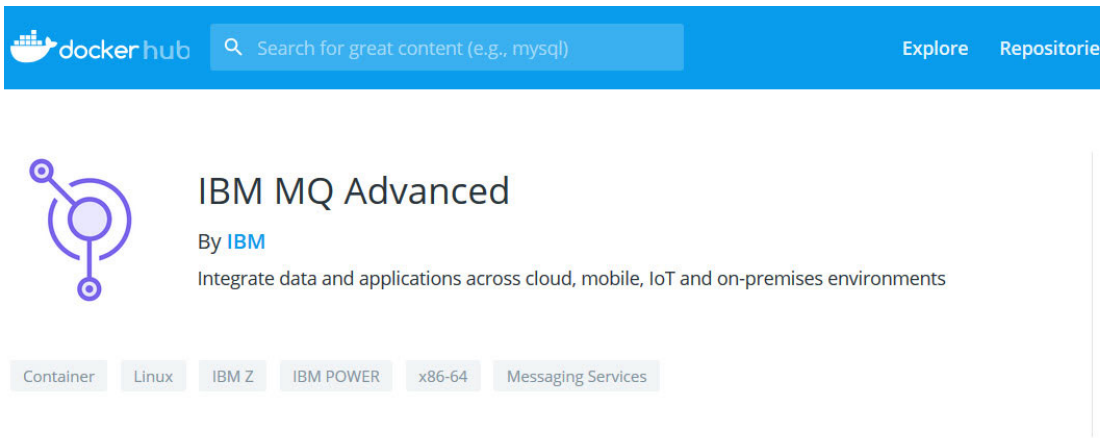


Figure 8-3 IBM MQ Advanced Image on dockerhub

- Follow the steps to get the IBM MQ Advanced image. Copy the following command as shown in Example 8-1. This example pulls version 9.1.0.0, which is specified as a tag in the command.

Example 8-1 Pull IBM MQ from dockerhub

```
docker pull store/ibmcorp/mqadvanced-server-dev:9.1.0.0
```

- Use your SSH client to connect to your zCX instance that was previously provisioned.
  - Use the userid that you used to run the SSH key generation to log on to USS with an SSH client tool (PuTTY or something similar). In this example, we created the SSH keys with the ZCXADM5 userid. So, we must log on with that ZCXADM5 userid.
  - Example 8-2 shows a command that connects to the zCX Docker CLI, where 129.40.23.76 is the IP address of the zCX instance and 8022 is the listener port.

Example 8-2 Log in as admin to zCX

```
ssh admin@129.40.23.76 -p 8022
```

After successful login, the screen looks as shown Figure 8-4.

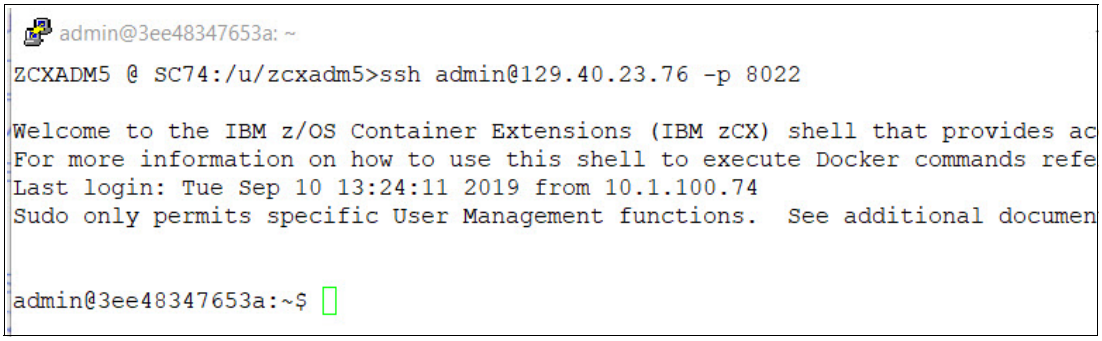


Figure 8-4 Succesful Login

- Use the following Docker commands to pull the IBM MQ Advanced V 9.1.0.0 image from dockerhub.

– Log in to dockerhub with your Docker credentials with the command shown in Example 8-3.

Example 8-3 Login to docker

```
docker login
```

– Pull IBM MQ Advanced image with the command that is shown in Example 8-4.

Example 8-4 Docker pull command

```
docker pull store/ibmcorp/mqadvanced-server-dev:9.1.0.0
```

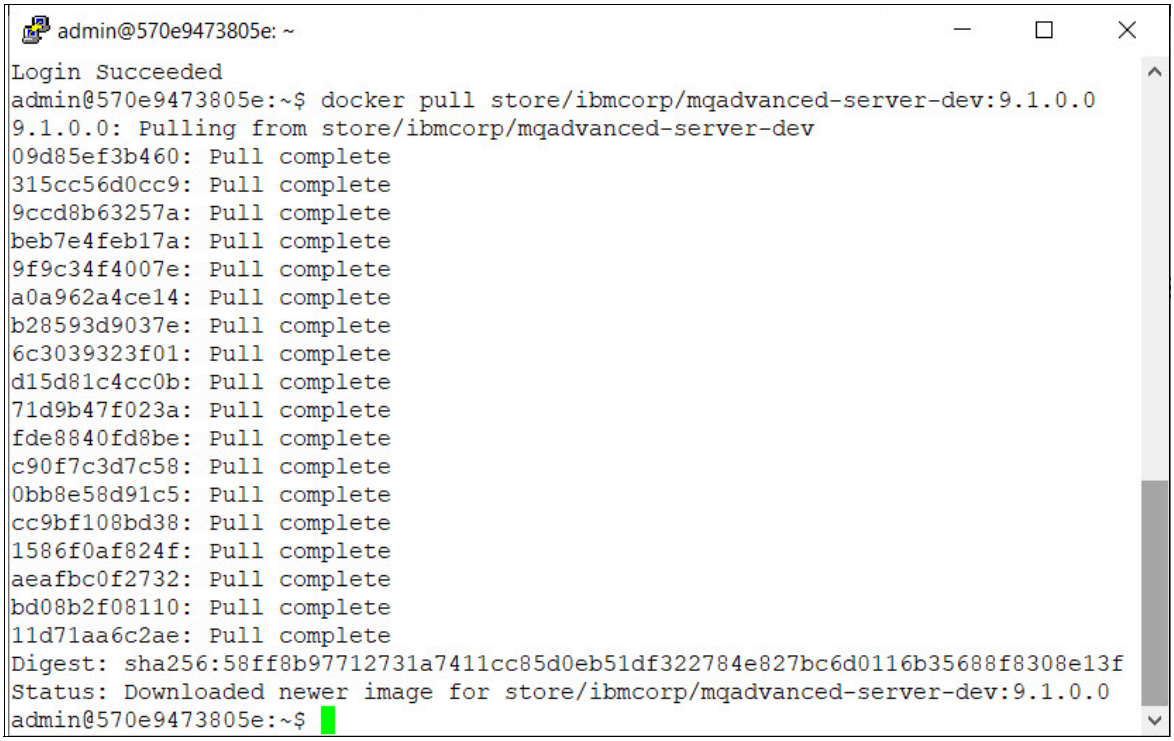


Figure 8-5 MQ Advanced Image Pull

Create Queue Manager inside the container

This section shows the Docker commands that were used to create a Queue Manager QMZCX that runs as a container inside the zCX instance. Specifically, a Docker volume and a Queue Manager named QMZCX had to be created.

Create a docker volume

As a first step, a new Docker volume is created to ensure data persistence. For more information on persistent storage, see chapter 10.2, “Using Docker volumes for persistent data” on page 235. When an image fails, it must be removed or an additional one must be started. A Docker volume simplifies data sharing between those images that run in the same zCX instance. For this example, a volume with the name **mqdataqmzcx** is created by issuing the following command:

```
docker volume create mqdataqmzcx
```

Example 8-5 on page 193 shows the **docker run** command that creates a new MQ Queue Manager named QMZCX to listen on port 1415, and the Web listener on port 9444 for the MQ

Web console. Here, we are updating with a newer version of IBM MQ V9.1.2.0 by using the tag that is shown.

Example 8-5 Create Queue Manager on zCX

```
docker volume create mqdataqmzcx
docker run \
  --env LICENSE=accept \
  --env MQ_QMGR_NAME=QMZCX \
  --volume mqdataqmzcx:/mnt/mqm \
  --publish 1415:1414 \
  --publish 9444:9443 \
  --detach \
  store/ibmcorp/mqadvanced-server-dev:9.1.2.0
```

Run the **docker ps** command as shown in Example 8-6 to verify that the Docker container has been created and is running:

Example 8-6 Display docker process

```
admin@3ee48347653a:~$ docker ps
CONTAINER ID        IMAGE                                       COMMAND                  NAMES
0c09eade8125        store/ibmcorp/mqadvanced-server-dev:9.1.2.0 "runmqdevserver"         elastic_bassi
fa44e995cf5b        store/ibmcorp/mqadvanced-server-dev:9.1.2.0 "runmqdevserver"         peaceful_bartik
3ee48347653a        ibm_zcx_zos_cli_image                       "sudo /usr/sbin/sshd…"   ibm_zcx_zos_cli
```

Verify that the Queue Manager is created by logging in to the MQ Web Console. Point the browser to the MQ Web Console by using the URL https://129.40.23.76:9444 where 129.40.23.76 is the IP address that is assigned to the provisioned zCX instance.

Use the following credentials to log in to the MQ Web Console. Those are the default admin user name and password. You can modify those values.

- User Name = admin
- Password = passw0rd

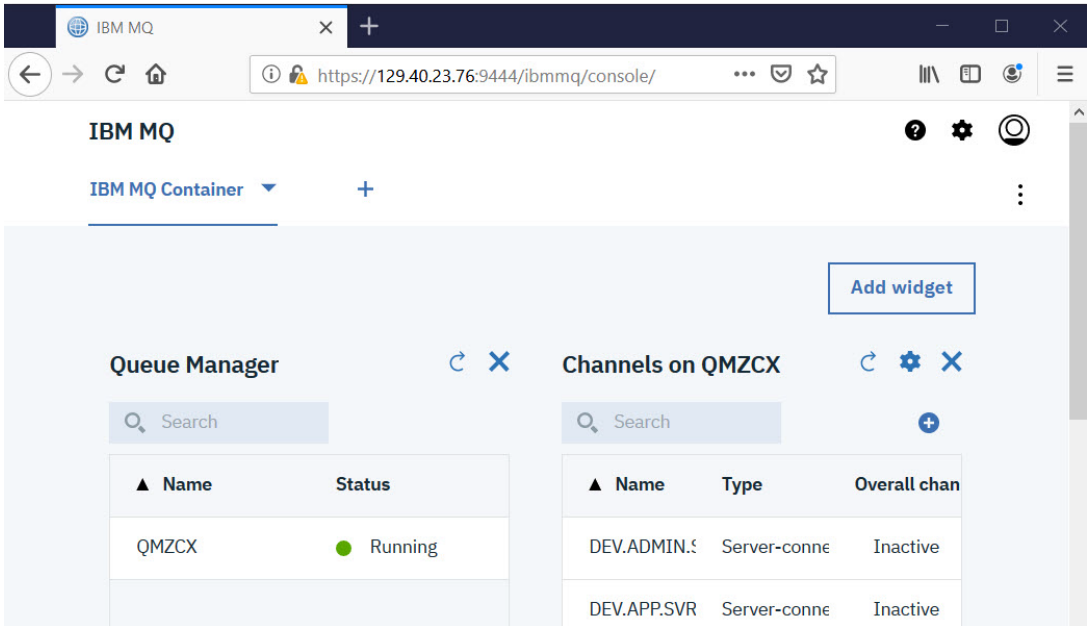


Figure 8-6 MQ Web Console

Create MQ Channels between Queue Manager on zCX and z/OS

A channel is a logical communication link. In IBM MQ, there are two kinds of Channels:

- Message Channel
- MQI Channel

We create Message channels between the Queue Managers. A message channel connects two queue managers. These channels are unidirectional because messages are only sent in one direction along the channel. There are subtypes of message channel, with the different types distinguished by these factors:

- Are they used for receiving or sending messages?
- Do they initiate the communication to the partner queue manager?
- Are they used within a cluster?

A channel definition on one queue manager must be matched with a channel definition of the same name with an appropriate subtype on the partner queue manager.

We define a SENDER channel on one queue manager with an identically named RECEIVER channel on a second queue manager. We also create the appropriate Transmission Queues for the channels. In this document, we do not focus on the MQ commands for creating the channels, because we assume that the reader knows how to create those.

- **Create Message Channels on QMZCX**

  Table 8-2 lists the channels that must be created on Queue Manager QMZCX.

Table 8-2 Sender-Receiver on QMZCX

Channel Name	Type	Conn Name *	Transmission Queue **
QMZCX.MQZ1	Sender	wtsc74.pbm.ihost.com(1414)	MQZ1
MQZ1.QMZCX	Receiver	NA	NA

*** Conn Name:** wtsc74.pbm.ihost.com is the z/OS LPAR where Queue Manager MQZ1 is running and listening on port 1414.
**** Transmission Queue:** A new local queue MQZ1 was also created with the usage type of XMITQ.

- **Create Message Channels on z/OS Queue Manager MQZ1**

  Table 8-3 lists the channels that must be created on Queue Manager MQZ1 on z/OS.

Table 8-3 Sender-Receiver on MQZ1

Channel Name	Type	Conn Name *	Transmission Queue **
QMZCX.MQZ1	Receiver	NA	NA
MQZ1.QMZCX	Sender	129.40.23.76(1415)	QMZCX

*** Conn Name:** 129.40.23.76 is the IP address of the provisioned zCX instance. 1415 is the port that the queue manager QMZCX is listening inside the container.
**** Transmission Queue:** A new local queue QMZCX was also created on Queue Manager MQZ1 with the usage type of XMITQ.

This completes creation of the message channels for Queue Manager communication.

Start SENDER channels, including debugging protocols

Typically, channels are configured to start automatically when messages are available on the associated transmission queue. We start the SENDER channels manually to make sure that

all the channels are running. We start SENDER channels on the Queue Manager QMZCX(Container) and MQZ1(zOS).

- We start the SENDER channel on QMZCX by logging in to the MQ Web Console. Select the SENDER channel QMZCX.MQZ1, start the channel. Then, ensure that it is running.

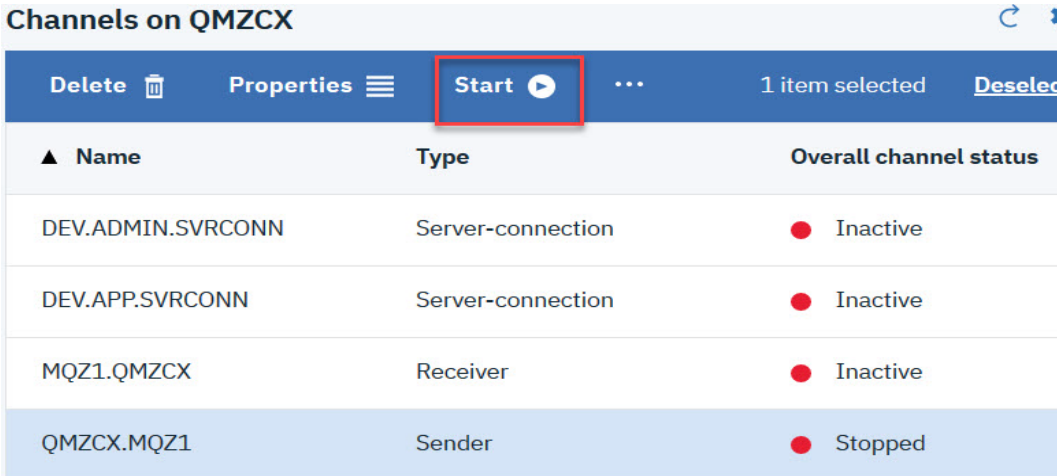


Figure 8-7 Channels on QMZCX

- We start SENDER channel on MQZ1 from MQ Explorer. Select the SENDER channel MQZ1.QMZCX and start the channel to ensure that it is running. The channel does not Start and we get an error. We then debug the error and fix it.
  - On the z/OS Queue Manager, PING the channel MQZ1.QMZCX and look at the results in the channel initiator address space. On the z/OS Console, run the following command to ping the channel as shown in Example 8-7.

Example 8-7 Ping Channel

```
-MQZ1 ping chl(MQZ1.QMZCX )
```

  - The following error is reported on the PING command as shown in Example 8-8. The error indicates that the remote channel is not available. This might occurr because the remote queue manager cannot be reached due to security settings.

Example 8-8 Output of Ping Channel

```
-MQZ1 ping chl(MQZ1.QMZCX)
CSQM134I -MQZ1 CSQMPCHL  PING CHL(MQZ1.QMZCX) COMMAND ACCEPTED
CSQX558E -MQZ1 CSQXPING Remote channel MQZ1.QMZCX not available
CSQ9023E -MQZ1 CSQXCRPS ' PING CHL' ABNORMAL COMPLETION
```

  - We look at the security settings on the container Queue Manager QMZCX. We must run the **docker exec** command to get inside the container. There, we execute IBM MQ commands against the queue manager.
    - Find the process ID of the container by running the **docker ps** command as shown in Example 8-9.

Example 8-9 Get process id of MQ container

```
admin@3ee48347653a:~$ docker ps
CONTAINER ID   IMAGE                                        COMMAND                  CREATED        STATUS
PORTS                                                      NAMES
0c09eade8125   store/ibmcorp/mqadvanced-server-dev:9.1.2.0  "runmqdevserver"         4 hours ago    Up 4 hours
9157/tcp, 0.0.0.0:1415->1414/tcp, 0.0.0.0:9444->9443/tcp   elastic_bassi
```

```
fa44e995cf5b   store/ibmcorp/mqadvanced-server-dev:9.1.2.0  "runmqdevserver"        4 days ago     Up 32 hours
0.0.0.0:1414->1414/tcp, 0.0.0.0:9443->9443/tcp, 9157/tcp   peaceful_bartik
3ee48347653a   ibm_zcx_zos_cli_image                        "sudo /usr/sbin/sshd…"  5 days ago     Up 33 hours
8022/tcp, 0.0.0.0:8022->22/tcp                              ibm_zcx_zos_cli
```

- Run the **docker exec** command to go inside the container for running MQ commands as shown in Example 8-10.

