IBM z/OS Container Extensions (zCX) use cases

Lydia Parziale
Marco Egli
Maike Havemann
Subhajit Maitra
Eric Marins
Edward McCarthy
Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (June 2020)

This edition applies to Version 2, Release 4, z/OS, Product number 5650-ZOS.

This document was created or updated on August 31, 2020.
## Contents

- **Notices** ................................................................. vii
- **Trademarks** ............................................................ viii
- **Preface** ................................................................. ix
- **Authors** ................................................................. ix
- **Now you can become a published author, too!** ............... x
- **Comments welcome** .................................................. xi
- **Stay connected to IBM Redbooks** ................................. xi

#### IBM zCX overview
- Why use IBM z/OS Container Extensions? .......................... 1
- IBM zCX architecture .................................................. 2
- Updates to zCX .......................................................... 3
  - zCX 90-day trial ........................................................ 3
  - Docker proxy certificate support ................................... 4
  - IBM License Manager Tool support .................................. 5
- Performance Improvements (zCX and z/OS Communication Server) .................................................. 5
- Private CA certificate support for Docker proxy .................... 6
- Vector / SIMD Support .................................................. 6

#### Part 1. Integration ....................................................... 9

**Chapter 1. Apache Kafka and ZooKeeper** .......................... 11
- 1.1 ZooKeeper overview ................................................ 13
- 1.2 ZooKeeper - configuration ......................................... 13
  - 1.2.1 Persistence with Docker volumes ............................ 13
  - 1.2.2 ZooKeeper default service ports ............................ 14
  - 1.2.3 ZooKeeper configuration file ............................... 14
- 1.3 Kafka overview ..................................................... 15
- 1.4 Kafka configuration ................................................ 16
  - 1.4.1 Persistence with Docker volumes ............................ 17
  - 1.4.2 Kafka configuration file ...................................... 17
- 1.5 Kafka configuration on one zCX instance ......................... 17
  - 1.5.1 Creating the Kafka cluster ................................... 18
- 1.6 Kafka configuration on three zCX instances ...................... 39
- 1.7 Diagnostic commands ............................................... 59
  - 1.7.1 ZooKeeper ........................................................ 60
- 1.8 Integrating IBM Z applications and Kafka ....................... 60
  - 1.8.1 Event driven application and Apache Kafka ............... 60
  - 1.8.2 Key usage patterns of Apache Kafka and IBM Z .......... 61
  - 1.8.3 Unlocking data from IBM Z to Kafka ....................... 62
- 1.9 Integration of IBM MQ with Apache Kafka ....................... 63
  - 1.9.1 Setting up MQ connectors to run on IBM z/OS .............. 64
  - 1.9.2 Starting Kafka Connect on z/OS ............................ 70
  - 1.9.3 Status of plug-ins and connectors .......................... 71
  - 1.9.4 Sending messages from MQ to Kafka ....................... 72
- 1.10 Sending CICS events to Kafka .................................... 74
  - 1.10.1 Why CICS events? ............................................ 74
  - 1.10.2 CICS to Kafka overview ..................................... 74
Chapter 2. IBM App Connect Enterprise
2.1 Technical and architectural concepts of ACE
  2.1.1 Key concepts of ACE
  2.1.2 Runtime Components of ACE
  2.1.3 ACE runtime in zCX
  2.1.4 Reasons to run ACE on zCX
2.2 Installing IBM App Connect Enterprise
  2.2.1 Create an intermediate image to pull data from GitHub
  2.2.2 Get the ACE installation binaries
  2.2.3 Build the ACE docker image
2.3 Configuration details
2.4 Deploying an application to ACE to integrate with CICS
  2.4.1 Deploy to ACE runtime in zCX
  2.4.2 Using the Web UI to test deployed REST API’s

Chapter 3. IBM Aspera fasp.io Gateway
3.1 Introduction to Aspera FASP.io Gateway
3.2 Aspera configuration details
  3.2.1 Configuration part 1 - fasp.io on the first zCX (sc74cn09) instance
  3.2.2 Configuration part 2 - fasp.io on the second zCX (sc74cn04) instance
3.3 Integration with MQ Advanced for z/OS, VUE
  3.3.1 Topology
  3.3.2 Create transmission queue on MQZ1
  3.3.3 Create sender channel on MQZ1
  3.3.4 Create receiver channel on MQZ2
  3.3.5 Create transmission queue on MQZ2
  3.3.6 Create sender channel on MQZ2
  3.3.7 Create receiver channel on MQZ1
  3.3.8 Verify the infrastructure and configuration
  3.3.9 Verify two way communication
  3.3.10 Benefits of running the FASP.IO gateway on zCX

Chapter 4. Using MQ on zCX as a client concentrator
4.1 Interconnecting MQ on z/OS with MQ in zCX
  4.1.1 Architecture
  4.1.2 Scenario
  4.1.3 Download an IBM MQ image inside zCX from dockerhub
  4.1.4 Create the queue manager inside the container
  4.1.5 Create MQ channels between the queue manager on zCX and z/OS
  4.1.6 Test the message flow from both directions
4.2 Using IBM MQ on zCX as a client concentrator
  4.2.1 IBM MQ Clients
  4.2.2 MQ client concentrator
  4.2.3 MQ on zCX as a client concentrator

Part 2. DevOps

Chapter 5. DevOps Overview
5.1 What is DevOps?
Chapter 6. Using Gitea as a code repository

6.1 Set up Gitea in zCX .................................................. 152
   6.1.1 Build a Gitea Docker image ........................................ 152
   6.1.2 Set up a private registry ........................................... 154
6.2 Running a Gitea Docker container ...................................... 155
   6.2.1 Option 1: Run a Gitea Docker container from a Dockerfile ........................................ 155
   6.2.2 Option 2: Run a Gitea Docker container from a private registry ........................................ 156
   6.2.3 Option 3: Run a Gitea Docker Container from dockerhub ........................................ 156
6.3 Check the Gitea Docker container status .................................. 156
6.4 Configure Gitea ................................................................ 157
   6.4.1 Upload Code to Gitea .................................................. 159

Chapter 7. Using Jenkins to automate builds

7.1 What is Jenkins? .......................................................... 164
7.2 Build a Jenkins server Docker image ..................................... 164
7.3 Run a Jenkins server Docker container .................................... 167
   7.3.1 Option 1: Run a Jenkins server Docker container from a Dockerfile ........................................ 167
   7.3.2 Option 2: Run a Jenkins server Docker Container from private registry ........................................ 167
   7.3.3 Option 3: Run a Jenkins server Docker Container from dockerhub ........................................ 167
7.4 Check Jenkins server Docker container status .................................. 168
7.5 Configure Jenkins ................................................................ 168
   7.5.1 Install Plugins manually ............................................... 174
7.6 Set up a Jenkins build agent ............................................... 176
   7.6.1 Create a Jenkins build node ........................................... 176
   7.6.2 Build a Jenkins agent Docker image .................................... 179
7.7 Run a Jenkins build agent Docker container .................................. 183
   7.7.1 Option 1: Run a Jenkins build agent Docker container from a Dockerfile ........................................ 183
   7.7.2 Option 2: Run a Jenkins build agent Docker container from a private registry ........................................ 183
   7.7.3 Option 3: Run a Jenkins build agent Docker Container from dockerhub ........................................ 184
7.8 Check Jenkins build agent Docker container status ............................. 184

Chapter 8. Using Ansible to automate deployment and tests

8.1 What is Ansible? .......................................................... 186
8.2 Set up Ansible using Jenkins ............................................... 186
   8.2.1 Create an Ansible playbook for deployment to a development environment ........................................ 186
   8.2.2 Create an Ansible playbook for integration tests. ........................................ 187
   8.2.3 Create an Ansible playbook for deployment to a test environment. ........................................ 187
   8.2.4 Create a deployment agent for Ansible in Jenkins ........................................ 188
8.3 Set up a Jenkins deployment agent ......................................... 188
   8.3.1 Create a Jenkins deployment node ........................................... 188
8.4 Run a Jenkins deploy agent Docker container ................................ 190
   8.4.1 Option 1: Run a Jenkins deploy agent Docker container from a Dockerfile ........................................ 190
   8.4.2 Option 2: Run a Jenkins deploy agent Docker container from a private registry ........................................ 190
   8.4.3 Option 3 Run a Jenkins deploy agent Docker Container from dockerhub ........................................ 191
8.5 Check Jenkins deploy agent Docker container status ............................. 191

Chapter 9. Putting it all together and run the pipeline

9.1 Pipeline Architecture .................................................... 194
9.2 Set up a pipeline in Jenkins ............................................... 194
9.3 Create a webhook in Gitea ................................................ 199
9.4 Run a pipeline in Jenkins .................................................. 202
9.5 Monitor a pipeline in Jenkins .............................................. 204
9.6 Summary ................................................................. 207

Part 3. Monitoring and managing ........................................... 209

Chapter 10. Monitoring ....................................................... 211
10.1 IBM Service Management Unite ....................................... 212
10.1.1 IBM Service Management Unite Automation ................. 212
10.2 IBM Service Management Unite Overview ...................... 213
10.3 Installation steps ....................................................... 214
10.4 Network and port configuration ....................................... 216
10.5 Start the SMU Image .................................................. 217
10.6 Uninstalling Service Management Unite ......................... 221
10.7 Command wrapper .................................................... 223
10.7.1 Commands provided by the SMU Docker command line utility ................. 223

Chapter 11. Container restart policy ..................................... 227
11.1 Introduction to Docker restart policies ......................... 228
11.2 Considerations ........................................................ 231

Appendix A. Additional material ......................................... 233
Locating the web material ................................................ 233
Using the web material ................................................... 233
  Downloading and extracting the web material ..................... 233

Related publications ....................................................... 235
IBM Redbooks ............................................................ 235
Other publications ........................................................ 235
Online resources .......................................................... 235
Help from IBM ............................................................ 236
Notices

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

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

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

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

IBM, the IBM logo, and ibm.com 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](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.

<table>
<thead>
<tr>
<th>AIX®</th>
<th>IBM API Connect®</th>
<th>Rational®</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aspera®</td>
<td>IBM Cloud®</td>
<td>Redbooks®</td>
</tr>
<tr>
<td>CICS®</td>
<td>IBM Z®</td>
<td>Redbooks (logo)</td>
</tr>
<tr>
<td>CICS Explorer®</td>
<td>IBM z15™</td>
<td>System z®</td>
</tr>
<tr>
<td>DB2®</td>
<td>InfoSphere®</td>
<td>WebSphere®</td>
</tr>
<tr>
<td>FASP®</td>
<td>Jazz®</td>
<td>z/OS®</td>
</tr>
<tr>
<td>IBM®</td>
<td>NetView®</td>
<td>z15™</td>
</tr>
</tbody>
</table>

The following terms are trademarks of other companies:

Evolution, are trademarks or registered trademarks of Kenexa, an IBM Company.

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.

Ansible, OpenShift, Red Hat, are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries.

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

Is it time for you to modernize your IBM z/OS® applications to allow for access to an entire ecosystem of open source and Linux on IBM® Z workloads? Is co-location of these workloads on the z/OS platform with no porting requirements of value to you?

Your existing open source or Linux on IBM Z® software can benefit from being co-located and managed inside a z/OS environment; leveraging z/OS quality of service for optimized business continuity.

Your software can be integrated with and can help complement existing z/OS workloads and environments. If your software can communicate with z/OS and external components by using TCP/IP, then now is the time to look into how IBM z/OS Container Extensions (IBM zCX) makes it possible to integrate Linux on Z applications with z/OS.

This IBM Redbooks® publication is a follow-on to Getting started with z/OS Container Extensions and Docker, SG24-8457, providing some interesting use cases for zCX.