Example 8-10 Docker exec

```
admin@3ee48347653a:~$ docker exec -it 0c09eade8125 bash
(mq:9.1.2.0)mqm@0c09eade8125:/$
```

- Display Queue Manager QMZCX attributes by running the **runmqsc** command and then the MQSC display command as shown in Example 8-11:

Example 8-11 Display Queue Manager QMZCX

```
(mq:9.1.2.0)mqm@0c09eade8125:/$ runmqsc
5724-H72 (C) Copyright IBM Corp. 1994, 2019.
Starting MQSC for queue manager QMZCX.


dis qmgr all
     1 : dis qmgr all
AMQ8408I: Display Queue Manager details.
   QMNAME(QMZCX)                           ACCTCONO(DISABLED)
   ACCTINT(1800)                           ACCTMQI(OFF)
   ACCTQ(OFF)                              ACTIVREC(MSG)
   ACTVCONO(DISABLED)                      ACTVTRC(OFF)
   ADVCAP(ENABLED)                         ALTDATE(2019-09-10)
   ALTTIME(18.24.20)                       AMQPCAP(NO)
   AUTHOREV(DISABLED)                      CCSID(819)
   CERTLABL(ibmwebspheremqqmzcx)           CERTVPOL(ANY)
   CHAD(DISABLED)                          CHADEV(DISABLED)
   CHADEXIT( )                             CHLEV(DISABLED)
   CHLAUTH(ENABLED)                        CLWLDATA( )
   CLWLEXIT( )                             CLWLLEN(100)
   CLWLMRUC(999999999)                     CLWLUSEQ(LOCAL)
   CMDEV(DISABLED)                         CMDLEVEL(912)
   COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)    CONFIGEV(DISABLED)
   CONNAUTH(DEV.AUTHINFO)                  CRDATE(2019-09-10)
```

- We see that the CHLAUTH is ENABLED to default value and the CONNAUTH is also set to default value. For simplicity, we run the DISABLE CHLAUTH command and allow CONNAUTH to use the ID of the connecting system. We use the **MQSC ALTER QMGR** command to do that, as shown in Example 8-12:

Example 8-12 Alter Queue Manager attributes

```
(mq:9.1.2.0)mqm@0c09eade8125:/$ runmqsc
5724-H72 (C) Copyright IBM Corp. 1994, 2019.
Starting MQSC for queue manager QMZCX.


ALTER QMGR CHLAUTH(DISABLED) CONNAUTH(SYSTEM.DEFAULT.AUTHINFO.IDPWOS)
     1 : ALTER QMGR CHLAUTH(DISABLED)
CONNAUTH(SYSTEM.DEFAULT.AUTHINFO.IDPWOS)
AMQ8005I: IBM MQ queue manager changed.
```

- In MQ Explorer, select the SENDER channel MQZ1.QMZCX and start the channel as shown in Figure 8-8. It should go into a running state.

  This completes the configuration of the channels for Queue Manager Communication.

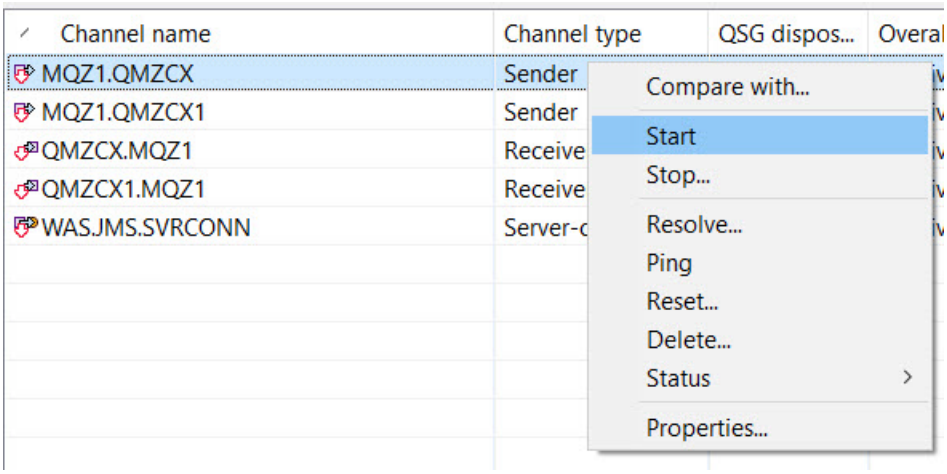


Figure 8-8 Start Sender Channel on MQZ1

Test the message flow both ways

We started the SENDER-RECEIVER channel pairs on both the Queue Managers in the previous step. We now test the infrastructure by putting messages in a Queue on one Queue Manager and getting messages on the destination Queue.

Table 8-4 shows the queues that we have created for sending messages from z/OS to zCX.

Table 8-4 From z/OS to zCX

Remote Queue on z/OS (MQZ1)	Local Queue on zCX Container (QMZCX)
MQZ1.TO.QMZCX.REMOTE	MQZ1.TO.QMZCX.LOCAL

Table 8-5 shows the queues that we have created for sending messages from zCX to z/OS.

Table 8-5 From zCX to z/OS

Remote Queue on zCX Container (QMZCX)	Local Queue on z/OS (QMZ1)
QMZCX.TO.MQZ1.REMOTE	QMZCX.TO.MQZ1.LOCAL

► **Send messages from z/OS to zCX**

a. In MQ Explorer, select the remote queue MQZQ.TO.QMZCX and put a test message on the Queue as shown in Figure 8-9 on page 198.

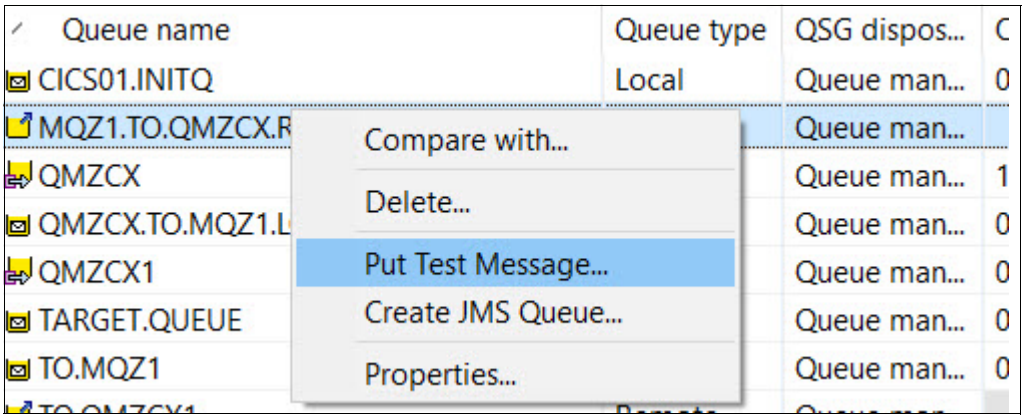


Figure 8-9 Put Test Message on QMZCX1

b. Enter the test message in the **Message data** field, as shown in Figure 8-10 and click **Put Message**.

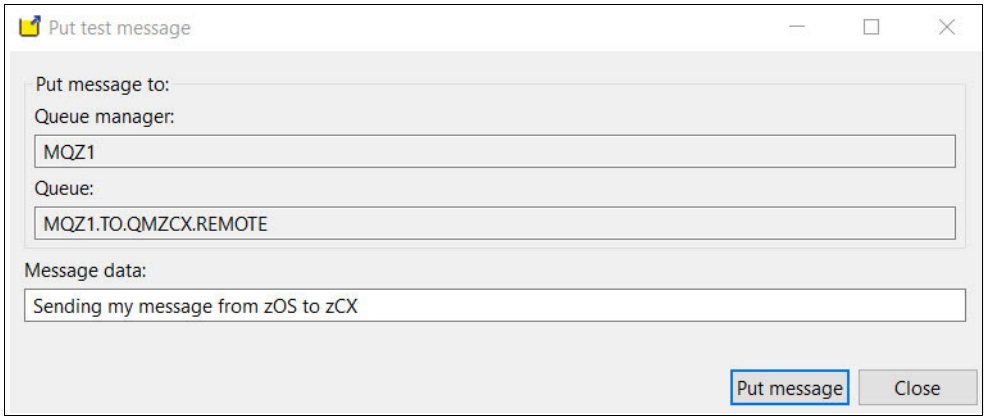


Figure 8-10 Put Message content

c. Point to the MQ Web Console that runs the MQ container. Select the MQZ1.TO.QMZCX.LOCAL queue, and browse it as shown in Figure 8-11.

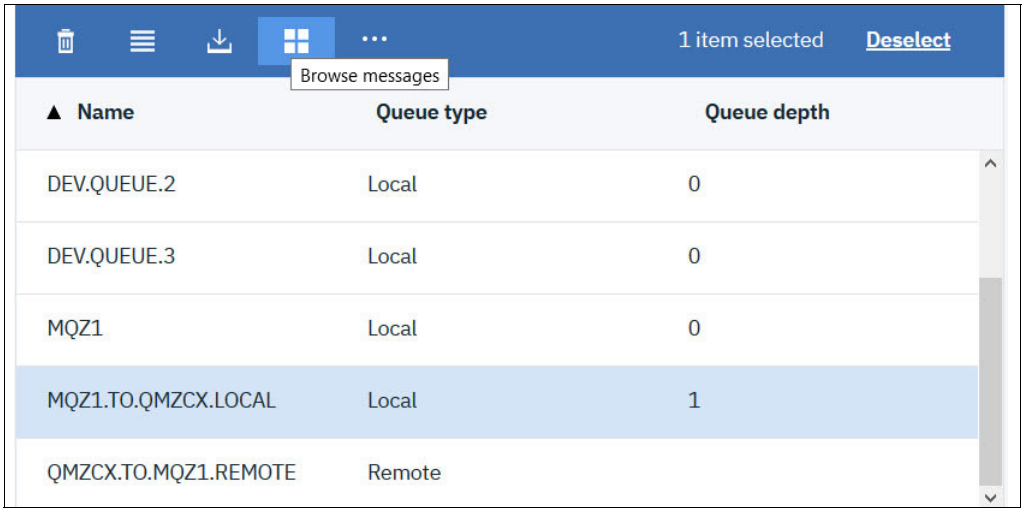


Figure 8-11 Browse message on QMZCX

Figure 8-12 shows the message that is displayed in the browser.

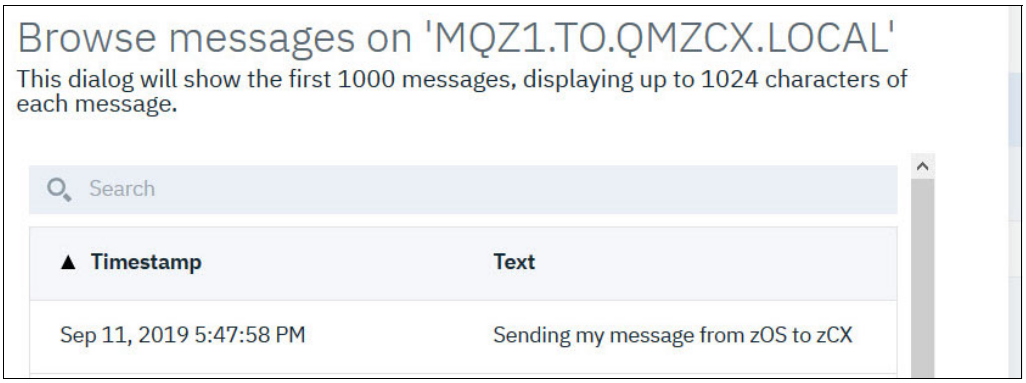


Figure 8-12 Content of browsed message

- **Send messages from zCX to z/OS**
  - Perform similar steps by putting a message on the Remote Queue QMZCX.TO.MQZ1.REMOTE on QMZCX, which runs inside the container. You can view the message in a web browser by using MQ Explorer on the Local Queue QMZCX.TO.MQZ1.LOCAL on Queue Manager MQZ1.

8.2 Accessing Db2 from within Docker container when you are using Jupyter Notebook

This section describes the steps required to get a Jupyter server running as an image in a zCX instance that accesses Db2 data via a JDBC Connection.

8.2.1 What is a Jupyter Notebook

A Jupyter Notebook is an interactive document that can execute code. Notebook code can be based on either Python, Julia, or R. Document output can be exported in either HTML or PDF format to any desired location. It works in a Client-Server setup. The following sections describe how to install and set up the Server part in a zCX instance.

You can enrich Jupyter Notebooks by using available packages to create graphs or to perform data analysis. For more information on Jupyter, see `https://jupyter.org/`

8.2.2 Scenario

An end user can use a web browser to access a Jupyter Server instance that is hosted in a z/OS Container Extension Docker image. Within the Jupyter Server, the end user creates a Jupyter Notebook. From that Jupyter Notebook, the end user establishes a connection to a Db2 subsystem to receive data through the use of an SQL query that is passed in the JDBC connection.

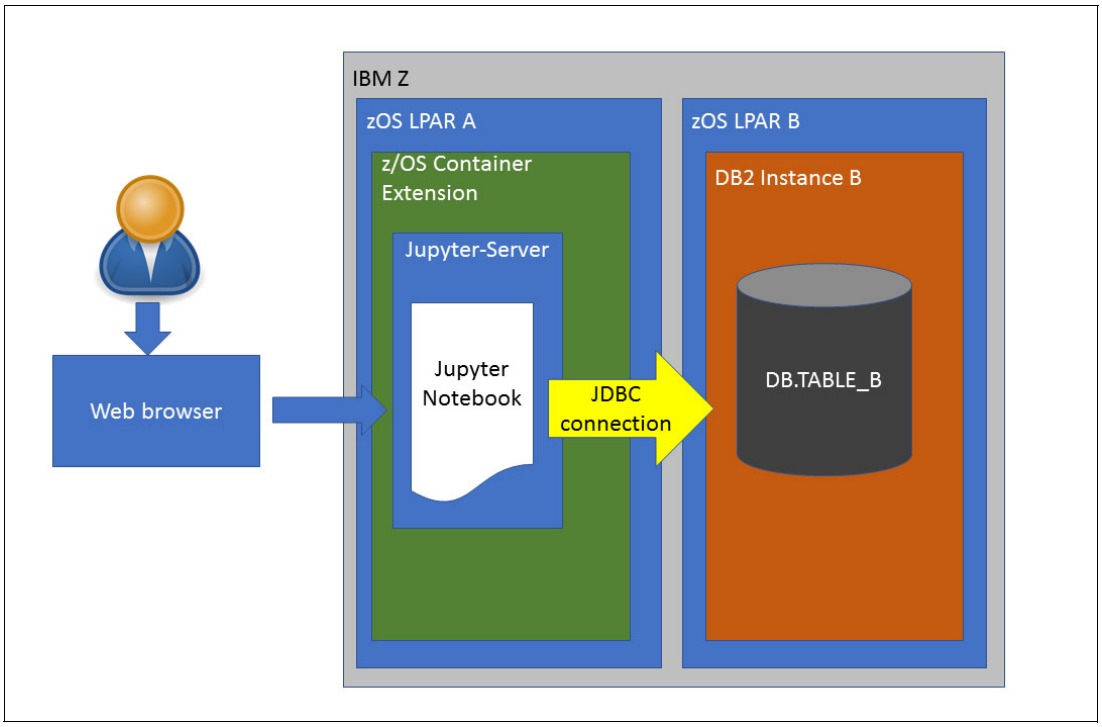


Figure 8-13 Scenario for the Db2 connection

8.2.3 Creating a Docker Image with Jupyter installed

Create a Docker volume

As a first step, a new Docker volume is created to ensure data persistence. The reason for using a Docker volume is to externalize the data from the container. If you do not use a volume, the data is stored within the container. If the container for some reason failed and was unable to be restarted, you have no way to recover data that is stored inside the container. Also, it is not possible to share this data with any other container.

For this example, a volume with the name **jupyter-data** is created by issuing the following command:

```
docker volume create jupyter-data
```

Further information about data persistence and volumes, see Chapter 10, “Persistent data” on page 231.

Create a Docker image based on Ubuntu

At this stage of the installation, the Python environment is created. That activity requires an s390x-based Ubuntu and Python (including Python packages) and also the jdbc drivers.

Before the image can be built, a few files must be created and transferred.

Get the JDBC driver ready

Important Note: Use **sftp** to transfer binary-type files between z/OS and the SSH container in a zCX instance. If you use scp, scp on z/OS detects that the zCX instance is ASCII-based and does EBCDIC-to-ASCII translation. You can use scp to transfer text-based files.

First, the jdbc drivers must be transferred into the zCX instance.

Example 8-13 shows how we used sftp to binary-transfer the Db2 JAR files from the USS directory on z/OS into the zCX instance. Before we issued the sftp command, we created the following target path in the ssh container in the zCX instance: **home/admin/db2jars**

You run the sftp command from the preceding directory in the zCX instance.

Example 8-13 Transfer jdbc driver with sftp

```
admin@33825a277992:~$ sftp zcxadm6@wtsc74.pbm.ihost.com:/pp/db2v12/base/d190904/jdbc/classes
The authenticity of host 'wtsc74.pbm.ihost.com (129.40.23.1)' can't be established.
RSA key fingerprint is SHA256:v27XurOE1iVOhnVzzsL4a3X97F9bHOMcumDxFpE3Ogc.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'wtsc74.pbm.ihost.com,129.40.23.1' (RSA) to the list of known hosts.
zcxadm6@wtsc74.pbm.ihost.com's password:
Connected to wtsc74.pbm.ihost.com.
Changing to: /pp/db2v12/base/d190904/jdbc/classes
sftp> get *.jar
Fetching /pp/db2v12/base/d190904/jdbc/classes/db2jcc.jar to db2jcc.jar
/pp/db2v12/base/d190904/jdbc/classes/db2jcc.jar
100% 3725KB  71.0MB/s   00:00
Fetching /pp/db2v12/base/d190904/jdbc/classes/db2jcc4.jar to db2jcc4.jar
/pp/db2v12/base/d190904/jdbc/classes/db2jcc4.jar
100% 4142KB  77.6MB/s   00:00
Fetching /pp/db2v12/base/d190904/jdbc/classes/db2jcc_javax.jar to db2jcc_javax.jar
/pp/db2v12/base/d190904/jdbc/classes/db2jcc_javax.jar
100%   23KB  23.2MB/s   00:00
Fetching /pp/db2v12/base/d190904/jdbc/classes/db2jcc_license_cisuz.jar to db2jcc_license_cisuz.jar
/pp/db2v12/base/d190904/jdbc/classes/db2jcc_license_cisuz.jar
100% 2755     5.6MB/s   00:00
sftp> quit
```

Prepare the Dockerfile

Second, you create the Dockerfile to use during the build of the image. Create this file in the **/home/admin** directory in the zCX instance and name it **dockerfile-base**.