We start with a brief overview of IBM zCX. In Part 1, “Integration” on page 9, we demonstrate use cases that integrate with zCX. In Part 2, “DevOps” on page 145, we describe how organizations can benefit from running a DevOps flow in zCX and we describe the set up of necessary components. Finally, in Part 3, “Monitoring and managing” on page 209, we discuss IBM Service Management Unite Automation, a free of charge customizable dashboard interface and a very important discussion of creating the proper container restart policy.

Authors

This was produced by a team of specialists from around the world working at IBM Redbooks, Poughkeepsie Center.

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.

Marco Egli is a Mainframe Engineer at Swiss Re, based in Switzerland. He works 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.

Maike Havemann is a Client Technical Specialist for systems hardware in Berlin, Germany. He/she has more than 5 years of experience in the IT industry and within IBM. Her areas of expertise include IBM Z as well as modern technologies in the field of AI and cloud services. She has written extensively on the DevOps chapter of this book.

Subhajit Maitra is a Consulting IT Specialist and member of the IBM America's System Z Hybrid Cloud Technical Sales team based in Hartford, CT. In addition to zCX and Red Hat OpenShift on System Z, his expertise includes IBM App Connect Enterprise, IBM Operational Decision Manager, IBM Integration Bus, IBM MQ, IBM Event Streams, IBM API Connect®,
IBM DB2® and CICS® on Systems Z. Subhajit, who has worked in IT for 27 years, has been an IBM Global Technical Ambassador for Central and Eastern Europe, where he helped IBM clients implement business-critical solutions on IBM Z servers. He has written IBM Redbooks publications about several IBM Middleware, CICS, and DevOps. He holds a master's degree in computer science from Jadavpur University, in Kolkata, India, and is an IBM zChampion.

**Eric Marins** is a Senior IT Architect in Brazil, focused on hybrid cloud solutions, Infrastructure and Platform solutions and competencies, including High Availability, Disaster Recovery, Networking, Linux and Cloud. Eric works with CIO Private Cloud designing and implementing complex hybrid cloud solutions involving Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) capabilities. 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), Getting Started with Docker Enterprise Edition on IBM Z (SG248429) and Getting started with z/OS Container Extensions and Docker (SG248457).

**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 Server such as tuning, automating administration, problem solving and integration. Prior to joining IBM in 2000, he was a CICS and 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 as various conferences.

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

Robert Haimowitz
IBM Redbooks, Poughkeepsie Center

Gus Kassimis, Gary Puchkoff, Joe Bostian, Allison Moshier,Ahilan Rajadeva
IBM USA

Matthew Leming, Mayur Raja, Katherine(Kate) Stanley, David Coles, Alison Lucas, Tony Sharkey
IBM UK

### 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 at: [ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)
Comments welcome

Your comments are important to us!

We want our to be as helpful as possible. Send us your comments about this 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:

- Stay current on recent Redbooks publications with RSS Feeds:
  http://www.redbooks.ibm.com/rss.html
IBM zCX 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 IBM Z applications, many of which were previously available only on a separately provisioned Linux instance. Most applications (including Systems Management components and development utilities/tools) that are currently available to run only on Linux will be able to run on z/OS as Docker containers.

Most container images with s390x architecture (the IBM Z opcode set) in the Docker Hub can be run in z/OS Container Extensions. The code is binary-compatible between Linux on IBM 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 and third-party software is now available for z/OS 2.4. Clients are now 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.

This chapter discusses the following:

- Section , “Why use IBM z/OS Container Extensions?” on page 2
- Section , “IBM zCX architecture” on page 2
- Section “Updates to zCX” on page 3
Why use IBM 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 zCX workloads 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, Collocation of Applications and Data, Integrated Disaster Recovery/Planned Outage Coordination, Improved Resilience, Security, and Availability.

IBM zCX architecture

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

There is limited visibility into the Linux environment. No root access is allowed, access is as defined by Docker interfaces. Additionally there is little Linux administrative overhead.

The basic architecture for zCX is described in Figure 1. zCX runs as an address space on z/OS which contains a Linux Kernel, Docker Engine and the containers that will run within that instance.
Within the zCX address space is the z/OS Linux Virtualization Layer which allows virtual access to z/OS Storage and Networking. It uses the virtio interface to communicate with the Linux kernel allowing zCX to support unmodified, open source Linux on Z kernels.

For a more detailed explanation of the zCX architecture, see *Getting started with z/OS Container Extensions and Docker*, SG24-8457.

**Updates to zCX**

Since the IBM Redbooks publication, *Getting started with z/OS Container Extensions and Docker*, SG24-8457, was published there has been some maintenance released. In this section, we provide some lists of recently released maintenance. General information about updates on the content related to zCX can be found in the IBM Knowledge Center: [https://www.ibm.com/support/z-content-solutions/container-extensions/](https://www.ibm.com/support/z-content-solutions/container-extensions/).

To get a comprehensive list of all available fixes related to IBM zCX, use the “IBM.function.zCX” Fixcat command as shown at this website: [https://www.ibm.com/support/pages/ibm-fix-category-values-and-descriptions](https://www.ibm.com/support/pages/ibm-fix-category-values-and-descriptions)

**zCX 90-day trial**

The feature listed in Table 1 became available in March 2020.
Table 1   90-day trial

<table>
<thead>
<tr>
<th>APAR</th>
<th>PTF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OA58969</td>
<td>UJ02411</td>
<td>New Function to provide zCX Trial support. Customers will be able to use zCX for a 90-day trial period. This trial period will not require the purchase of the Container Hosting Foundation (feature code 0104). (<a href="https://www.ibm.com/support/pages/apar/OA58969">https://www.ibm.com/support/pages/apar/OA58969</a>)</td>
</tr>
</tbody>
</table>

**Docker proxy certificate support**

Table 2 provides links to support for Docker proxy certificates along with a description of the support function.