Example 8-14 /home/admin/dockerfile-base

```
ARG BASE_CONTAINER=s390x/ubuntu
FROM $BASE_CONTAINER
# Define the label for this image
LABEL maintainer="IBM Redbook"

# Switch to root user for the following steps
USER root
# Install all OS dependencies for notebook server that starts but lacks all features.
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update && apt-get -yq dist-upgrade \
     && apt-get install -yq --no-install-recommends \
     wget \
     bzip2 \
     ca-certificates \
     sudo \
     locales \
     git \
     default-jre \
     vim \
     curl \
     fonts-liberation \
```

```
     run-one
# Get the installed code-page
RUN echo "en_US.UTF-8 UTF-8" > /etc/locale.gen && \
locale-gen
# Configure environment
ENV SHELL=/bin/bash \
LC_ALL=en_US.UTF-8 \
LANG=en_US.UTF-8 \
LANGUAGE=en_US.UTF-8

ENV PYTHON_PACKAGE_NAME="python"
ENV PYTHON_PACKAGE_VERSION="3.7.4"
#Configure and build and install Python3
RUN apt-get install -y gcc g++ libbz2-dev libdb-dev libffi-dev libgdbm-dev liblzma-dev
libncurses-dev libreadline-dev libsqlite3-dev libssl-dev make tar tk-dev uuid-dev wget
xz-utils zlib1g-dev \
     && wget
"https://www.python.org/ftp/${PYTHON_PACKAGE_NAME}/${PYTHON_PACKAGE_VERSION}/Python-${PYTHO
N_PACKAGE_VERSION}.tgz" \
     && tar -xzvf "Python-${PYTHON_PACKAGE_VERSION}.tgz" \
     && rm "Python-${PYTHON_PACKAGE_VERSION}.tgz" \
     && cd "Python-${PYTHON_PACKAGE_VERSION}" \
     && ./configure \
     && make \
     && make install \
     && rm -rf /var/lib/apt/lists/*

RUN python3 -m pip install --upgrade pip \
&& python3 -m pip install notebook \
     jupyterlab \
     plotly \
     numpy \
     pandas \
     ipywidgets \
     jaydebeapi \
     ibm_db \
     matplotlib \
&& python3 -m pip install JPype1==0.6.3 --force-reinstall
RUN jupyter notebook --generate-config

# Copy all files from local zCX db2jars folder into the ubuntu images /db2jars folder
RUN mkdir /db2jars
ADD db2jars /db2jars
# Create a new group and user to run the notebook server
RUN groupadd jupyter-group &&  useradd -ms /bin/bash -G jupyter-group jupyter-user
USER jupyter-user
RUN mkdir /home/jupyter-user/notebooks
ENTRYPOINT
["jupyter","notebook","--ip=0.0.0.0","--port=8888","--allow-root","--notebook-dir=/home/jup
yter-user/notebooks"]
```

What the dockerfile-base file does

The first part of the Dockerfile uses the FROM command to specify that the image must be based on an Ubuntu image.

Example 8-15 Define base properties for the Dockerfile

```
ARG BASE_CONTAINER=s390x/ubuntu
FROM $BASE_CONTAINER
```

```
# Define the label for this image
LABEL maintainer="IBM Redbooks"
```

In the next step, we use **root** user ID to install additional products in the base Ubuntu system. Then, we set the environment variables for codepages and the shell.

Example 8-16 Enrich the base ubuntu with additional software programs.

```
# Switch to root user for the following steps
USER root
# Install all OS dependencies for notebook server that starts but lacks all
# features
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update && apt-get -yq dist-upgrade \
     && apt-get install -yq --no-install-recommends \
     wget \
     bzip2 \
     ca-certificates \
     sudo \
     locales \
     git \
     default-jre \
     vim \
     curl \
     fonts-liberation \
     run-one
# Get the installed code-page
RUN echo "en_US.UTF-8 UTF-8" > /etc/locale.gen && \
locale-gen
# Configure environment
ENV SHELL=/bin/bash \
LC_ALL=en_US.UTF-8 \
LANG=en_US.UTF-8 \
LANGUAGE=en_US.UTF-8
```

The next part of the Dockerfile installs Python and related packages that Python requires, by using the **pip** command.

In this case, the installation of Python relies on the availability of the tgz file that is hosted on http://www.python.org. The file is downloaded from their FTP directory. The downloaded file is then extracted and the installation is performed with the **make install** command.

Example 8-17 Installing Python

```
ENV PYTHON_PACKAGE_NAME="python"
ENV PYTHON_PACKAGE_VERSION="3.7.4"
#Configure and build and install Python3
RUN apt-get install -y gcc g++ libbz2-dev libdb-dev libffi-dev libgdbm-dev liblzma-dev libncurses-dev
libreadline-dev libsqlite3-dev libssl-dev make tar tk-dev uuid-dev wget xz-utils zlib1g-dev \
     && wget
"https://www.python.org/ftp/${PYTHON_PACKAGE_NAME}/${PYTHON_PACKAGE_VERSION}/Python-${PYTHON_PACKAGE_VERSIO
N}.tgz" \
     && tar -xzvf "Python-${PYTHON_PACKAGE_VERSION}.tgz" \
     && rm "Python-${PYTHON_PACKAGE_VERSION}.tgz" \
     && cd "Python-${PYTHON_PACKAGE_VERSION}" \
     && ./configure \
     && make \
     && make install \
     && rm -rf /var/lib/apt/lists/*
```

Installation of further Python packages comes next, to extend the Python capabilities. When Docker installs the **jaydebeapi** package this pulls in V0.7.0 of the JPype1 package. However, there is an issue with this version, so we use a command to get the V0.6.3 version of JPype1 package.

Not all of the installed packages are required for the base jdbc connection. However, they enable you to create graphs and perform further data analysis tasks. The last line is generates a default jupyter configuration file for reference. It won‘t be used in this scenario, because all the required settings are passed as arguments during the start of the Jupyter Notebook.

Example 8-18 Install additional Python packages

```
RUN python3 -m pip install --upgrade pip \
&& python3 -m pip install notebook \
     jupyterlab \
     plotly \
     numpy \
     pandas \
     ipywidgets \
     jaydebeapi \
     ibm_db \
     matplotlib \
&& python3 -m pip install JPype1==0.6.3 --force-reinstall
RUN jupyter notebook --generate-config
```

The last section is the creation of a folder to hold the Db2 JAR files. They are copied from the **/home/admin/db2jars** folder of the ssh container of the zCX instance into the Ubuntu image to the folder created as the first command (**RUN mkdir /db2jars**).

Then, a new group and user are created. This non-root user ID is used so that the Jupyter Notebook server does not run as root in the container.

In the home directory of that user, a folder called **notebooks** is created for use as the mount point for the Docker volume that was created in the first step. The final statement of the Dockerfile contains the startup command for the Notebook server.

Use of the value 0.0.0.0 for TCPIP address means that the Jupyter notebook server can accept requests from any TCPIP address, rather than defaulting to an internal localhost type address. Use of the internal address would prevent the acceptance of requests from outside the container.

For the port, you can choose any port that not yet used within the zCX instance.

Example 8-19 Prepare the Notebook start

```
# Copy all files from local zCX db2jars folder into the ubuntu images /db2jars folder
RUN mkdir /db2jars
ADD db2jars /db2jars
# Create a new group and user to run the notebook server
RUN groupadd jupyter-group &&  useradd -ms /bin/bash -G jupyter-group jupyter-user
USER jupyter-user
RUN mkdir /home/jupyter-user/notebooks
ENTRYPOINT
["jupyter","notebook","--ip=0.0.0.0","--port=8888","--allow-root","--notebook-dir=/home/jup
yter-user/notebooks"]
```

Build the image

Third, the Docker image can now be built based on files that were just created. To ensure that the Docker build works as described in this chapter, a connection to the internet is required because all packages must be received from the internet. Alternatively, you can make them available in a local registry.

More details about a local registry can be found Chapter 6, “Private registry implementation” on page 107.

Execute the following command to build the Docker image:

```
docker build /home/admin --tag jupyter-server:latest -f dockerfile-base
```

It can take a while to build the image, depending on your internet bandwidth and the processor capacity that is available to build the image.

Verify the installation

To verify that the installation worked as expected, you can run the newly created image, log on to it using a web browser, and run an SQL query. Example 8-20 shows the commands that do this.

The first command starts the image, names it as *jupyter-server-instance-1*, and maps to the Docker volume that we created in the very first step. Also, it forwards requests that are made to port 8888 to the Jupyter Server. That server is listening on the same port that was defined in the ENTRYPOINT statement from **dockerfile-base**.

Example 8-20 Run the Jupyter server

```
admin@33825a277992:~$ docker run -it --name jupyter-server-instance-1 -v
jupyter-data:/home/jupyter-user/notebooks:rw -p 8888:8888 -d jupyter-server
87ce6ff2b86f8532ef76d41c2ddec8139c0cc1d904c340d37e9ac292d1e3a045
```

When the image is started, you must get the current token from the Jupyter Server to log in. To get the token, you must check the log from the running *jupyter-server-instance-1*. To do so, issue the **docker logs jupyter-server-instance-1** command.

> **Note:** A Jupyter container that is set up in the manner that we describe is not appropriate for supporting multiple users. If your Jupyter environment must support multiple users, configure a Jupyter Hub. The steps to do so are not described within this Redbooks docu ent. However, you can use the approach that is described here as a starting point to which you can add a Jupyter Hub instance.

Example 8-21 Get login data with the generated token

```
admin@33825a277992:~$ docker logs jupyter-server-instance-1
[I 20:03:54.831 NotebookApp] Writing notebook server cookie secret to
/home/jupyter-user/.local/share/jupyter/runtime/notebook_cookie_secret
[I 20:03:55.061 NotebookApp] JupyterLab extension loaded from
/usr/local/lib/python3.7/site-packages/jupyterlab
[I 20:03:55.061 NotebookApp] JupyterLab application directory is
/usr/local/share/jupyter/lab
[I 20:03:55.064 NotebookApp] Serving notebooks from local directory:
/home/jupyter-user/notebooks
[I 20:03:55.064 NotebookApp] The Jupyter Notebook is running at:
[I 20:03:55.064 NotebookApp]
http://b6fb0e92bef1:8888/?token=f5405d923d8295d9194e990e42edf34c8effcc323309ee91
```

```
[I 20:03:55.064 NotebookApp]  or
http://127.0.0.1:8888/?token=f5405d923d8295d9194e990e42edf34c8effcc323309ee91
[I 20:03:55.065 NotebookApp] Use Control-C to stop this server and shut down all
kernels (twice to skip confirmation).
[W 20:03:55.069 NotebookApp] No web browser found: could not locate runnable
browser.
[C 20:03:55.069 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///root/.local/share/jupyter/runtime/nbserver-6-open.html
    Or copy and paste one of these URLs:

http://87ce6ff2b86f:8888/?token=f5405d923d8295d9194e990e42edf34c8effcc323309ee91
     or
http://127.0.0.1:8888/?token=f5405d923d8295d9194e990e42edf34c8effcc323309ee91
```

On the last line, you get the generated token. This information must now be combined with the DNS that is defined for the zCX instance where this Jupyter Server is running. In our case, the link to access the Jupyter Server would be as follows:

```
http://sc74cn11.pbm.ihost.com:8888/?token=f5405d923d8295d9194e990e42edf34c8effcc323309ee91
```

Now, the setup and configuration of a base Jupyter Server is complete.

8.2.4 Using a Jupyter Notebook to access Db2

Paste the generated link from the previous step into a browser of your choice. The Jupyter Server shows the directory structure of the Ubuntu container as shown in Figure 8-14.

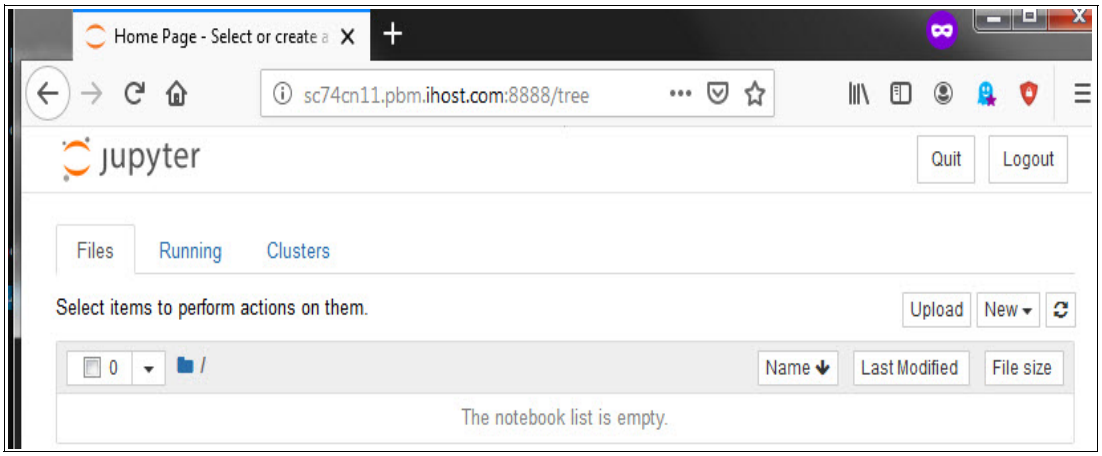


Figure 8-14 Root directory of the Jupyter notebook server

Navigate to the **notebooks** folder in the root directory and create an initial notebook there. It is important to create the notebook in that folder, because this is the folder that is backed by the Docker volume. So, in case the instance that is currently running is shut down and a new instance is started, your Notebook files are still there and accessible. To create a new Notebook, click the **New** menu, and select **Python 3**, as shown in Example 8-15 on page 207.

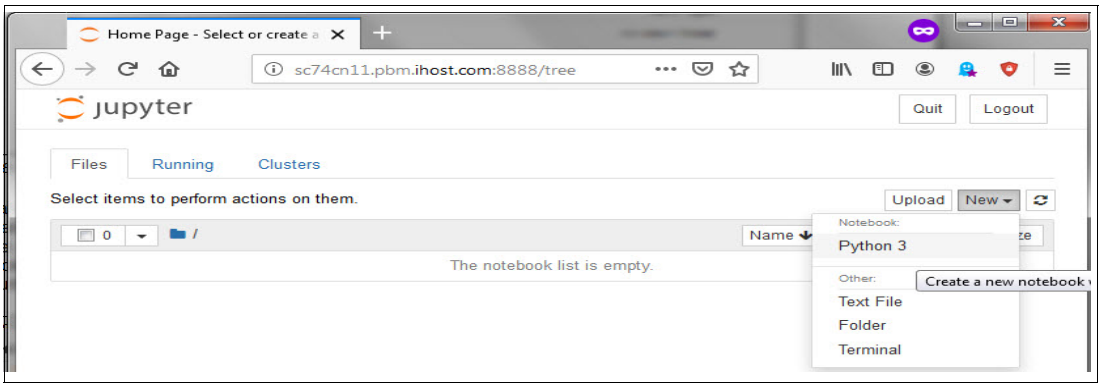


Figure 8-15 Create a new Notebook file

When the file is created, give it a meaningful name, for example **`DB2 Access with JayDeBeAPI`**. When you click in the **`Untitled`** title of the notebook, a rename dialog is displayed, as shown in Figure 8-16.

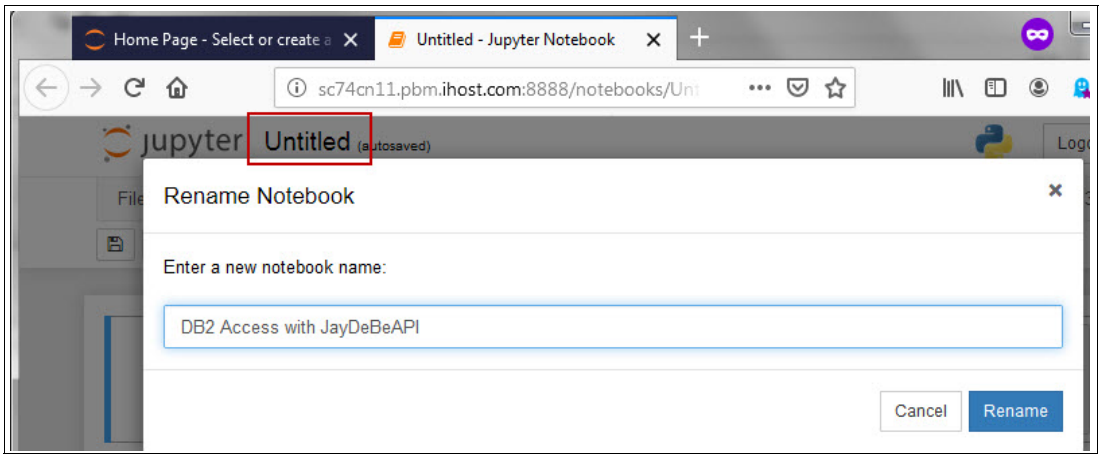


Figure 8-16 Rename the notebook.

After the notebook file is created and a meaningful name is given to the file, we start to add content that allows access to a Db2 table via JDBC. Add the same content as shown in Example 8-17 on page 208.

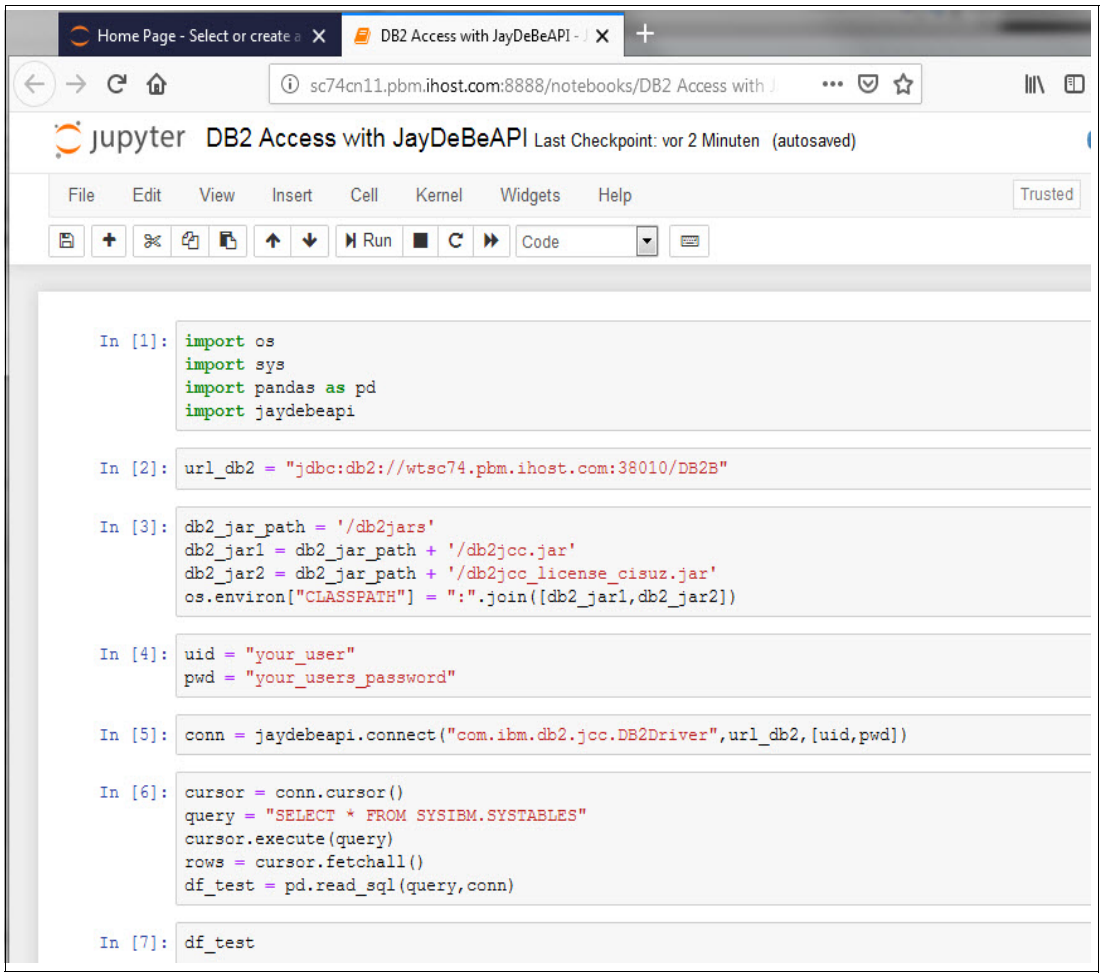


Figure 8-17 Notebook to access Db2 via JDCBC

When the Notebook is ready, you can *Run all* cells at once as shown in Example 8-18 on page 208. This action automatically executes each cell consecutively.