Table 2   Docker proxy certificate support

<table>
<thead>
<tr>
<th>APAR</th>
<th>PTF</th>
<th>Description</th>
</tr>
</thead>
</table>
| OA58236  | UJ01141 | TRSQ (4) DEFECTS/CORRECTIONS IN ZCX DOCKER GUEST PART 1  
1) The zCX connect message in the zCX instance joblog is delayed until the zCX Docker CLI container is running.  
2) Support for proxy information for the Docker daemon is added. Proxy information is provided via the zCX provision and/or reconfigure workflows. HTTPS proxy servers protected by a common CA certificate are supported. HTTPS proxy servers protected by a private CA certificate are not supported.  
3) Support for manually setting a time zone in the zCX Docker CLI container is added.  
4) Support is updated to better track captured debug data, so that debug data is not processed multiple times.  
5) Support is added to set the zCX Docker CLI hostname to the hostname provided on the zCX provision workflow or zCX reconfigure workflow.  
6) Support is added to persist zCX Docker CLI user definitions across reconfigure and upgrade workflows.  
7) A memory leak in the zCX memory monitor caused the reboot. Disabled the memory monitor until a redesign can occur. (https://www.ibm.com/support/pages/apar/OA58236) |
| OA58237  | UJ01142 | TRSQ (4) DEFECTS/CORRECTIONS IN ZCX DOCKER GUEST PART 2  
https://www.ibm.com/support/pages/apar/OA58237 |
| OA58238  | UJ01143 | TRSQ (4) DEFECTS/CORRECTIONS IN ZCX DOCKER GUEST PART 3  
https://www.ibm.com/support/pages/apar/OA58238 |
| OA58267  | UJ01146 | Following fixes and new functions are provided with this APAR:  
1. Docker proxy configuration support via provision and reconfigure workflows  
2. ZCX_DOCKER_ADMIN_SSH_KEYS renamed to ZCX_DOCKER_ADMIN_SSH_KEY  
3. Leading slashes are required for workflow variables path/file input  
4. Trailing slashes are not allowed for workflow variables path/file input  
IBM License Manager Tool support

Table 3 provides links to support for the IBM License Manager Tool along with a description of the support function.

<table>
<thead>
<tr>
<th>APAR</th>
<th>PTF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OA58131</td>
<td>UJ01108</td>
<td>zCX is upgrading the level of Linux that is being used in the appliance. This particular upgrade required a small change in the zCX hypervisor code. This hypervisor code has been adjusted to continue to work with the current appliance as well as the new level of appliance when it is applied to the system. The GLZ procedure that is used with the START command to instantiate the zCX server has been updated to include a TIME=NOLIMIT. We believe this is a more appropriate default. However, the installation can adjust this as needed. The zCX proc can be handled like any other proc on the system. zCX STOP processing has been extended to allow the operator to issue the modify commands while the instance is stopping. This can be used to collect possible failure information while the appliance is attempting to perform the STOP. <a href="https://www.ibm.com/support/pages/apar/OA58131">https://www.ibm.com/support/pages/apar/OA58131</a></td>
</tr>
</tbody>
</table>

IBM License Manager Tool support

Table 3 provides links to support for the IBM License Manager Tool along with a description of the support function.

Table 3  IBM License Manager Tool support

<table>
<thead>
<tr>
<th>APAR</th>
<th>PTF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OA58598</td>
<td>UJ01574</td>
<td>Add ILMT support <a href="https://www.ibm.com/support/pages/apar/OA58598">https://www.ibm.com/support/pages/apar/OA58598</a></td>
</tr>
<tr>
<td>OA58621</td>
<td>UJ01576</td>
<td>ILMT disconnected scanner enablement support in zCX provisioning and reconfiguration workflows. <a href="https://www.ibm.com/support/pages/apar/OA58621">https://www.ibm.com/support/pages/apar/OA58621</a></td>
</tr>
<tr>
<td>OA58587</td>
<td>UJ01572</td>
<td>Enables zCX to capture capacity information in support of licensing metrics for containers running in the zCX hypervisor. Updates to GLZ Messages: GLZB005I z/OS system service &lt;service&gt; failed, RC=&lt;return_code&gt; RSN=&lt;reason_code&gt; &lt;optional_failure_explanation&gt;. Added value: optional_failure_explanation: Unable to obtain LPAR information. <a href="https://www.ibm.com/support/pages/apar/OA58587">https://www.ibm.com/support/pages/apar/OA58587</a></td>
</tr>
</tbody>
</table>

Performance Improvements (zCX and z/OS Communication Server)

Table 4 lists maintenance features that can provide performance improvements.
Table 4  Performance related improvements

<table>
<thead>
<tr>
<th>APAR</th>
<th>PTF</th>
<th>Description</th>
</tr>
</thead>
</table>
| OA58296| UJ02045 | zCX is adjusting various internal processing to provide improved performance. This is achieved by adjusting latch handling, adjusting timing algorithms, and refactoring code.  
https://www.ibm.com/support/pages/apar/OA58296                                                                                           |
| OA58300| UJ01511 | This APAR includes enhancements to z/OS CommServer to support zCX workloads. Highlights of the changes include:  
- Enhancements to support Inbound Workload Queueing (IWQ) for IBM z/OS Container Extensions (zCX) workloads for OSA-Express in QDIO mode.  
- Enhancements to off-load zCX network processing to zIIPs  
- zCX blocking/batching of work elements for more efficient processing of zCX traffic  
| PH16581| UI66733 | This APAR includes enhancements to z/OS CommServer to support zCX workloads. Highlights of the changes include:  
- Enhancements to support Inbound Workload Queueing (IWQ) for IBM z/OS Container Extensions (zCX) workloads for OSA-Express in QDIO mode.  
- Enhancements to off-load zCX network processing to zIIPs  
- zCX blocking/batching of work elements for more efficient processing of zCX traffic  
https://www.ibm.com/support/pages/apar/PH16581                                                                         |

Private CA certificate support for Docker proxy

Table 5 provides a list of improvements that you will use to define the CA Certificate if a proxy is used to connect via HTTPS.