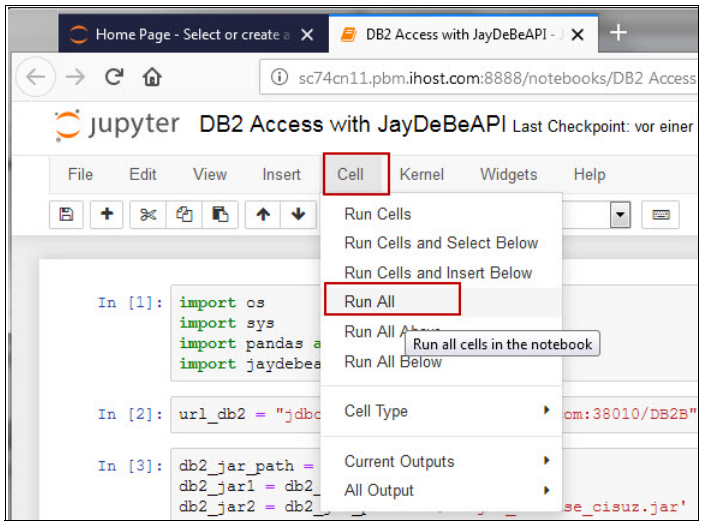
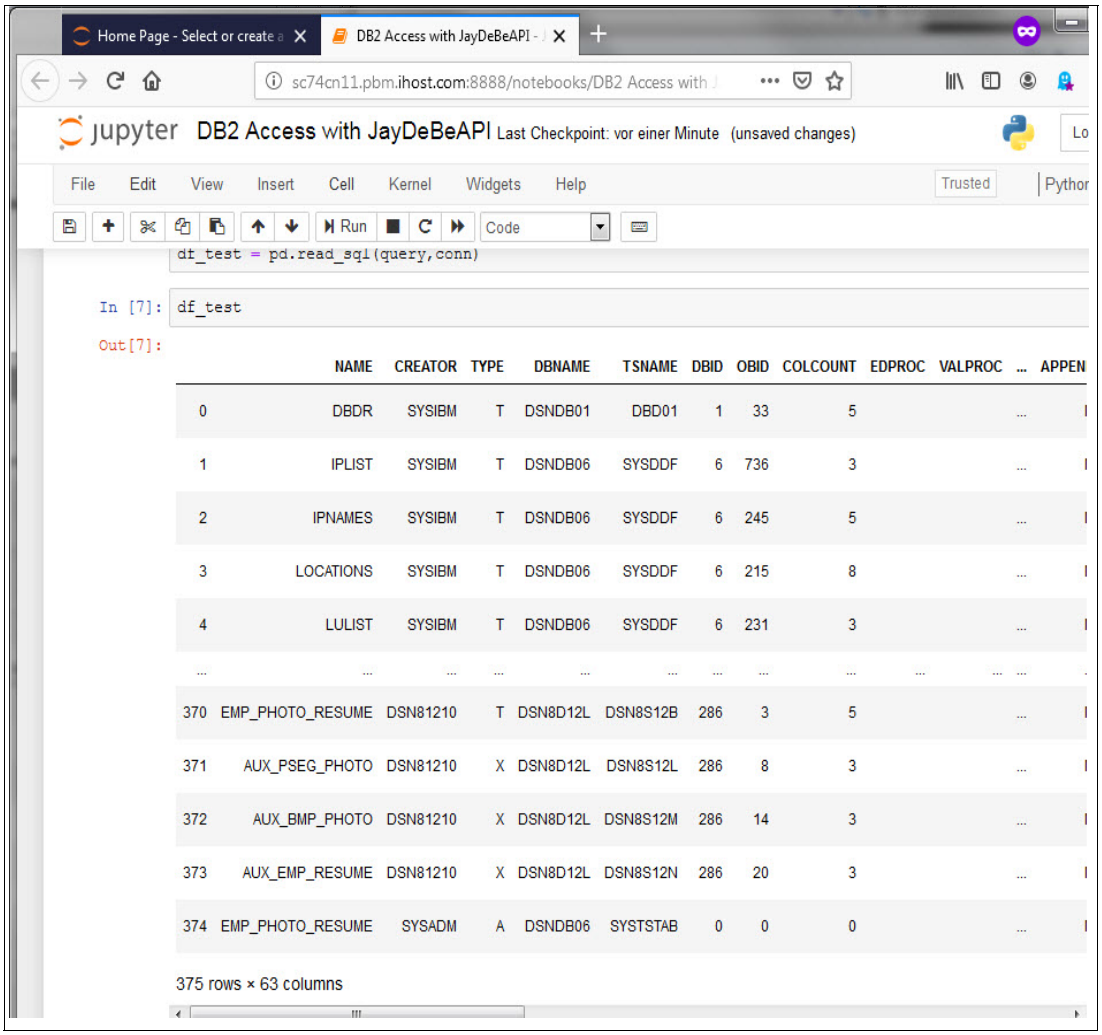


Figure 8-18 Run all cells in a Jupyter Notebook

Execution stops if there is an issue in one of the cells. When there are no issues, the output of the SQL statement is displayed and looks similar to what in shown in Figure 8-19.



	NAME	CREATOR	TYPE	DBNAME	TSNAME	DBID	OBID	COLCOUNT	EDPROC	VALPROC	...	APPEN
0	DBDR	SYSIBM	T	DSNDB01	DBD01	1	33	5			...	I
1	IPLIST	SYSIBM	T	DSNDB06	SYSDDF	6	736	3			...	I
2	IPNAMES	SYSIBM	T	DSNDB06	SYSDDF	6	245	5			...	I
3	LOCATIONS	SYSIBM	T	DSNDB06	SYSDDF	6	215	8			...	I
4	LULIST	SYSIBM	T	DSNDB06	SYSDDF	6	231	3			...	I
...	...	...	...	...	...	...	...	...	...	...	...	.
370	EMP_PHOTO_RESUME	DSN81210	T	DSN8D12L	DSN8S12B	286	3	5			...	I
371	AUX_PSEG_PHOTO	DSN81210	X	DSN8D12L	DSN8S12L	286	8	3			...	I
372	AUX_BMP_PHOTO	DSN81210	X	DSN8D12L	DSN8S12M	286	14	3			...	I
373	AUX_EMP_RESUME	DSN81210	X	DSN8D12L	DSN8S12N	286	20	3			...	I
374	EMP_PHOTO_RESUME	SYSADM	A	DSNDB06	SYSTSTAB	0	0	0			...	I

Figure 8-19 Result set returned from Db2

At this stage, you have a running Jupyter Server instance that allows access to Db2 through a JDBC connection.

8.3 Accessing application in a zCX container from z/OS

In this section, we describe how an application that runs in native z/OS can access an application that is running in a Docker container that runs in a zCX instance.

8.3.1 Target application in Docker container

To demonstrate access to an application in a Docker container, we chose to use the **etcd** open source product. **Etcd** is used to store key/value pairs of information in a reliable and consistent way. Information about **etcd** can be found here: `https://etcd.io/`

8.3.2 Setting up the etcd container

We used the following Docker image of **etcd** that had previously been created for Linux on Z.

https://hub.docker.com/r/ibmcom/etcd-s390x/tags
Figure 8-20 shows part of the content from this link:

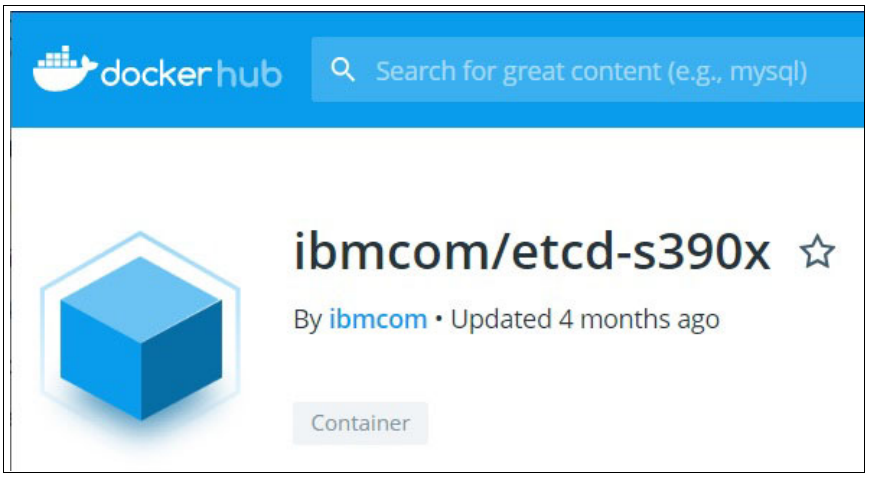


Figure 8-20 etcd Docker image for Linux on Z

On this web page, we saw multiple versions of the **etcd** image, each with its own Docker tags, including the latest image, which had the tag 3.2.24. See Figure 8-21,

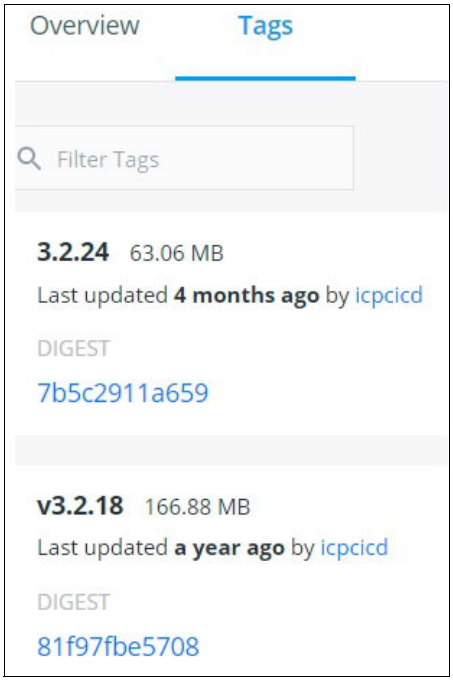


Figure 8-21 Tags for the **etcd** *images*

You can also use a command like the following one to list all the tags that are available:

```
curl -s -k https://registry.hub.docker.com/v1/repositories/ibmcom/etcd/tags | sed -e
's/[][]//g' -e 's/"//g' -e 's/ //g' | tr ',' "\n" |tr -d '}'| grep name | awk -F\: '{print
$2}' |head -n 15
```

The preceding command produced the output that is shown in Example 8-22.

Example 8-22 List of available tags for the **etcd** *image for Linux on Z*

```
3.2.24
3.2.24.2
v2.3.7
v3.1.5
v3.2.14
v3.2.18
```

Create Dockerfile to build the etcd image

In our zCX instance, we created a directory called **etcd** in the **/home/admin** path.

In that **etcd** directory, we then created a file called Dockerfile. In that Dockerfile, we inserted the content shown in Example 8-23:

Example 8-23 Dockerfile to build **etcd** *image*

```
ARG BASE_CONTAINER=ibmcom/etcd-s390x:3.2.24
FROM $BASE_CONTAINER

LABEL maintainer="IBM Redbooks etcd for zCX"

ENV ETCD_LISTEN_CLIENT_URLS=http://0.0.0.0:2379 \
    ETCD_ADVERTISE_CLIENT_URLS=http://0.0.0.0:2379

CMD /usr/local/bin/etcd
```

The ARG command lets us pass in a different tag name of the **etcd** image for Linux on Z as required.

We needed to set two **etcd**-related environment variables through the ENV command in the Dockerfile. These environment variables are explained here: https://etcd.io/docs/v3.4.0/op-guide/configuration/

By using the TCPIP value of 0.0.0.0 in these two environment variables, we tell the **etcd** process to listen on all interfaces.

The CMD command in Example 8-23 starts the **etcd** process when the image is run.

8.3.3 Build the etcd Docker image

We then issued this command to build the Docker image:

```
docker build -t zcxetcd .
```

The preceding command builds the image, gives it a name of **zcxetcd**, and reads the Dockerfile from the current directory we were in, which was **/home/admin/etcd**. This command produces the output shown in Example 8-24.

Example 8-24 Building the etcd Docker image

```
Step 1/5 : ARG BASE_CONTAINER=ibmcom/etcd-s390x:3.2.24
Step 2/5 : FROM $BASE_CONTAINER
 ---> 9f992b38e597
Step 3/5 : LABEL maintainer="IBM Redbooks etcd for zCX"
 ---> Running in 6081a9668b03
Removing intermediate container 6081a9668b03
```

```
 ---> 2d4d30a50bd1
Step 4/5 : ENV ETCD_LISTEN_CLIENT_URLS=http://0.0.0.0:2379
ETCD_ADVERTISE_CLIENT_URLS=http://0.0.0.0:2379
 ---> Running in 68f92e8e4285
Removing intermediate container 68f92e8e4285
 ---> 2bc95e032a09
Step 5/5 : CMD /usr/local/bin/etcd
 ---> Running in 57c249cb43e7
Removing intermediate container 57c249cb43e7
 ---> 357074519f9b
Successfully built 357074519f9b
Successfully tagged zcxetcd:latest
```

8.3.4 Run the etcd Docker image

After we built our **etcd** image, we ran it using the following command:

```
docker run -d --name myetcd -p 2379:2379 zcxetcd
```

- The **-p** parameters on the preceding command expose ports within the container to the outside world.
- The **-name** parameter names the running container as **myetcd**.

This command returned the following container ID for the created container:

```
0b8710d5f16325ae2a48453e8a66ffaea1a87f2be8aaeb7993c051bd08c93cd0
```

Example 8-25 shows the output of the **`docker ps`** command, which confirms that the **etcd** container is up and running:

Example 8-25 Confirming that etcd container is running

```
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS
NAMES
c5d0b1684479   zcxetcd   "/bin/sh -c /usr/loc…"   21 seconds ago   Up 15 seconds   2380/tcp, 4001/tcp,
0.0.0.0:2379->2379/tcp, 7001/tcp   myetcd
```

8.3.5 The z/OS application to call etcd

Because interaction with the **etcd** application takes place through standard REST API calls, we decided to use the z/OS Web Enablement toolkit and Rexx as an example of an application on z/OS doing REST calls into an application that runs inside a Docker container in our zCX instance.

The z/OS Web Enablement toolkit provides native APIs that traditional applications on z/OS can use to do REST API calls. Here is link to the z/OS Knowledge center that is a starting point for information about the z/OS Web Enablement toolkit:
https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.ieac100/ieac1-client-web-enablement.htm

For further information about the z/OS Web Enablement toolkit, see this presentation:
http://conferences.gse.org.uk/attachments/presentations/wAmX3q_1541239877.pdf

For more examples of using Rexx to invoke the z/OS Web Enablement toolkit APIs, see:
https://github.com/IBM/zOS-Client-Web-Enablement-Toolkit

For the source of the REXX code that was used for one of these examples, see the Example-GeoServices section at this link:

https://github.com/IBM/zOS-Client-Web-Enablement-Toolkit/blob/master/Example-GeoServices/RXEXEC1

A similar sample also is supplied with z/OS in this member: **SYS1.SAMPLIB(HWTHXRX1)**

We then modified the REXX code to be able to do the following two actions: 1) Insert a new key/value pair into the **etcd** container and 2) query the value of a key from the **etcd** container.

Appendix A, “Obtaining the additional material” on page 251 describes how to download the full REXX code.

Next, we set up the JCL shown in Example 8-26.

Example 8-26 JCL to run Rexx to interact with **etcd**

```
//ZCXETCD  EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//SYSTSIN DD *
   EXEC 'ZCX.REDB.EXEC(ZCXETCD)' +
 'PUT 129.40.23.68 2379/v2/keys/zcxkey2'
   EXEC 'ZCX.REDB.EXEC(ZCXETCD)' +
 'GET 129.40.23.68 2379/v2/keys/zcxkey2'
```

The first invocation of the Rexx program uses a PUT REST API call to insert a new key called **zcxkey2** into the **etcd** container. The Rexx program includes the following line to specify the value to set the key to: **queryParms = '?value=helloFromzOSRexx'**

This first call produced the output shown in Example 8-27:

Example 8-27 Output from doing PUT REST API call

```
HTTP Web Enablement Toolkit Sample (Begin)

Key in etcd set to value:  helloFromzOSRexx2

HTTP Web Enablement Toolkit Sample (End)
```

The second call uses a GET REST API call to retrieve the value of the key from the **etcd** container and produces the output shown in Example 8-28.

Example 8-28 Output from doing GET REST API call

```
HTTP Web Enablement Toolkit Sample (Begin)

Value of the key from etcd:  helloFromzOSRexx2

HTTP Web Enablement Toolkit Sample (End)
```

We further verified that the key was set, by using the URL to send a GET request:

`http://129.40.23.68:2379/v2/keys/zcxkey2`

Figure 8-22 shows the result.

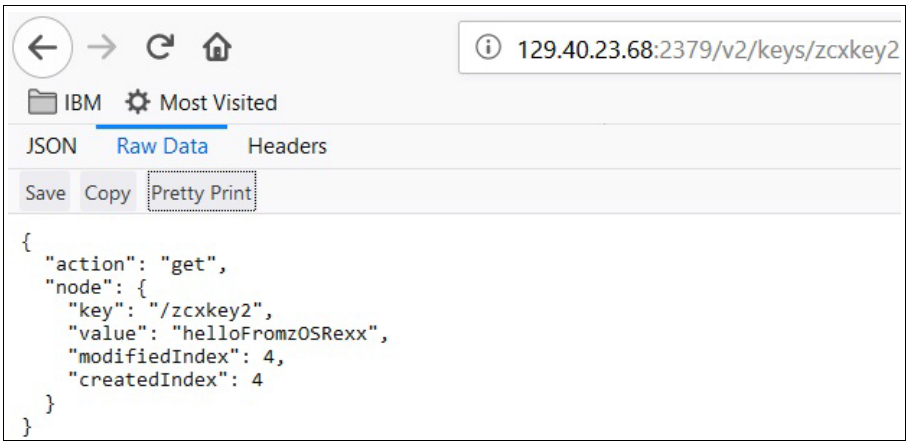


Figure 8-22 Output of GET request to etcd

8.3.6 Detail on setting up an etcd container

Getting a Docker image to run in a container is not always straightforward. In this section, we describe the steps that we went through to determine how to get the Linux on System Z **etcd** image working. That way, you have an example of how you can approach this task.

When we initially tried to set up the Linux on System Z version of the **etcd** image, there was no information about how to run this image.

From researching **etcd**, we learned that the image would have some number of TCPIP ports. These ports would need to be exposed so that REST APIs' requests can be sent to **etcd** in the container.

We clicked on the **digest** link that you see in Figure 8-21 on page 210. Figure 8-23 shows the resulting output.

Figure 8-23 Locating ports **etcd** *image exposes*

This output implied that we needed to expose the three ports shown.

We consulted the following web page to find out what **etcd** uses these ports for:

https://github.com/etcd-io/etcd/blob/master/Documentation/v2/configuration.md

This page includes the following explanation:

"The official **etcd** ports are 2379 for client requests, and 2380 for peer communication. Some legacy code and documentation still references ports 4001 and 7001, but all new **etcd** use and discussion should adopt the assigned ports."

From the preceding statement, it seemed that we needed to expose only the 2379 port, because we would not be setting up an **etcd** cluster.

We then issued this command to run a container for the **etcd** image:

```
docker run -d --name etcdzcx -p 2379:2379 -p 2380:2380 -p 4001:4001
ibmcom/etcd-s390x:3.2.24
```

We then issued the following command to check on the status of the container:

```
docker container ps -all
```

Example 8-29 shows the resulting output.

Example 8-29 Output showing **etcd** *container status*

```
CONTAINER ID        IMAGE                       COMMAND
CREATED             STATUS                         PORTS
NAMES
ba16f412028b        ibmcom/etcd-s390x:3.2.24   "sh"
About a minute ago   Exited (0) About a minute ago
etcdzcx
```

This output shows that our container was not running. We pursued the following theories:

- the container had tried to run **etcd** when it started but that had failed OR
- the container never tried to start **etcd**

To test whether we could run **etcd** in the container we run the following command to start the container. Notice that we passed an **sh** command at the end to provide a shell environment in the container:

```
docker rm etcdzcx
docker run -it --name etcdzcx ibmcom/etcd-s390x:3.2.24 sh
```

After we run the preceding commands, we got a shell prompt inside the container. There, we ran the following command to start **etcd**: `etcd`

Example 8-30 shows the resulting output.

Example 8-30 Output from starting **etcd** *inside the container*

```
2019-09-17 13:47:43.390482 W | etcdmain: running etcd on unsupported architecture "s390x" since ETCD_UNSUPPORTED_ARCH is
set
2019-09-17 13:47:43.399691 W | pkg/flags: unrecognized environment variable ETCD_UNSUPPORTED_ARCH=s390x
2019-09-17 13:47:43.399929 I | etcdmain: etcd Version: 3.2.24
2019-09-17 13:47:43.399940 I | etcdmain: Git SHA: 420a452
2019-09-17 13:47:43.399943 I | etcdmain: Go Version: go1.8.7
2019-09-17 13:47:43.399945 I | etcdmain: Go OS/Arch: linux/s390x
2019-09-17 13:47:43.399948 I | etcdmain: setting maximum number of CPUs to 1, total number of available CPUs is 1
...
2019-09-17 13:49:02.572230 I | etcdserver: published {Name:default ClientURLs:[http://localhost:2379]} to cluster
cdf818194e3a8c32
2019-09-17 13:49:02.572646 I | embed: ready to serve client requests
2019-09-17 13:49:02.575045 N | embed: serving insecure client requests on 127.0.0.1:2379, this is strongly discouraged!
```

The preceding output confirms that **etcd** could start successfully in the container.

We then started another session to the zCX instance and accessed the container by using this command:
docker exec -it etcdzcx sh

We then ran the following command to display what TCPIP ports the **etcd** container was listening on:
netstat -an
This command produced the output that is shown in Example 8-31:

Example 8-31 Output from netstat command

```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:2379          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:2380          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:44936         127.0.0.1:2379
ESTABLISHED
tcp        0      0 127.0.0.1:2379          127.0.0.1:44936
ESTABLISHED
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node Path
```

The preceding output confirmed that **etcd** was running but that the TCPIP ports that it was listening on were bound ***to the 127.0.0.1 localhost address***. That is why the ports could not be accessed from outside the container. Our laborious research revealed the following points:

- It is possible to make **etcd** listen on a different TCPIP address so that it can access ports from outside the container.
- The following URL documents the **etcd** environment variables that can be set: https://etcd.io/docs/v3.4.0/op-guide/configuration/
- We needed to set **etcd** environment variables as follows:
  export ETCD_ADVERTISE_CLIENT_URLS=http://0.0.0.0:2379
  export ETCD_LISTEN_CLIENT_URLS=http://0.0.0.0:2379

We then stopped the running **etcd** in the container by doing a CONTROL C, exited the container, and used Docker commands to kill and remove the image.

We then started the image again, this time exposing the 2379 port by using this command:

```
docker run -it --name etcdzcx -p 2379:2379 ibmcom/etcd-s390x:3.2.24 sh
```

Then, inside the container we ran the export commands that we found through our research. Example 32 shows the output that we saw after we started **etcd** again.

Example 32 Output showing how etcd now listening on the 2379 port

```
2019-09-17 14:14:15.308231 I | etcdserver: published {Name:default ClientURLs:[http://0.0.0.0:2379]} to
cluster cdf818194e3a8c32
2019-09-17 14:14:15.313888 I | embed: ready to serve client requests
```

From a browser, we then issued the following URL to access **etcd**:
http://129.40.23.68:2379/v2/keys
This URL produced the following JSON code: `{"action":"get","node":{"dir":true}}`

This result meant that we now had **etcd** up and running in a way that it can be accessed from outside the container.

Now, we were able to create the Dockerfile shown in Example 8-23 on page 211 and proceed as described there to work with our Rexx program.

8.3.7 Summary

In this example, we achieved two goals:

- Set up of a Docker container in a zCX instance.
- Successfully access the container from an application on z/OS.

9

zCX user administration

This chapter describes how to configure a zCX instance to use an existing LDAP server for authentication. LDAP authentication can be preferable to local user authentication because it provides an easy way to create identical user and group configurations across multiple instances. Modifications to users and groups are also much simpler when you use LDAP because changes to the LDAP architecture are immediately propagated to the authentication scheme of the instances.

When LDAP authentication is enabled for an instance, local user management and public key authentication are disabled.

All servers that abide by the RFC 4511 protocol (https://tools.ietf.org/html/rfc4511) are candidates for authentication of zCX. Setting up an LDAP server from scratch is beyond the scope of this chapter. For information about setting up a server see: https://www-01.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosv2r4sc236788/$file/glpa200_v2r4.pdf

The following user administration details are covered in this chapter:

- 9.1, “Local user management” on page 220
- 9.2, “Configuring zCX to use an LDAP server” on page 222
- 9.3, “Resources on the provisioning server and verifying that LDAP is enabled” on page 228

© Copyright IBM Corp. 2019. All rights reserved.

9.1 Local user management

After provisioning, the zCX instance Docker administrator is responsible for managing the Docker daemon and Docker users within the zCX appliance instance. The user information for the Docker administrator must be given during provisioning so that the user ID can be configured to manage the Docker daemon in the zCX instance. The Docker administrator does not necessarily need to be a z/OS user or zCX appliance administrator. How to configure and set up the Docker admin user is described in section 4.2.6, “Create SSH Keys” on page 51 .

9.1.1 Adding Docker users

Docker users have the ability to run Docker commands only. This condition exists because they are added to the Docker user group. Therefore, they cannot create or modify other Docker users.

Adding Docker users

To create a new Docker user with user name *username* and add the new user to the existing group Docker, the administrator runs the command in Example 9-1.

Example 9-1 Add a new docker user

```
admin@1edc6a418cd8:~$ sudo adduser --ingroup docker local-user
Adding user `local-user' ...
Adding new user `local-user' (1002) with group `docker' ...
Creating home directory `/home/local-user' ...
Copying files from `/etc/skel' ...
Enter new UNIX password: <password>
Retype new UNIX password: <password>
passwd: password updated successfully
Changing the user information for local-user
Enter the new value, or press ENTER for the default
        Full Name []: <details>
        Room Number []: <details>
        Work Phone []: <details>
        Home Phone []: <details>
        Other []: <details>
Is the information correct? [Y/n] Y
```

While creating the users, you must define the password for the user. After the password is set, further details can be given, or the default values can be used. Confirmation at the end is required with a yes (**Y**) entry.

To confirm that the user has been created and the users home directory is created, issue the **ls -l /home** command as shown in Example 9-2.

Example 9-2 Display the home directories

```
admin@1edc6a418cd8:~$ ls -l /home
total 0
drwxr-xr-x 1 admin      admin  186 Sep 17 20:27 admin
drwxr-xr-x 1 local-user docker  54 Sep 17 20:27 local-user
```

The administrator can force a new Docker user to change the user password at login by running the following command: **sudo passwd -e local-user**

Example 9-3 Force a password change during first logon

```
admin@1edc6a418cd8:~$ sudo passwd -e local-user
passwd: password expiry information changed.
```

After it is created, the new Docker user can SSH into the container and use Docker with the following command:

```
ssh local-user@129.40.23.79 -p 8022
```

This command results in a logon on the local zCX instance.

Example 9-4 SSH Logon with the newly created user

```
admin@1edc6a418cd8:~$ ssh local-user@129.40.23.79 -p 8022
local-user@129.40.23.79's password: <current_password>
You are required to change your password immediately (root enforced)

Welcome to the IBM z/OS Container Extensions (IBM zCX) shell that provides access to Docker
commands.
For more information on how to use this shell to execute Docker commands refer to "IBM z/OS
Container Extensions Guide".
WARNING: Your password has expired.
You must change your password now and login again!
Changing password for local-user.
(current) UNIX password: <current_password>
Enter new UNIX password: <new_password>
Retype new UNIX password: <new_password>
passwd: password updated successfully
```

If this is the first time that you are logging on to the machine, you must confirm the ssh key with **YES**. Afterward, you must log on with the password that was initially set as shown in Example 9-1 on page 220. At that point, you are forced to change it to a new password.

Adding Docker administrators

You create additional Docker administrators by adding Docker users to the Groups ‘Docker’ and giving them sudo access. The only abilities that the Docker Admin has beyond that of a standard Docker User is to add, modify, and delete other Users and Admins. An administrator can create additional Docker administrators by using the command shown in Example 9-5.

Example 9-5 Add a new docker administrator

```
admin@1edc6a418cd8:~$ sudo adduser local-admin
Adding user `local-admin' ...
Adding new group `local-admin' (1002) ...
Adding new user `local-admin' (1003) with group `local-admin' ...
Creating home directory `/home/local-admin' ...
Copying files from `/etc/skel' ...
Enter new UNIX password: <password>
Retype new UNIX password: <password>
passwd: password updated successfully
Changing the user information for local-admin
Enter the new value, or press ENTER for the default
        Full Name []: <details>
        Room Number []: <details>
        Work Phone []: <details>
        Home Phone []: <details>
        Other []: <details>
Is the information correct? [Y/n] y
```

While creating the users, you must define the password for the user. When the password is set, further details can be given or the default values can be used. Confirmation at the end is required with **Y**.

Then, add the user to the groups **docker** and **sudo** by using the commands shown in Example 9-6.

Example 9-6 Modify admin user to be part of docker and sudo group

```
admin@1edc6a418cd8:~$ sudo adduser local-admin docker
admin@1edc6a418cd8:~$ sudo adduser local-admin sudo
```

If needed, the administrator can then switch to the new user by using the command shown in Example 9-7 on page 222

Example 9-7 Switch the user

```
admin@1edc6a418cd8:~$ su local-admin
Password:
Sudo only permits specific User Management functions.  See additional
documentation for details.


local-admin@1edc6a418cd8:/home/admin$
```

To verify the groups that a user is connected to, you can just issue the **groups** command.

Example 9-8 Display the groups a user is connected to

```
local-admin@1edc6a418cd8:/home/admin$ groups
local-admin sudo docker
```

9.2 Configuring zCX to use an LDAP server

In this section, we discuss how to configure zCX to use an LDAP server.

9.2.1 LDAP server configuration

When LDAP authentication is enabled, the zCX instance queries the configured search base for all **posixAccount** and **posixGroup** entries that match any specified filters. Then, it allows those users with **posixAccount entries** to log in to the zCX instance by using those entries, **uid** and **userPassword**. Therefore, it is important to verify that all users who should be able to authenticate have an entry in the LDAP configuration that includes a **posixAccount objectClass** defined as follows:

```
objectClass: posixAccount
```

The zCX instance respects all required and optional parameters of the **posixAccount** object class. It is recommended that the optional **loginShell** attribute be specified (Bourne shell is default). For more information about required and optional fields see: https://ldapwiki.com/wiki/PosixAccount

Figure 9-1 illustrates an example LDAP directory tree that contains **posixAccount** and **posixGroup** entries. The remaining examples in this chapter are based on this directory tree.

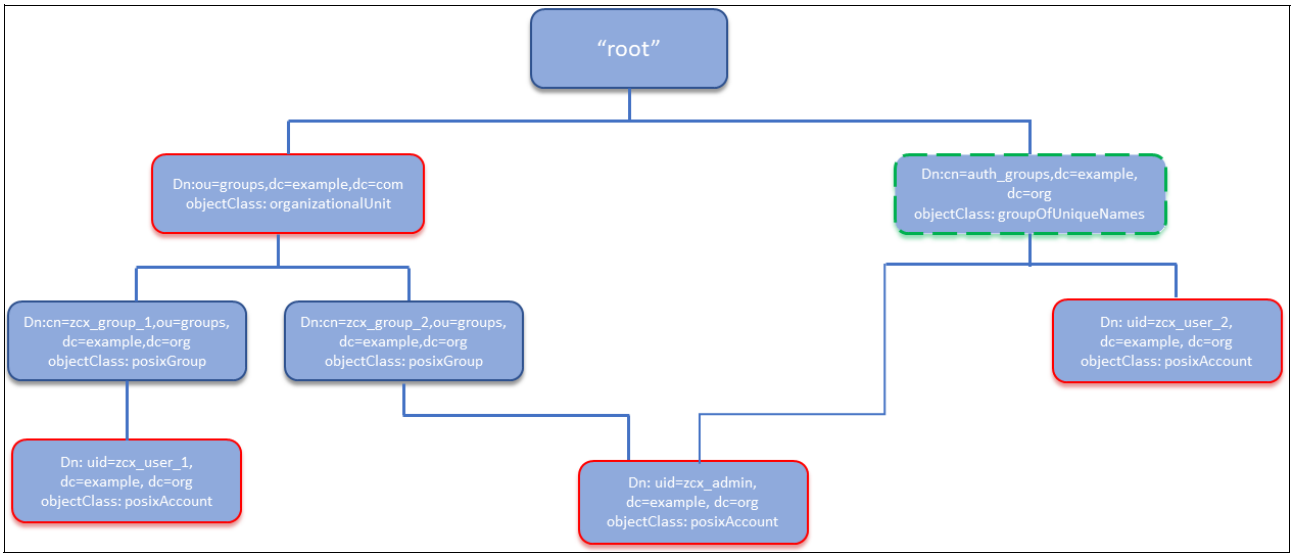


Figure 9-1 Example LDAP architecture

Notice the following aspects of Figure 9-1:

- There are posix accounts defined: **zcx_user_1**, **zcx_user_2**, **zcx_admin**.
- Two posix groups are defined: **zcx_group_1**, **zcx_group_2**
- User **zcx_admin** is a member of both groups, **zcx_user_1** is a member of **zcx_group_1** and **zcx_user_2** is a member of **zcx_group_2**
- A single **groupOfUniqueNames** is defined called **auth_group**, which contains **zcx_admin** and **zcx_user_2**. This is used in Example 9-9 on page 223 to enable group-based authentication. This approach is mandatory at the time of this writing, because **posixGroup** entries are not valid candidates for the "memberof" function for **pam** filters.

Each of the three users that are created with their own home directories and are started when the bash shell runs.

To create such an LDAP directory tree, **ldapadd** commands can be issued to insert the seven **ldif** files, which are shown in Example 9-9 through Example 9-15.