Table 5  Private CA certificate support

<table>
<thead>
<tr>
<th>APAR</th>
<th>PTF</th>
<th>Description</th>
</tr>
</thead>
</table>
| OA58698| UJ02348 | Provide private CA certificate support for docker proxy configuration in IBM z/OS Container Extensions (IBM zCX).  
https://www.ibm.com/support/pages/apar/OA58698                                                                                           |
| OA58936| UJ02326 | zCX allows the specification of a HTTPS proxy server for Docker, but does not allow the user to provide the needed private CA certificate.  
https://www.ibm.com/support/pages/apar/OA58936                                                                                           |
| OA58949| UJ02327 | Refer to OA58936  
https://www.ibm.com/support/pages/apar/OA58949                                                                                           |
| OA58950| UJ02328 | Refer to OA58936  
https://www.ibm.com/support/pages/apar/OA58950                                                                                           |

Vector / SIMD Support

The APAR listed in Table 6 will allow the zCX instance to exploit the latest hardware instructions for Single-Instruction, Multiple-Data (SIMD). Unlike instructions which perform a single operation on a single data point, SIMD instructions can perform the same operation on multiple data points. Further details can be found in *SIMD Business Analytics Acceleration on z Systems*, REDP-5145.
This maintenance will involve Vector/SIMD support and make better use of hardware instructions.

Table 6  Vector/SIMD support

<table>
<thead>
<tr>
<th>APAR</th>
<th>PTF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OA59111</td>
<td>UJ03302</td>
<td>AZIF0144E An unexpected error occurred in LPAR xxxxxx GLZM009I zCX instance xxxx stored failure data</td>
</tr>
</tbody>
</table>
Integration

In this part, we demonstrate use cases that can exploit zCX. We include:

- Chapter 1, “Apache Kafka and ZooKeeper” on page 11
- Chapter 2, “IBM App Connect Enterprise” on page 83
- Chapter 3, “IBM Aspera fasp.io Gateway” on page 109
- Chapter 4, “Using MQ on zCX as a client concentrator” on page 127

These chapters demonstrate the capabilities of zCX when used for high-performance coordination, communication between applications, fast file transfer and streaming solutions, and seamless flow of multiple types of data between heterogeneous systems.
Apache Kafka and ZooKeeper

Apache Kafka is an open source, publish-and-subscribe messaging system that is built for high throughput, speed, availability, and scalability. It is an event-streaming software platform for handling real-time data feeds.

Today, billions of data sources continuously generate streams of data records, including streams of events. An event is a digital record of an action that happened and the time that it happened. Typically, an event is an action that drives another action as part of a process. A customer placing an order, choosing a seat on a flight, or submitting a registration form are all examples of events. An event doesn’t have to involve a person—for example, a connected thermostat's report of the temperature at a given time is also an event.

Kafka has three primary capabilities:

- It enables applications to publish or subscribe to data or event streams.
- It stores records accurately (i.e., in the order in which they occurred) in a fault-tolerant and durable way.
- It enables records to be processed in real-time (using something like Kafka Streams).

Kafka is managed with ZooKeeper which provides capabilities to coordinate the cluster topology and a consistent file system for configuration information.

ZooKeeper is an open source Apache project that provides a centralized service for providing configuration information, naming, synchronization and group services over large clusters in distributed systems. The goal is to make these systems easier to manage with improved, more reliable propagation of changes. Note, however, that Zookeeper is meant for use by application developers, rather than by administrators.

**Note:** Soon, Apache Kafka will not require ZooKeeper due to KIP 500. ZooKeeper is currently a dependency of Kafka rather than it being an extra technology that is required. For more information, see: https://cwiki.apache.org/confluence/display/KAFKA/KIP-500%3A+Replace+ZooKeeper+with+a+Self-Managed+Metadata+Quorum
Together, these technologies provide reliability, scalability, durability and performance for many applications.

Typically a Kafka environment is run on distributed servers as it cannot be run natively on z/OS. By using zCX, you can run Docker containers that host Kafka servers and bring this technology into z/OS.

This chapter provides a step by step guide on how to set up and start using Kafka with ZooKeeper for high availability. For a successful installation of Kafka, you must set up ZooKeeper first, hence we will discuss ZooKeeper first and then Kafka.

The following topics will be discussed:
- “ZooKeeper overview” on page 13
- “ZooKeeper - configuration” on page 13
- “ZooKeeper overview” on page 13
- “Kafka overview” on page 15
- “Diagnostic commands” on page 59
- “Integrating IBM Z applications and Kafka” on page 60
- “Integration of IBM MQ with Apache Kafka” on page 63
- “Sending CICS events to Kafka” on page 74
1.1 ZooKeeper overview

If you are running a Kafka cluster, you would need centralized management of the entire cluster in terms of leader selection, name service, locking, synchronization and configuration management. Embedding ZooKeeper means you don’t have to build synchronization services from scratch and reinvent services. Kafka uses ZooKeeper to maintain all the cluster metadata including topics, leader and followers in Zookeeper.

ZooKeeper provides an infrastructure for cross-node synchronization by maintaining status type information in memory on ZooKeeper instances. A ZooKeeper server keeps a copy of the state of the entire system and persists this information in local log files. Large Kafka clusters are supported by multiple ZooKeeper servers, with a controller server synchronizing the top-level servers.

Within ZooKeeper, Kafka can create what is called a znode (ZooKeeper data nodes), which is a file that persists in memory on the ZooKeeper servers. The znode can be updated by any node in the cluster, and any node in the cluster can register to be notified of changes to that znode.

1.2 ZooKeeper - configuration

To set up a Kafka environment first requires setting up ZooKeeper. There is no official pre-built ZooKeeper image available at present for the s390x platform. Therefore, we built a Docker image using a Dockerfile and instructions detailed in this section. The benefit of Dockerfiles is that they store the whole procedure on how an image is created.

1.2.1 Persistence with Docker volumes

ZooKeeper needs to have a persistence layer for its containers in case a container or host fails or is deleted, an environment with a shared persistence layer can be recovered and its state retrieved by restarting the container on another zCX.

zCX offers persistence via Docker volumes.

Important: Keep in mind persistence is not enabled by default, so define and enable them for production use and whenever you want your data to survive any disruptive conditions.

For each ZooKeeper container instance, create two Docker volumes for the following folders.

/conf - ZooKeeper configuration files
/data - ZooKeeper data files

Further information about the docker volume command can be found in Docker documentation which can be found at:

https://docs.docker.com/storage/volumes/

The following is an example of the docker volume command.

docker volume create [OPTIONS] [VOLUME]

In this chapter, we provide examples on how to create the required Docker volumes as we create them for our examples.
Remember to assign persistent volumes by using the `-v` parameter when you execute the command `docker run` to build your ZooKeeper containers.

### 1.2.2 ZooKeeper default service ports

This section provides information for each of the default ports used for a ZooKeeper instance. Note in Table 1-1 that we changed the port numbers to make it possible to load multiple instances on the same zCX without conflicts.

**Table 1-1  ZooKeeper default ports**

<table>
<thead>
<tr>
<th>Port Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2181</td>
<td>ZooKeeper clients to connect to the ZooKeeper servers</td>
</tr>
<tr>
<td>2888</td>
<td>Peer ZooKeeper servers to communicate with each other</td>
</tr>
<tr>
<td>3888</td>
<td>Leader election</td>
</tr>
<tr>
<td>8080</td>
<td>AdminServer</td>
</tr>
</tbody>
</table>

You may choose any ports of your liking. It is usually recommended that you use different ports on the ZooKeeper instances to avoid conflicts in a high availability environment.

Our ZooKeeper cluster consists of three containers - #1, #2 and #3. Each container uses two ports (3888 and 2888) for internal cluster communications and exposes port 2128 for clients.

Because some of the containers can be located on the same zCX instance, you can add a suffix (container ID) to each port to prevent port collisions. See Table 1-2 as an example on how to define these ports in this way.

**Table 1-2  ZooKeeper cluster ports**

<table>
<thead>
<tr>
<th>Container ID</th>
<th>Container Name</th>
<th>Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>zk-app-1</td>
<td>12181, 12888, 13888, 18080</td>
</tr>
<tr>
<td>#2</td>
<td>zk-app-2</td>
<td>22181, 22888, 23888, 28080</td>
</tr>
<tr>
<td>#3</td>
<td>zk-app-3</td>
<td>32181, 32888, 33888, 38080</td>
</tr>
</tbody>
</table>

During container creation ("docker run"), we do not expose the AdminServer port (8080) due to security reasons. The administration can be performed by using `curl` and `wget` commands inside the container.

### 1.2.3 ZooKeeper configuration file

The `/conf/zoo.cfg` file contains the configuration definition for the ZooKeeper instance and should be have the following parameters updated.

- **tickTime**: Specifies the duration (in milliseconds) at which Zookeeper checks the status of the hosts. For example:
  ```
  tickTime=2000
  ```

- **dataDir**: Specifies the directory to store the in-memory database. If this directory does not exist, create it and ensure that the user has read-write permissions. For example:
  ```
  dataDir=/data
  ```
clientPort: Specifies the port that the Zookeeper client listens on for connections. For example:

```
clientPort=2181
```

maxClientCnxns: Limits the maximum number of client connections. For example:

```
maxClientCnxns=60
```

server.n—(Optional) Specifies host names and ports for management servers in order of failover if you have replicated servers, where \( n \) identifies the main management server, followed by other servers in the order of priority for failover. For example:

```
server.1=dbhost1:2888:3888
server.2=dbhost2:2888:3888
```

A tickTime of 2000 milliseconds is the suggested interval between heartbeats. A shorter interval could lead to system overhead with limited benefits. The dataDir parameter points to the /data. Typically, ZooKeeper uses port 2181 to listen for client connections. However, we will use a different port number to identify the container instance. In most situations, 60 allowed client connections are plenty for development and testing. You can increase this setting for production systems.

For more information about ZooKeeper settings, please go to https://zookeeper.apache.org/doc/r3.1.2/zookeeperStarted.html

### 1.3 Kafka overview

The Kafka cluster stores streams of records in categories called topics. Kafka topics are feeds of messages in categories. Processes that publish messages to a topic are called producers. Processes that subscribe to topics and process the feed of published messages are called consumers. You can find more information and examples of producers and consumers at:

```
https://kafka.apache.org/intro
```

You can see all messages from all suite products, and you can consume these messages with any Kafka client implementation. Any application that works with any type of data (logs, events, and more), and requires that data be transferred can benefit from Kafka.

A consumer can direct messages to a window. You can use the consumer to see these messages. Kafka uses the Zookeeper API, which is a centralized service that maintains configuration information.

Kafka is used primarily for creating two kinds of applications:

- **Real-time streaming data pipelines**: Applications designed specifically to move millions and millions of data or event records between enterprise systems—at scale and in real-time—and move them reliably, without risk of corruption, duplication of data, and other problems that typically occur when moving such huge volumes of data at high speeds.

- **Real-time streaming applications**: Applications that are driven by record or event streams and that generate streams of their own. If you spend any time online, you encounter scores of these applications every day, from the retail site that continually updates the quantity of a product at your local store, to sites that display personalized recommendations or advertising based on click stream analysis.
Additionally, Kafka is an especially capable solution whenever you are dealing with large volumes of data and require real-time processing to make that data available to others. Kafka has the following key capabilities:

- Publish and subscribe streams of records. Data are stored so consuming applications can pull the information they need, and keep track of what they have seen so far.
- It can handle hundreds of read and write operations per second from many producers and consumers.
- Atomic broadcast, send a record once, every subscriber gets it once.
- Store streams of data records on disk and replicate within the distributed cluster for fault-tolerance. Persist data for a given time period before deleting.
- Built on top of the ZooKeeper synchronization service to keep topic, partitions and metadata highly available.