Example 9-9 Creation of "groups" organizational unit: groups.ldif

```
dn: ou=groups,dc=example,dc=org
objectclass: organizationalunit
ou: groups
```

Example 9-10 Creation of "auth_group" group of unique names : auth_group.ldif

```
dn: cn=auth_group,dc=example,dc=org
objectClass: top
objectClass: groupOfUniqueNames
cn: auth_group
uniqueMember: zcx_user_2,dc=example,dc=org
uniqueMember: zcx_admin,dc=example,dc=org
```

Example 9-11 Creation of "zcx_group_1" posixGroup: zcx_group_1.ldif

```
dn: cn=zcx_group_1,ou=groups,dc=example,dc=org
objectClass: top
```

```
objectClass: posixGroup
cn: zcx_group_1
gidNumber: 2001
memberUid: zcx_user_1
memberUid: zcx_admin
```

Example 9-12 Creation of “zcx_group_2” posixGroup: zcx_group_2.ldif

```
dn: cn=zcx_group_2,ou=groups,dc=example,dc=org
objectClass: top
objectClass: posixGroup
cn: zcx_group_2
gidNumber: 2002
memberUid: zcx_user_2
memberUid: zcx_admin
```

Example 9-13 Creation of “zcx_user_1” posixAccount: zcx_user_1.ldif

```
dn: uid=zcx_user_1,dc=example,dc=org
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
userPassword: pass
cn: zcx_user_1
sn: zcx_user_1
uid: zcx_user_1
uidNumber: 1001
gidNumber: 2001
homedirectory: /home/zcx_user_1
loginShell: /bin/bash
```

Example 9-14 Creation of “zcx_user_2.ldif” posixAccount: zcx_user_2.ldif

```
dn: uid=zcx_user_2,dc=example,dc=org
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: posixAccount
userPassword: pass
cn: zcx_user_2
sn: zcx_user_2
uid: zcx_user_2
uidNumber: 1002
gidNumber: 2002
homedirectory: /home/zcx_user_2
loginShell: /bin/bash
```

Example 9-15 Creation of “zcx_admin” posixAccount: zcx_admin.ldif

```
dn: uid=zcx_admin,dc=example,dc=org
objectClass: top
objectClass: person
objectClass: organizationalPerson
```

```
objectClass: inetOrgPerson
objectClass: posixAccount
userPassword: pass
cn: zcx_admin
sn: zcx_admin
uid: zcx_admin
uidNumber: 1003
gidNumber: 2001
homedirectory: /home/zcx_admin
loginShell: /bin/bash
```

9.2.2 Creating the LDAP configuration file

Creation of an LDAP configuration file is an integral part of enabling a connection to the LDAP server. The file describes where the server can be found. And the file defines the search domain from which user and group information should be retrieved. For a full list of options see the following web page:

http://manpages.ubuntu.com/manpages/trusty/man5/pam_ldap.5.html

This configuration is based entirely on how each LDAP server is configured, but some of the more common options are as follows:

- **URI -** The full URI of the LDAP server, starting with ldap:// for non-ssl enabled connections or ldaps:// enabled connections, followed by the hostname and an optional port (defaults to 389 for non-ssl and 636 for ssl)
- **BASE -** The distinguished name of the search base in which to look for **posixAccount** entries.
- **BINDDN -** The fully distinguished name of the bind user to use for authentication, this is required for LDAP servers that do not allow anonymous binds.
- **BINDPW -** The password of the bind user, which is required for LDAP servers that do not allow for anonymous binds.
- **TLS_CACERTFILE -** Location of the CA Certificate that signed the LDAP servers certificates. This location refers to a path on the zCX instance, not on the provisioning server. For instances with TLS enabled, this value is always */etc/ldap/ldap-ca.crt* regardless of the original name of the provided CA Certificate file.
- **SSL -** Indicates whether SSL should be enabled for the connection to the LDAP server. For instances that are configured to use TLS without SSL, set this value to **starttls**.
- **PAM _FILTER -** Allows for additional filtering of entries that appear within the search base. An example of a PAM_FILTER used to limit access to a specific group within the search base can be found in Example 9-16. For more information on creating PAM filters see:

  http://www.ldapexplorer.com/en/manual/109010000-ldap-filter-syntax.htm

Building on the structure that is defined in Figure 9-1 on page 223, assume that the LDAP server is running on IP 129.40.23.84, uses the default port of 389, and that SSL is not enabled. TLS authentication is enabled for this instance. The configuration shown in Example 9-16 on page 225 defines properties that connect to the example server and allow access to the **ZCX_USER_2** and **ZCX_ADMIN** user by defining a **PAM_FILTER** to allow access only to users in the **AUTH_GROUP**.

Example 9-16 Sample configuration for a PAM_FILTER

```
# IP and port of the LDAP server
```

```
URI ldap://129.40.23.84:389/
# Distinguished name of base search path
BASE dc=example,dc=org
# Distinguished name of bind user for authentication
binddn cn=admin,dc=example,dc=org
# Location of CA Certificate. This value is always
TLS_CACERTFILE /etc/ldap/ldap-ca.crt
# SSL Configuration, not using SSL in this scenario
SSL starttls
# Password of bind user
bindpw admin
# Restrict access to posixaccount entries in the search path that are members of
AUTH_GROUP. The group specified as the target of "memberof" cannot be a
posixGroup, hence the creation of specific groupOfUniqueNames entry that contains
the users that should be filtered.
pam_filter
&(objectclass=posixaccount)(memberof=cn=auth_group,ou=groups,dc=example,dc=org)
```

9.2.3 Enabling LDAP server authentication through z/OSMF Workflows

LDAP server authentication can be optionally enabled in either the *provision* z/OSMF workflow for new instances or in the *reconfigure* z/OSMF workflow for instances that have already been provisioned. For more information about using workflows for zCX configuration, see Chapter 4, "Provisioning and managing your first z/OS Container" on page 47. In either type of workflow, there are four options that must be considered and configured. Table 9-1 describes these fields.

Table 9-1 LDAP server parameters for z/OSMF workflow

Field Name	Description	Required
Enable LDAP Authentication	Whether LDAP authentication should be used for this instance instead of local user authentication. This value should be set to TRUE.	Yes
LDAP Client Configuration File Path	The absolute path to the LDAP client configuration file on the provisioning system, this file must be accessible to the provisioning user. This file describes: ► How the LDAP server can be reached. ► The search base that users should be queried from. ► Security for connections that are made to the server. An example of a simple configuration is provided in Appendix 9.2.1, "LDAP server configuration" on page 222. For a full list of options see: https://manpages.ubuntu.com/manpages/trusty/man5/pam_ldap.5.html	Yes
Enable LDAP Client TLS Authentication	Whether TLS authentication should be enabled for the LDAP client connection: ► For LDAP servers with TLS enabled this value should be set to TRUE. ► Otherwise, this should be set to FALSE.	Yes
LDAP Client TLS CA Certificate	The absolute path on the provisioning system to the TLS CA Certificate that signed the LDAP server certificate. This file must be accessible to the provisioning user and encoded in EBCDIC. This file should be provided by your LDAP server administrator.	This value is only required when Client TLS Authentication is enabled

Each of these four parameters is located in the”Gather IBM zCX appliance instance properties” step as shown in Figure 9-2 on page 227 for the provision workflow and in the same step as shown in Figure 9-3 on page 227 for the reconfigure workflow.

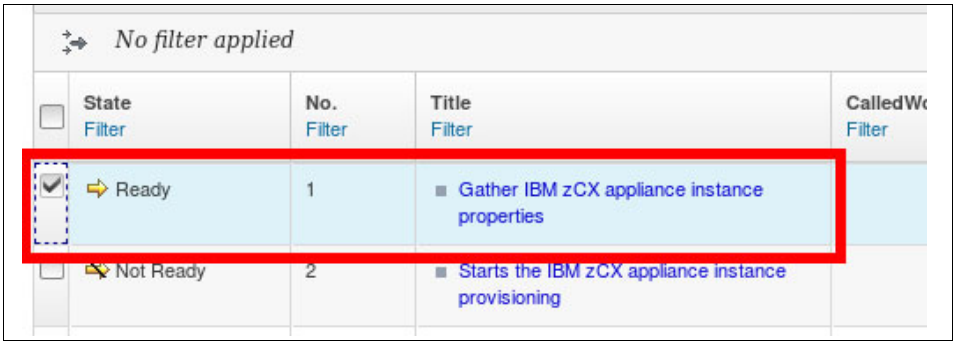


Figure 9-2 Gather IBM zCX appliance instance properties step in provision workflow

Figure 9-3 Gather IBM zCX appliance instance properties step in reconfigure workflow

After you reach the “Gather IBM zCX appliance instance properties” step of the provision workflow, navigate to the “zCX Docker User Management Configuration” substep as shown in Figure 9-4 on page 228. Dummy values have been filled in for the Docker Admin User ID and Docker Admin SSH Key fields. These dummy values serve to remind you that the Docker admin user will be disabled when LDAP Authentication is enabled.

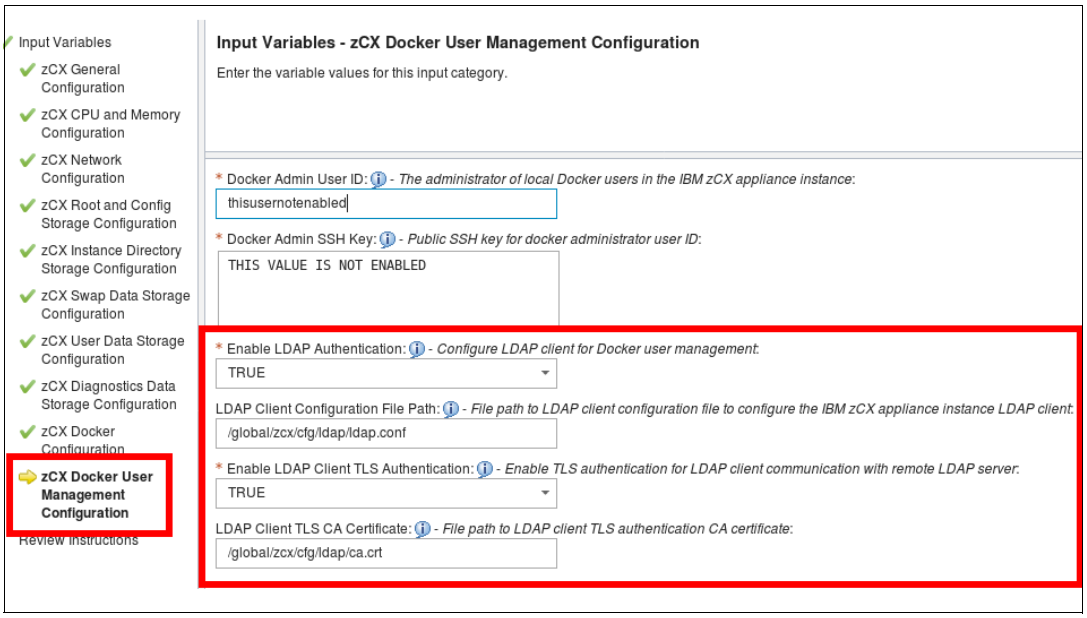


Figure 9-4 zCX Docker User Management Configuration for provision workflow

For the reconfigure workflow, navigate to the “zCX Docker User Management Configuration” substep as shown in Figure 9-5.

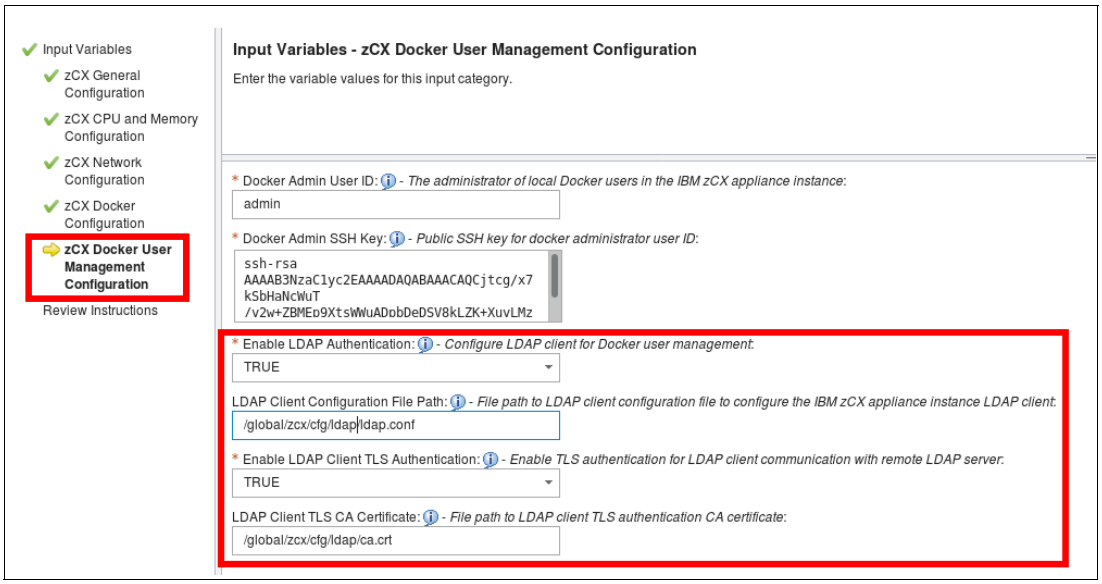


Figure 9-5 zCX Docker User Management Configuration for configure workflow

9.3 Resources on the provisioning server and verifying that LDAP is enabled

This section continues with the example in Example 9-2 on page 227. The zCX instance is configured to use the previously defined **ldap.conf** and to enable TLS authentication. The zCX instance runs with IP 129.40.23.85.

The **ldap.conf** and **ca.crt** files must be placed on the provisioning system and must be accessible to the provisioning user, which is ZCXPRV9 as seen in Figure 9-6. The **ca.crt** file must be encoded in EBCDIC.

```
ZACHB:/global/zcx/cfg/ldap: >ls -l
total 32
-rw-r--r--   1 ZCXPRV9  ZCXPRVG      949 Sep 12 10:43 ca.crt
-rw-r--r--   1 ZCXPRV9  ZCXPRVG      392 Sep 12 10:44 ldap.conf
```

Figure 9-6 LDAP configuration directory on provisioning system

After completing the provisioning or reconfigure workflow, you can verify that LDAP server authentication is enabled by attempting to log in with the **zcx_admin** user as shown in Figure 9-7.

```
[zacburns@oc3386673386 ~]$ ssh zcx_admin@129.40.23.85 -p 8022
zcx_admin@129.40.23.85's password:

Welcome to the IBM z/OS Container Extensions (IBM zCX) shell that provides access to Docker commands.
For more information on how to use this shell to execute Docker commands refer to "IBM z/OS Container
Last login: Fri Sep 13 21:22:53 2019 from 9.160.6.244
$ id
uid=1003(zcx_admin) gid=2001(zcx_group_1) groups=2001(zcx_group_1),109(docker),2002(zcx_group_2)
$ ls -la
total 12
drwxr-xr-x 1 zcx_admin zcx_group_1   66 Sep 13 14:20 .
drwxr-xr-x 1 root      root          58 Sep 13 14:20 ..
-rw-r--r-- 1 zcx_admin zcx_group_1  220 Sep 13 14:20 .bash_logout
-rw-r--r-- 1 zcx_admin zcx_group_1 3771 Sep 13 14:20 .bashrc
drwx------ 1 zcx_admin zcx_group_1   40 Sep 13 14:20 .cache
-rw-r--r-- 1 zcx_admin zcx_group_1  807 Sep 13 14:20 .profile
$ pwd
```

Figure 9-7 Verifying zcx_admin user access

At this point, the admin account that was configured during provisioning no longer exists. You can confirm this fact by attempting to ssh in to the admin account. A password prompt like the one in Figure 9-8 is displayed.

```
File   Edit   View   Search   Terminal   Help
[zacburns@oc3386673386 ~]$ ssh admin@129.40.23.85 -p 8022
admin@129.40.23.85's password:
```

Figure 9-8 Instance admin account disabled

10

Persistent data

This chapter provides information about persistent data and how it is important in the container world.

This chapter presents the following topics:

- 10.1, “Overview” on page 232
- 10.2, “Using Docker volumes for persistent data” on page 235

© Copyright IBM Corp. 2019. All rights reserved.

10.1 Overview

Persistent data is data that you want to be available, even if the container is removed. Application developers and Docker administrators naturally want to know how to preserve data for applications and for users. They must also understand the requirements of their applications, the types of data that are involved, and how the data is accessed.

Containers are inherently ephemeral. They are routinely destroyed and rebuilt from a previously pushed application image. Keep in mind that after a container is removed, all container data is gone. With containers, it is necessary that you take specific actions to deal with the ephemeral behavior. Docker offers efficient ways to manage data of an application while respecting this ephemeral nature of containers.

Docker volumes are a key data management tool. You use Docker volumes to:

- Persist data
- Share data volumes between containers

By default, data that is generated inside the container is only available from within the container and only for the lifetime of the container instance.

There are two principal options for storing persistent data in containers. This section covers the management of persistent data through the use of bind mounts and Docker volumes.

Table 10-1 Bind mounts and Docker volumes

Option for persistent data	Description
Bind Mounts	Permits the mapping of a directory on the Docker host to a directory in container.
Docker Volumes	Directories and files that are located outside of the Docker containers to save and share data between Docker containers.

The Docker volumes ensure that runtime configuration changes are persisted on the zCX and are not lost when you stop the Docker containers.

zCX provides limited access to bind the mounts to the following root directories of the underlying Linux host:

- Access to the base Linux directories is read-only.
- Access to the **/var/run/docker.sock** UNIX socket file will be allowed.

If you try to bind a non-authorized directory, you see the error that is listed in Example 10-1.

Example 10-1 Trying to access a non-authorized directory

```
admin@570e9473805e:~$ docker run -dit --name my-apache-app -p 8080:80 -v
"$PWD":/usr/local/apache2/htdocs/ httpd:2.4
docker: Error response from daemon: authorization denied by plugin zcxauthplugin:
Request to bind mount the path "/home/admin" with rw mode is disabled for Docker
running on IBM zCX appliance instance.
See 'docker run --help'.
```

Docker volumes are currently the preferred mechanism for persisting data within zCX application containers. You can create new or use existing volume storage to save data in your container environment.

The Figure 10-1 details how containers use Docker volumes.

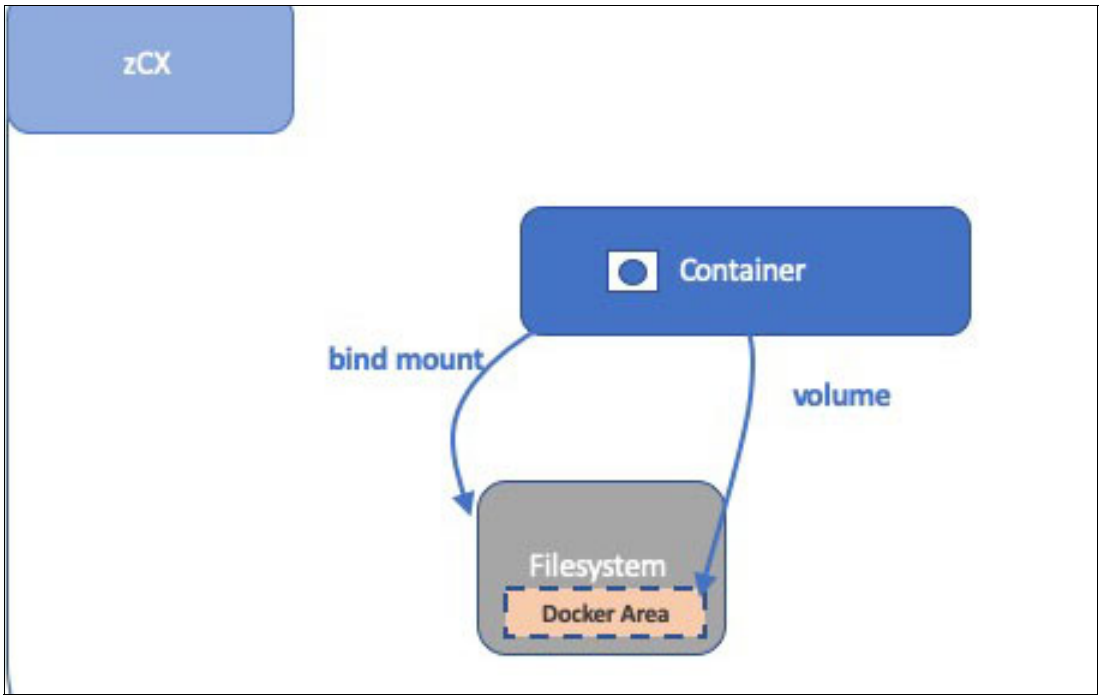


Figure 10-1 Docker Volumes

Persistent data is an important topic if you are going to deploy containerized applications. In the next sections, we demonstrate how to manage Docker volumes.