Figure 1-1 illustrates Kafka’s key components.

### 1.4 Kafka configuration

In this section we describe configuring a Kafka cluster consisting of three servers running in Docker containers on zCX. We also provide the steps that are required to set up our cluster environment on zCX.

In our environment we defined the three servers in one zCX instance which meant we needed to assign a different TCPIP port to each one as shown in Table 1-3.
1.4.1 Persistence with Docker volumes

For each Kafka container instance, you will need to create a Docker volume that will store the following folder:

- `/home/kafka/config` - Kafka configuration files
- `/home/kafka/logs/` - Kafka log files

Further information about the `docker volume` command can be found in Docker documentation which can be found at:

https://docs.docker.com/storage/volumes/

Below is an example of the `docker volume` command.

```
docker volume create [OPTIONS] [VOLUME]
```

Remember to assign persistent volumes by using the `-v` parameter when you execute the `docker run` command to build your Kafka containers.

1.4.2 Kafka configuration file

The `/home/kafka/config/server.properties` file contains the configuration definition for the Kafka instance and should have the following parameters updated:

- `broker.id`: The id of the broker. This must be set to a unique integer for each broker.
- `listeners`: The address the socket server listens on
- `advertised.listeners`: Hostname and port the broker will advertise to producers and consumers
- `zookeeper.connect`: Zookeeper connection string

For additional details about Kafka parameters, see:

https://kafka.apache.org/documentation/

1.5 Kafka configuration on one zCX instance

Once you have reviewed section 1.4.1, “Persistence with Docker volumes” on page 17 and section 1.4.2, “Kafka configuration file” on page 17, you will continue with creating a Kafka cluster. At this point, you must decide whether to configure Kafka on one zCX instance, described in this section, or on multiple zCX instances described in section 1.6, “Kafka configuration on three zCX instances” on page 39.
1.5.1 Creating the Kafka cluster

Figure 1-2 on page 18 and Table 1-4 on page 18 illustrate a one zCX use case in our IBM Redbooks environment. The IBM Z machines that we used for this lab are IBM z15™.

This section provides information on how to set up and deploy a Kafka cluster on one zCX instance that has three nodes for each application (three ZooKeeper and three Kafka).

The Docker container names we will use for this publication are zk-app-1, zk-app-2, zk-app-3 for ZooKeeper and kafka-app-1, kafka-app-2 and kafka-app-3 for Kafka. We appended the container instance number to help to identify each instance and also this number will be appended to the port number to help to allow all containers to run in one zCX instance, as shown in Figure 1-2.

![Figure 1-2 Kafka cluster diagram](image)

Table 1-4 provides an overview of our Kafka cluster information.

<table>
<thead>
<tr>
<th>Container Name</th>
<th>Instance Number</th>
<th>Application</th>
<th>Persistent Volumes (PV)</th>
<th>Application Ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>zk-app-1</td>
<td>1</td>
<td>ZooKeeper</td>
<td>zoo1-conf:/conf</td>
<td>12181, 12888,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>zoo1-data:/data</td>
<td>13888</td>
</tr>
<tr>
<td>zk-app-2</td>
<td>2</td>
<td>ZooKeeper</td>
<td>zoo2-conf:/conf</td>
<td>22181, 22888,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>zoo2-data:/data</td>
<td>23888</td>
</tr>
<tr>
<td>zk-app-3</td>
<td>3</td>
<td>ZooKeeper</td>
<td>zoo3-conf:/conf</td>
<td>32181, 32888,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>zoo3-data:/data</td>
<td>33888</td>
</tr>
<tr>
<td>kafka-app-1</td>
<td>1</td>
<td>Kafka</td>
<td>kafka1-conf:/home/kafka</td>
<td>19092</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>config/kafka1-logs:/home/kafka/logs</td>
<td></td>
</tr>
</tbody>
</table>
Notice that the container names, persistent volumes and application ports have the instance number appended to them to support the running of all containers in a single zCX host. This can also be done when using multiple zCX instances.

Important: Be aware that setting up multiple servers on a single machine will not create any redundancy and your environment will be vulnerable in case of a host failure. If something were to happen which caused the machine to stop, all of the ZooKeeper servers would be offline. Full redundancy requires that each container have its own zCX instance and on separate pieces of hardware.

We will deploy ZooKeeper and Kafka containers to sc74cn09.pbm.ihost.com (129.40.23.76). We preformed the instructions found in the next section, “Create ZooKeeper file”, while connected to our system named sc74cn09.

Create ZooKeeper file

No official ZooKeeper Docker image is available on the Docker hub at this time. The steps described in this section will build a ZooKeeper image to run on the IBM Z platform. For the latest Dockerfile examples for ZooKeeper, see:

https://github.com/linux-on-ibm-z/dockerfile-examples

The following steps outline how to create the ZooKeeper file.

1. Create a folder and download the two required files from github, Use the commands shown in Example 1-1 on your zCX instance.

Example 1-1 Create the ZooKeeper file

```
mkdir -p /home/admin/zookeeper
cd zookeeper

chmod 755 docker-entrypoint.sh
```
2. Edit the `Dockerfile.zookeeper` file with your details as shown Example 1-2 and highlighted in red.