Shared Docker volumes

Docker volumes can also be used to share data between containers. You can use shared Docker volumes to cover a business requirement where multiple Docker containers must access the same data or share required information between them.

Although it's considered a best practice to have consistent user and group IDs (UIDs and GIDs) among the containers, consistency is not practical when you have multiple containers that access a shared Docker volume based on different images. In Example 10-2 on page 234, there is one Docker volume that is shared between different containers based on two images.

Unless you control both Dockerfiles when the images are created, you cannot guarantee that both users and groups in the two images have the same UID and GID. Consider an example where you do not have this control. Assume that there are two directories that are named *img_1_data* and *img_2_data*. They are owned by two different users and groups. As a result, these users and groups cannot share between each other without further manipulation of the file system permissions.

However, if the user and groups are set up properly in those two images, data sharing between those instances is possible without any further actions.

Consideration needs to be given to avoid problems with file system access. If you do not have the same UID and GID, you might lose access from one of the containers. So, user management planning is a good practice before you deploy applications on a shared Docker volume.

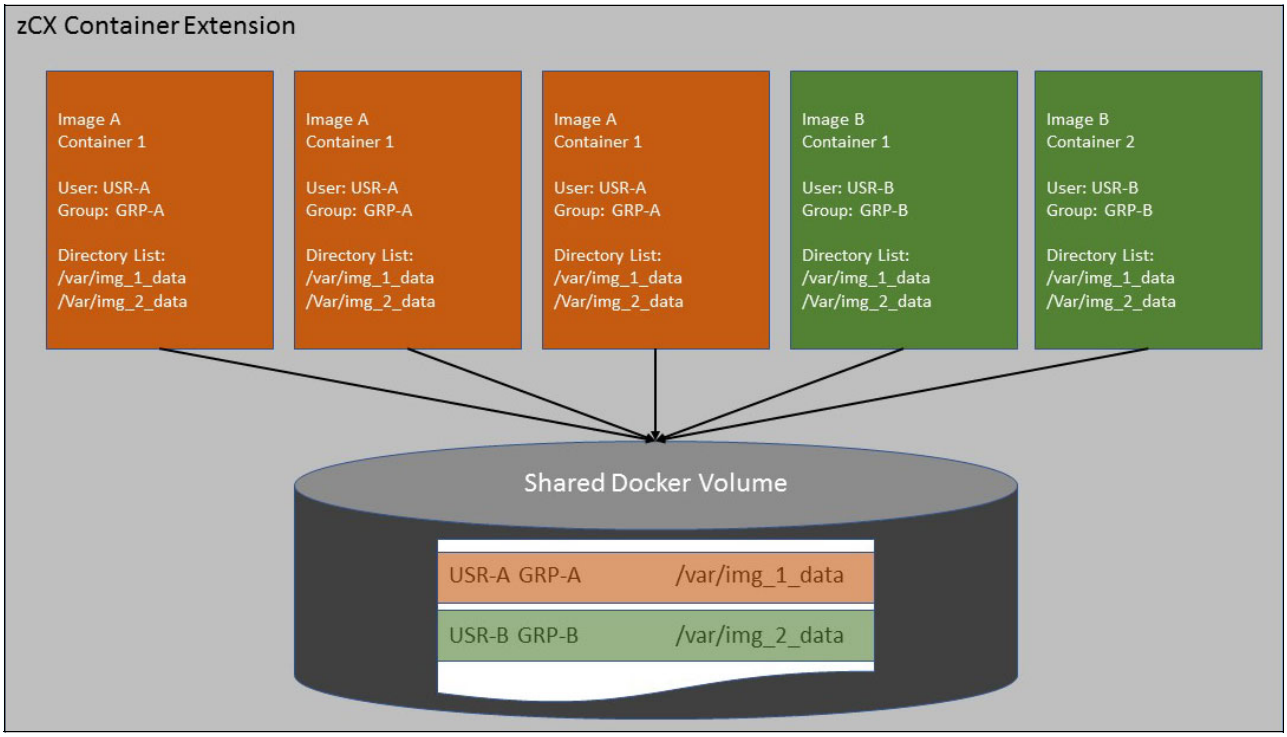


Figure 10-2 Shared Volume across containers based on different images

When consistent UIDs and GIDs cannot be guaranteed in two different images and data sharing between those images is not required, you can set up the Docker volumes as illustrated in Figure 10-3. A setup like this guarantees that another container cannot access data from another container's volume.

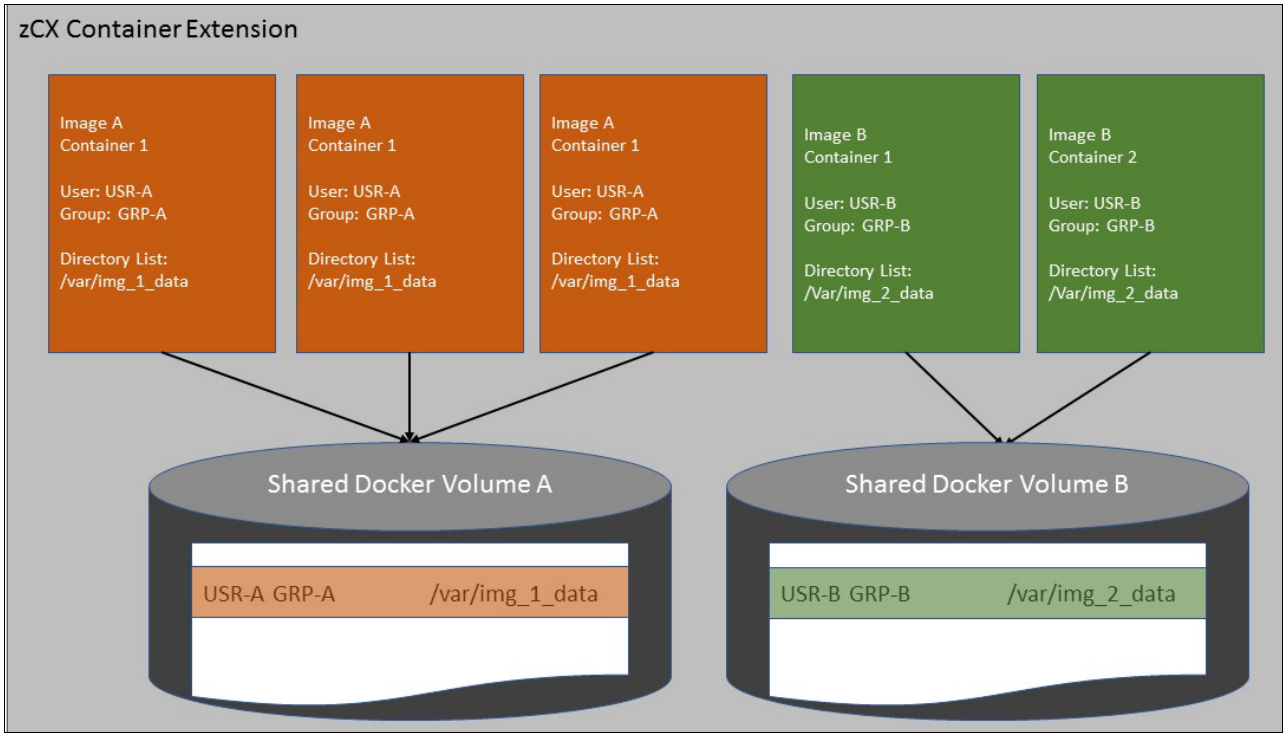


Figure 10-3 Shared Volume across containers based on the same image

10.2 Using Docker volumes for persistent data

Defining a Docker volume is not a complex task. But if you run multiple containers that require persistent data, apply a consistent naming scheme. Give new Docker volumes names that are based on the containers' names or the application's name. A naming scheme helps you to easily identify who might be using a Docker volume.

In our lab, we defined two persistent data volumes to use in our secure private registry.

- **certs_volume** = This volume contains the TLS certificates.
- **registry_volume** = This volume contains the Docker images layers.

The **`docker volume create`** command shown in Example 10-2 defines the two new volumes.

Example 10-2 Creaating two new volumes

```
admin@a29d70c73c55:~$ docker volume create certs_volume
certs_volume

admin@a29d70c73c55:~$ docker volume create registry_volume
registry_volume
```

To confirm that the volumes were successfully created, run **`docker volume ls`** command as shown in Example 10-3. A list of all available volumes is displayed, including the new volumes, which are bolded for emphasis.

Example 10-3 Listing available Docker images

```
admin@a29d70c73c55:~$ docker volume ls
DRIVER              VOLUME NAME
local               34b06c9956dba12b0313f9e71ebef6b02edf00fdbc55773ba6cab8662dbc4e69
local               AZD_HOME_VOLUME
local               AZD_SHARED_VOLUME
local               certs_volume
local               registry_volume
```

You can add volumes for persistent data when you start the container. You do this by adding **`-v`** to your **`docker run`** command, as in the following example:

```
docker run -d -v <volume_name>:/<destination_folder>/ <image_name>
```

Table 10-2 Parameters details

Parameter	Description
<volume_name>	The name of the volume to add
<destination_folder>	The destination folder where volume will be mounted
<image_name>	The image name to be used on the new container

The command listed in Example 10-4 shows the creation of a new container.

Example 10-4 Creating a new container

```
admin@a29d70c73c55:~$ docker run -d --restart=always --name registry -v
certs_volume:/certs -v registry_volume:/var/lib/registry -e
REGISTRY_HTTP_TLS_CERTIFICATE=/certs/docker_sc74cn03.pbm.ihost.com.crt -e
```

```
REGISTRY_HTTP_TLS_KEY=/certs/docker_sc74cn03.pbm.ihost.com.key -p 5000:5000
ibmcom/registry-s390x:2.6.2.3
7e39a4e4fb2c21b0e739a80ea20b826774e5cee5d45e96e21392efdb9f80e4d9
```

The following two volumes were mounted on the new container.

- **certs_volume** was mounted on **/certs**
- **registry_volume** was mounted on **/var/lib/registry**

With Docker volumes, your application data is maintained outside the Docker container. It brings benefits such as these: you keep the container size remains small, and you maintain control of your data.

Keeping the size of your container under control is a good practice. It offers the following advantages:

- Faster provision of new container instances
- Easy to manage, because application data is separate from the overall application code
- Easy to share small images

Removing Docker Containers and all associated data

Deletion of a previously created container can be achieved by the **docker rm** command. Don't forget to remove all associated data volumes, too. You can specify the **-v** option to remove all volumes that are associated with the container that you are deleting. In associated volumes, wasted disk and space become difficult to manage later.

To remove a container, use the following command.

```
docker rm -v <container_name>
```

Note: Before you remove a container, ensure that it is stopped by using the **docker stop** command.

In Example 10-5, we remove a registry container

Example 10-5 Removing a container

```
admin@a29d70c73c55:~$ docker rm -v registry
registry
```

Note: Sometimes the -v option does not delete the Docker volumes for the container that you are deleting. In such cases, you must use **docker volume ls** to list all available volumes and then issue **docker voume rm <volume_name>** command to delete them.

To remove a container and all associated data, follow these steps:

1. Use **docker inspect** command to review all volumes associated to the container.

   ```
   docker inspect registry
   ```

   The output of **docker inspect** is quite long. In Example 10-6, we show only the Mounts section to list all volumes that are associated for that container.

Example 10-6 Inspecting a docker container

```
"Mounts": [
            {
                "Type": "volume",
                "Name": "certs_volume",
```

```
                "Source": "/media/data/docker/24000.109/volumes/certs_volume/_data",
                "Destination": "/certs",
                "Driver": "local",
                "Mode": "z",
                "RW": true,
                "Propagation": ""
            },
            {
                "Type": "volume",
                "Name": "registry_volume",
                "Source": "/media/data/docker/24000.109/volumes/registry_volume/_data",
                "Destination": "/var/lib/registry",
                "Driver": "local",
                "Mode": "z",
                "RW": true,
                "Propagation": ""
            }
]
```

1. Stop the container by using the **docker stop registry** command.
2. Remove the container by using the **docker rm registry** command.
3. Remove all Docker volumes that are associated with the container by using the following commands.

   docker volume rm certs_volume
   docker volume rm registry_volume

4. Now, confirm that related volumes were deleted by using **docker volume ls** command as shown in Example 10-7.

Example 10-7 Listing available Docker volumes

```
admin@a29d70c73c55:~$ docker volume ls
DRIVER              VOLUME NAME
local               34b06c9956dba12b0313f9e71ebef6b02edf00fdbc55773ba6cab8662dbc4e69
local               AZD_HOME_VOLUME
local               AZD_SHARED_VOLUME
```

11

Swarm on zCX

This chapter describes the use of Docker swarm mode on zCX, including the necessary configuration steps for setting up a clustering container environment.

Upon completion of this chapter, you should be able to deploy a container by using a web application on IBM Z and use orchestration tools.

This chapter includes the following topics:

- 11.1, “Swarm introduction” on page 240
- 11.2, “zCX in swarm mode” on page 241

© Copyright IBM Corp. 2019. All rights reserved.

11.1 Swarm introduction

The Docker website describes a *swarm* as follows:

> 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 given 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 available to it, ports the service exposes 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.[1]

The Docker command-line interface (CLI) is used to create a swarm, deploy application services to a swarm, and manage swarm behavior. (A web interface is not used for 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 happens through 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 be run inside the container. Manager nodes assign tasks to worker nodes according to the number of replicas that are

[1] https://docs.docker.com/engine/swarm/key-concepts/#what-is-a-swarm

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.

Figure 11-1 shows the architecture of a swarm.

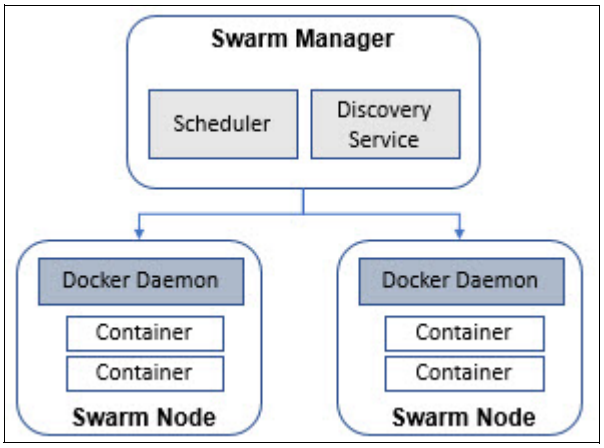


Figure 11-1 Swarm architecture diagram

- Load balancing

  The swarm manager uses ingress load balancing to expose 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 a DNS entry to each service in the swarm. The swarm manager uses internal load balancing to distribute requests among services within the cluster, which are based on the DNS name of the service.

11.2 zCX in swarm mode

In this section, we configure Docker Swarm mode on our zCX instances to demonstrate the application portability that can be achieved by zCX.

As discussed earlier in this chapter, Docker swarm is part of Docker Engine. Therefore, we do not have to install any more components to have functionality.

In this example, we complete the following steps:

1. Leave node from Swarm.
2. Start the swarm mode.
3. Add node manager to the cluster.
4. Add worker nodes to the Manager node.
5. Create a Docker service.
6. Scale up a container.

7. Change node availability to simulate a scheduled maintenance.
8. Promote a worker node.
9. Demote a manager node.
10. Scale down a container.

Our environment

The specific components of our swarm environment are listed in Table 11-1.

Table 11-1 Our zCX swarm environment

Server Name	Server IP	Swarm Node role	z/OS LPAR
sc74cn03.pbm.ihost.com	129.40.23.70	Manager	wtsc74.pbm.ihost.com
sc74cn04.pbm.ihost.com	129.40.23.71	Worker	wtsc75.pbm.ihost.com
sc74cn09.pbm.ihost.com	129.40.23.76	Manager	wtsc74.pbm.ihost.com
sc74cn10.pbm.ihost.com	129.40.23.77	Worker	wtsc75.pbm.ihost.com

The Example 11-2 depicts our zCX cluster environment.

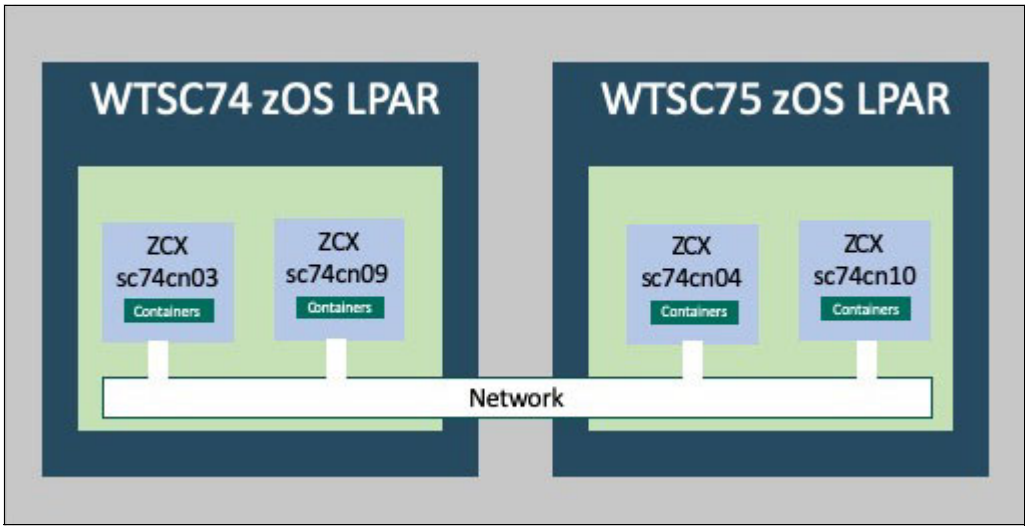


Figure 11-2 Our zCX environment

Note: Swarm uses server names and IP addresses for cluster configuration. Ensure that you correctly set up DNS with the name and IP addresses of the zCX instances.

11.2.1 Removing node from swarm

Use the **`docker swarm leave`** command if you need to remove a specific node from swarm cluster. If your zCX instance has never joined a swarm cluster, you can skip this step. Otherwise, issue the command that is shown in Example 11-1.

Example 11-1 Swiping swarm configuration

```
admin@570e9473805e:~$ docker swarm leave
Node left the swarm.
```

Note: For Manager Node, you must append **`--force`** to force the node to leave the cluster.

11.2.2 Initializing swarm mode

The first node in swarm is called Manager leader. This node is responsible for advertising its IP address. In our case, we use the **sc74cn03.pbm.ihost.com** (129.40.23.70) instance to act as leader. (See Example 11-2.)

Example 11-2 Initializing a swarm cluster

```
admin@570e9473805e:~$ docker swarm init --advertise-addr 129.40.23.70
Swarm initialized: current node (1xk53pdq2ujayiea2vbfu3bs1) is now a manager.

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

    docker swarm join --token
SWMTKN-1-17aalrjaxjmlx43ytth47xb23enxaek0ris89826ane1d6o99z-1zzz4qvmc8bxhb2le3gulg3nx
129.40.23.70:2377

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

Note: After issuing the preceding **`docker swarm init`** command, the master node is defined and displays the command that allows you to add new worker nodes to the cluster. Make a note of it because we use this command in the next steps.

Example 11-2 demonstrates a command that adds another master node into the cluster. We ran the command as instructed and Example 11-3 shows the output.

Example 11-3 Getting the command to join manager nodes into the cluster

```
admin@570e9473805e:~$ docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token
SWMTKN-1-17aalrjaxjmlx43ytth47xb23enxaek0ris89826ane1d6o99z-cafuexvwlvojqhuosk7lqe07q
129.40.23.70:2377
```

11.2.3 Adding node manager to the cluster

In this step, we add the second manager node to our swarm cluster.

To join **sc74cn09.pbm.ihost.com** as master node, follow these steps:

1. Connect to **sc74cn09.pbm.ihost.com** zCX instance by using ssh (in other words, **`ssh admin@sc74cn09.pbm.ihost.com`**).
2. Issue **`docker swarm join`**, including the token for the manager node that you got in Example 11-2. The following example shows the output:

Example 11-4 Joining a manager node

```
admin@01f543067055:~$ docker swarm join --token
SWMTKN-1-17aalrjaxjmlx43ytth47xb23enxaek0ris89826ane1d6o99z-cafuexvwlvojqhuosk7lqe
07q 129.40.23.70:2377
This node joined a swarm as a manager.
```

11.2.4 Adding worker nodes to the manager node

The worker nodes load the Docker workloads. Therefore, you can add worker nodes to process more workloads (see Example 11-5).

To join our first worker node (**sc74cn04.pbm.ihost.com**), complete the following steps:

1. Connect to **sc74cn04.pbm.ihost.com** zCX instance by using ssh (in other words, **ssh admin@sc74cn04.pbm.ihost.com**)
2. Issue **`docker swarm join`**, including the token for the worker node that you got in Example 11-2. The following example shows the output:

Example 11-5 Joining a worker node

```
admin@a29d70c73c55:~$ docker swarm join --token
SWMTKN-1-17aalrjaxjmlx43ytth47xb23enxaek0ris89826ane1d6o99z-1zzz4qvmc8bxhb2le3gulg
3nx 129.40.23.70:2377
This node joined a swarm as a worker.
```

Example 6 shows successful output for the command for the **sc74cn10.pbm.ihost.com** server.

Now, check the status of the cluster by running the command that is shown in Example 6 from one of the manager nodes.

Example 6 Checking status of our swarm cluster

```
admin@570e9473805e:~$ docker node ls
ID                            HOSTNAME                STATUS   AVAILABILITY  MANAGER STATUS    ENGINE VERSION
1xk53pdq2ujayiea2vbfu3bs1 *   sc74cn03.pbm.ihost.com  Ready    Active        Reachable         18.09.2
cidllaskmus095rrb221ii9b9     sc74cn04.pbm.ihost.com  Ready    Active                          18.09.2
4y9a3mej93xvq4denfphhz53u     sc74cn09.pbm.ihost.com  Ready    Active        Leader            18.09.2
j2b9596uktzhupuxv6td22mm1     sc74cn10.pbm.ihost.com  Ready    Active                          18.09.2
```

By default, all nodes in the cluster can receive Docker workloads. We are going to prevent the manager nodes from receiving application workloads. In our setup, only worker nodes will receive application workloads from Swarm. The following commands prevent the manager nodes from receiving workloads. Run these commands on any of the manager nodes. In our example, we are running it on the **sc74cn03.pbm.ihost.com** server and **sc74cn09.pbm.ihost.com** as shown in Example 11-7.

Example 11-7 Node update command

```
admin@570e9473805e:~$ docker node update --availability drain sc74cn03.pbm.ihost.com
sc74cn03.pbm.ihost.com

admin@570e9473805e:~$ docker node update --availability drain sc74cn09.pbm.ihost.com
sc74cn09.pbm.ihost.com

admin@570e9473805e:~$ docker node ls
ID                            HOSTNAME                STATUS  AVAILABILITY    MANAGER STATUS   ENGINE VERS
1xk53pdq2ujayiea2vbfu3bs1 *   sc74cn03.pbm.ihost.com  Ready   Drain           Reachable        18.09.2
cidllaskmus095rrb221ii9b9     sc74cn04.pbm.ihost.com  Ready   Active                           18.09.2
4y9a3mej93xvq4denfphhz53u     sc74cn09.pbm.ihost.com  Ready   Drain           Leader           18.09.2
j2b9596uktzhupuxv6td22mm1     sc74cn10.pbm.ihost.com  Ready   Active                           18.09.2
```

As you can see in Example 11-7, the manager nodes are now in Drain status. 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.

11.2.5 Creating a Docker service

In this step, we defined a cluster service called *web server*. This service provides the nginx application. Example 8 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.

These commands must run in any manager node that is part of the cluster. In our example, we ran it in the **sc74cn09.pbm.ihost.com** server.

Example 8 Creating a Docker service

```
admin@01f543067055:~$ docker service ls
ID              NAME           MODE           REPLICAS      IMAGE          PORTS

admin@01f543067055:~$ docker service create --name webserver -p 80:80 nginx:latest
pwu25hj3wpwwovo73d2is31gn
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service converged

admin@01f543067055:~$ docker service ls
ID              NAME          MODE           REPLICAS       IMAGE           PORTS
pwu25hj3wpww    webserver     replicated     1/1            nginx:latest    *:80->80/tcp
```

You can get more information about the web server service by running the command that is shown in Example 11-9 (the nginx container was started in **sc74cn04.pbm.ihost.com** server, which is the worker node. Remember that we removed **sc74cn03.pbm.ihost.com** and **sc74cn09.pbm.ihost.com** from receiving application workloads because they were drained in a previous step. By default, only one replica is started.

Example 11-9 Checking web server service

```
admin@01f543067055:~$ docker service ps webserver
ID             NAME           IMAGE          NODE                     DESIRED STATE   CURRENT STATE         ERROR   PORT
vkqgrr6aypi3   webserver.1    nginx:latest   sc74cn04.pbm.ihost.com   Running         Running 2 minutes ago
```

Now, we stopped the container on **sc74cn04.pbm.ihost.com** server to see the behavior when someone takes down a container that is managed by swarm (see Example 11-10).

Example 11-10 Stopping nginx container on itsosleb server

```
admin@a29d70c73c55:~$ docker ps
CONTAINER ID    IMAGE                   COMMAND                  CREATED          STATUS          PORTS             NAME
0f0fac618197    nginx:latest            "nginx -g 'daemon of…"   5 minutes ago    Up 5 minutes    80/tcp
webserver.1.vkqgrr6aypi3e8ue88wuqedxw
a29d70c73c55    ibm_zcx_zos_cli_image   "sudo /usr/sbin/sshd…"   2 hours ago      Up 2 hours      8022/tcp,
0.0.0.0:8022->22/tcp   ibm_zcx_zos_cli

admin@a29d70c73c55:~$ docker stop 0f0fac618197
0f0fac618197

admin@a29d70c73c55:~$ docker ps
CONTAINER ID    IMAGE                   COMMAND                  CREATED          STATUS                  PORTS     NAME
e5a065c3177b    nginx:latest            "nginx -g 'daemon of…"   6 seconds ago    Up Less than a second   80/tcp
webserver.1.2y909gfnju13ybtxwsknljuyr
a29d70c73c55    ibm_zcx_zos_cli_image   "sudo /usr/sbin/sshd…"   2 hours ago      Up 2 hours              8022/tcp,
0.0.0.0:8022->22/tcp   ibm_zcx_zos_cli
```

Swarm identified that container nginx was down on the **sc74cn04.pbm.ihost.com** worker node and automatically restarted a new container on the same node. Typically that restart happens on the other worker node. But the decison depends on the calculations of the swarm algorithm.

To confirm that the container was restarted, we ran the command listed in Example 11-11 on sc74cn03.pbm.ihost.com (master node) to show the status of the web server.

Example 11-11 Checking status of web server service after container stops

```
admin@570e9473805e:~$ docker service ps webserver
ID             NAME              IMAGE           NODE                      DESIRED STATE   CURRENT STATE         ERROR
PORTS
2y909gfnju13   webserver.1       nginx:latest    sc74cn04.pbm.ihost.com   Running         Running 5 minutes ago
vkqgrr6aypi3    \_ webserver.1   nginx:latest    sc74cn04.pbm.ihost.com   Shutdown        Complete 5 minutes ago
```

11.2.6 Scaling up a container

Scaling a container allows you to increase the number of containers. This activity is common when you need to increase capacity for your application and allow more requests to be processed by the cluster.

We scaled up the web server service from 1 to 4 instances, as shown in Example 11-12. We ran this command on sc74cn03.pbm.ihost.com (master node).

Example 11-12 Scaling up your web server service

```
admin@570e9473805e:~$ docker service scale webserver=4
webserver scaled to 4
overall progress: 4 out of 4 tasks
1/4: running   [==================================================>]
2/4: running   [==================================================>]
3/4: running   [==================================================>]
4/4: running   [==================================================>]
verify: Service converged

admin@570e9473805e:~$ docker service ps webserver
ID             NAME              IMAGE           NODE                      DESIRED STATE  CURRENT STATE      ERROR  PORT
2y909gfnju13   webserver.1       nginx:latest    sc74cn04.pbm.ihost.com   Running        Running 7 minutes ago
vkqgrr6aypi3    \_ webserver.1   nginx:latest    sc74cn04.pbm.ihost.com   Shutdown       Complete 7 minutes ago
omled936b5dt   webserver.2       nginx:latest    sc74cn10.pbm.ihost.com   Running        Running 16 seconds ago
4ztuxznjrc8t   webserver.3       nginx:latest    sc74cn10.pbm.ihost.com   Running        Running 16 seconds ago
hqb26nea8vmf   webserver.4       nginx:latest    sc74cn04.pbm.ihost.com   Running        Running 17 seconds ago

admin@570e9473805e:~$ docker service ls
ID             NAME        MODE          REPLICAS    IMAGE           PORTS
pwu25hj3wpww   webserver   replicated    4/4         nginx:latest    *:80->80/tcp
```

For the next example, we scaled the web server service from 4 to 30, as shown in Example 11-13.

Example 11-13 Scaling web server service from 4 to 30

```
admin@570e9473805e:~$ docker service scale webserver=30
webserver scaled to 30
overall progress: 30 out of 30 tasks
verify: Service converged

admin@570e9473805e:~$ docker service ls
ID             NAME        MODE         REPLICAS   IMAGE         PORTS
```

```
pwu25hj3wpww     webserver   replicated   30/30      nginx:latest  *:80->80/tcp
```

11.2.7 Changing node availability to simulate a scheduled maintenance

Setting node availability from active to drain is often used when you need to perform maintenance on 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 sc74cn04.pbm.ihost.com (worker node) server changed from active to drain (see Example 11-14).

Example 11-14 Putting server in DRAIN mode

```
admin@570e9473805e:~$ docker node update --availability drain sc74cn04.pbm.ihost.com
sc74cn04.pbm.ihost.com

admin@570e9473805e:~$ docker node ls
ID                            HOSTNAME                STATUS    AVAILABILITY  MANAGER STATUS  ENGINE VERSI
1xk53pdq2ujayiea2vbfu3bs1 *   sc74cn03.pbm.ihost.com  Ready     Drain         Reachable       18.09.2
cidllaskmus095rrb221ii9b9     sc74cn04.pbm.ihost.com  Ready     Drain                         18.09.2
4y9a3mej93xvq4denfphhz53u     sc74cn09.pbm.ihost.com  Ready     Drain         Leader          18.09.2
j2b9596uktzhupuxv6td22mm1     sc74cn10.pbm.ihost.com  Ready     Active                        18.09.2

admin@570e9473805e:~$ docker service ls
ID             NAME         MODE         REPLICAS    IMAGE         PORTS
pwu25hj3wpww   webserver    replicated   30/30       nginx:latest  *:80->80/tcp
```

The output that is shown in Example 11-15 shows that all containers for the web server service were stopped.

Example 11-15 Show containers that are available

```
admin@a29d70c73c55:~$ docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS
PORTS                           NAMES
a29d70c73c55        ibm_zcx_zos_cli_image   "sudo /usr/sbin/sshd…"   2 hours ago         Up 2 hours
8022/tcp, 0.0.0.0:8022->22/tcp   ibm_zcx_zos_cli
```

To return the **sc74cn04.pbm.ihost.com** (worker node) server back to active, run the command that is shown in Example 11-16. We also are scaling the web server service to 105 to see instances that are running on **sc74cn04.pbm.ihost.com** server. This is because swarm does not automatically rebalance the cluster.

Example 11-16 Returning the server back to accept workloads

```
# Making sc74cn04.pbm.ihost.com server available to the cluster
admin@570e9473805e:~$ docker node update --availability active sc74cn04.pbm.ihost.com
sc74cn04.pbm.ihost.com

# Scaling cluster from 30 to 105
admin@570e9473805e:~$ docker service scale webserver=105
webserver scaled to 105
overall progress: 105 out of 105 tasks
verify: Service converged
admin@570e9473805e:~$

#Connected on sc74cn04.pbm.ihost.com worker node to confirm containers running after scale
the cluster. 53 instances of nginx running on sc74cn04.
```

```
admin@a29d70c73c55:~$ docker ps |grep nginx |wc -l
53
```

11.2.8 Promoting a worker node

You might promote a worker node to be a manager node in scenarios like these:

- You need to take a manager node down for maintenance.
- You need to increase the number of manager nodes for your swarm cluster.

Example 11-17 shows the method for promoting a node.

Example 11-17 Promoting itsoredb node

```
admin@570e9473805e:~$ docker node promote sc74cn04.pbm.ihost.com
Node sc74cn04.pbm.ihost.com promoted to a manager in the swarm.

admin@570e9473805e:~$ docker node ls
ID                            HOSTNAME                STATUS   AVAILABILITY   MANAGER STATUS  ENGINE VERSI
1xk53pdq2ujayiea2vbfu3bs1 *   sc74cn03.pbm.ihost.com  Ready    Drain          Reachable       18.09.2
cidllaskmus095rrb221ii9b9     sc74cn04.pbm.ihost.com  Ready    Active         Reachable       18.09.2
4y9a3mej93xvq4denfphhz53u     sc74cn09.pbm.ihost.com  Ready    Drain          Leader          18.09.2
j2b9596uktzhupuxv6td22mm1     sc74cn10.pbm.ihost.com  Ready    Active                         18.09.2
```

11.2.9 Demoting a manager node

We demoted sc74cn04.pbm.ihost.com server, as shown in Example 11-18. Therefore, this server does not process manager nodes operations.

Example 11-18 Demoting a manager node

```
admin@570e9473805e:~$ docker node demote sc74cn04.pbm.ihost.com
Manager sc74cn04.pbm.ihost.com demoted in the swarm.

admin@570e9473805e:~$ docker node ls
ID                            HOSTNAME                STATUS   AVAILABILITY  MANAGER STATUS  ENGINE VERSIO
1xk53pdq2ujayiea2vbfu3bs1 *   sc74cn03.pbm.ihost.com  Ready    Drain         Reachable       18.09.2
cidllaskmus095rrb221ii9b9     sc74cn04.pbm.ihost.com  Ready    Active                        18.09.2
4y9a3mej93xvq4denfphhz53u     sc74cn09.pbm.ihost.com  Ready    Drain         Leader          18.09.2
j2b9596uktzhupuxv6td22mm1     sc74cn10.pbm.ihost.com  Ready    Active                        18.09.2
```

11.2.10 Scaling down a service

Use the command that is shown in Example 11-19 to scale down your service. In this example, we set the web server service to have only four instances.

Example 11-19 Example of how to scale down service

```
admin@570e9473805e:~$ docker service ls
ID              NAME          MODE          REPLICAS    IMAGE           PORTS
pwu25hj3wpww    webserver     replicated    105/105     nginx:latest    *:80->80/tcp

admin@570e9473805e:~$ docker service scale webserver=4
webserver scaled to 4
overall progress: 4 out of 4 tasks
1/4: running   [==================================================>]
```

```
2/4: running   [==================================================>]
3/4: running   [==================================================>]
4/4: running   [==================================================>]
verify: Service converged

admin@570e9473805e:~$ docker service ls
ID              NAME         MODE           REPLICAS    IMAGE           PORTS
pwu25hj3wpww    webserver    replicated     4/4         nginx:latest    *:80->80/tcp
```

11.2.11 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, this approach 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 that you implement an odd number, 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.

A

Obtaining the additional material

Several procedures in this book reference additional material that you can download from the Internet, as described in the following sections.

The web material that is associated with this book is available in softcopy on the Internet from the IBM Redbooks web server:
ftp://www.redbooks.ibm.com/redbooks/SG248457

Alternatively, you get the material by following these steps:

1. Access the Redbooks home page: **ibm.com**/redbooks
2. Search for SG248547, select the title, and then click **Additional materials**.
   You see a directory that corresponds with the IBM Redbooks form number, SG248547.
3. Download the web material, which you need to complete several procedures in the chapters of this document. These materials are listed here:

File name	Description
`ZCXETCD.txt`	REXX code that performs these tasks: 1) Inserts a new key/value pair. 2) Queries the value of a key from the **etcd** container.
`prometheus.yml`	YAML file for the setup of Prometheus.
`IBM zCX for z_OS Monitoring-v3.json`	A dashboard that provides at-a-glance information about your environment.

4. Create a subdirectory (folder) on your workstation, and extract the contents of the web material **ZIP** file into that folder.

© Copyright IBM Corp. 2019. All rights reserved.

SG24-8457-00

ISBN 0738458155

Printed in U.S.A.

Get connected

Redbooks®
ibm.com/redbooks