   **Example 1-2  Dockerfile.zookeeper file**

   ```
   FROM s390x/ubuntu:18.04
   ENV JAVA_HOME=/usr/lib/jvm/java-11-openjdk-s390x
   ENV ZOO_CONF_DIR=/conf 
       ZOO_DATA_DIR=/data 
       ZOO_DATA_LOG_DIR=datalog 
       ZOO_LOG_DIR=/logs 
       ZOO_TICK_TIME=2000 
       ZOO_INIT_LIMIT=5 
       ZOO_SYNC_LIMIT=2 
       ZOO_AUTOPURGE_PURGEINTERVAL=0 
       ZOO_AUTOPURGE_SNAPRETAINCOUNT=3 
       ZOO_MAX_CLIENT_CNXNS=60 
       ZOO_STANDALONE_ENABLED=true 
       ZOO_ADMINSERVER_ENABLED=true
   # Add a user with an explicit UID/GID and create necessary directories
   RUN set -eu; 
       groupadd -r zookeeper --gid=1000; 
       useradd -r -g zookeeper --uid=1000 zookeeper; 
       mkdir -p "$ZOO_DATA_LOG_DIR" "$ZOO_DATA_DIR" "$ZOO_CONF_DIR" "$ZOO_LOG_DIR"; 
       chown zookeeper:zookeeper "$ZOO_DATA_LOG_DIR" "$ZOO_DATA_DIR" "$ZOO_CONF_DIR" "$ZOO_LOG_DIR"
   # Install required packages
   RUN set -eu; 
       apt-get update; 
       DEBIAN_FRONTEND=noninteractive 
       apt-get install -y --no-install-recommends 
       openjdk-11-jre 
       tar 
       wget 
   ```

   **Note:** If you run into problems using the curl commands to download the required files, download these files from a Linux server and then upload to your zCX instance by using the `scp` or `ssh` commands. Use the following instructions:

   a. Download the Dockerfile and put into /tmp/ and name it as `Dockerfile.zookeeper`
   b. Download `docker-entrypoint.sh` and put into /tmp/
   c. Issue the following commands to upload

   ```
   cat /tmp/Dockerfile.zookeeper | ssh -p 8022 admin@<zcx IP address> "cat > /home/admin/Dockerfile.zookeeper"
   cat /tmp/docker-entrypoint.sh | ssh -p 8022 admin@<zcx IP address> "cat > /home/admin/docker-entrypoint.sh"
   ```

   The Dockerfile.zookeeper file is written to: `/home/admin/Dockerfile.zookeeper`

   The `docker-entrypoint.sh` file is written to: `/home/admin/docker-entrypoint.sh`
vim \
ca-certificates \
dirmngr \
gosu \
gnu\ 
netcat \
rm -rf /var/lib/apt/lists/*;

# Verify that gosu binary works
gosu nobody true

ARG GPG_KEY=BBE7232D7991050B54C8EA0ADC08637CA615D22C
ARG SHORT_DISTRO_NAME=zookeeper-3.6.1
ARG DISTRO_NAME=apache-zookeeper-3.6.1-bin

# Download Apache Zookeeper, verify its PGP signature, untar and clean up
RUN set -eux; \
ddist() { \
    local f="$1"; shift; \
    local distFile="$1"; shift; \
    local success=; \
    local distUrl=; \
    for distUrl in \n        'https://www.apache.org/dyn/closer.cgi?action=download&filename=' \n        https://www-us.apache.org/dist/ \n        https://www.apache.org/dist/ \n        https://archive.apache.org/dist/ \n    ; do \n        if wget -q -O "$f" "$distUrl$distFile" && [ -s "$f" ]; then \n            success=1; \
            break; \n        fi; \n    done; \
    [ -n "$success" ]; \
}; \
ddist "$DISTRO_NAME.tar.gz"
"zookeeper/$SHORT_DISTRO_NAME/$DISTRO_NAME.tar.gz"; 
    ddist "$DISTRO_NAME.tar.gz.asc"
"zookeeper/$SHORT_DISTRO_NAME/$DISTRO_NAME.tar.gz.asc"; 
    export GNUPGHOME="$(/mktemp -d)"; \
    gpg --keyserver ha.pool.skgs-keyservers.net --recv-key "$GPG_KEY" || \
    gpg --keyserver pgp.mit.edu --recv-keys "$GPG_KEY" || \
    gpg --keyserver keys.pgp.com --recv-keys "$GPG_KEY"; \
    gpg --batch --verify "$DISTRO_NAME.tar.gz.asc" "$DISTRO_NAME.tar.gz"; \
    tar -zxf "$DISTRO_NAME.tar.gz"; \
    mv "$DISTRO_NAME/conf/** "$ZOO_CONF_DIR"; \
    rm -rf "$GNUPGHOME" "$DISTRO_NAME.tar.gz" "$DISTRO_NAME.tar.gz.asc"; \
    chown -R zookeeper:zookeeper "/$DISTRO_NAME"

WORKDIR $DISTRO_NAME
VOLUME ["$ZOO_DATA_DIR", "$ZOO_DATA_LOG_DIR", "$ZOO_LOG_DIR"]

#EXPOSE 2181 2888 3888 8080

ENV PATH=$PATH:/$DISTRO_NAME/bin \
    ZOOCFGDIR=$ZOO_CONF_DIR
COPY docker-entrypoint.sh /
ENTRYPOINT ["/docker-entrypoint.sh"]
CMD ["zkServer.sh", "start-foreground"]

**Note:** Writing EXPOSE in the Dockerfile is merely a hint that a certain port is useful. Docker won’t do anything with that information by itself.

Important to note is that ZooKeeper ports will be different for each container instance and therefore we can safely comment out the `EXPOSE 2181 2888 3888 8080` line from this file.

3. Issue the following `docker build` command to build the ZooKeeper image.

   ```
   docker build -t zcx-zookeeper-img -f Dockerfile.zookeeper .
   ```

   We named the new image `zcx-zookeeper-img`.

4. Create two persistent volumes for each zookeeper instance by using the following commands:

   For zk-app-1:
   ```
   docker volume create zoo1-conf
   docker volume create zoo1-data
   ```

   For zk-app-2:
   ```
   docker volume create zoo2-conf
   docker volume create zoo2-data
   ```

   For zk-app-3:
   ```
   docker volume create zoo3-conf
   docker volume create zoo3-data
   ```

   Use the following steps to populate the volumes for each container instance. We use a dummy temporary image to help with it.

1. For zk-app-1, execute the commands shown in Example 1-3:

   ```
   Example 1-3 Populate the volumes for zk-app-1
   docker run --name dummy  -v zoo1-conf:/conf1 -v zoo1-data:/data1 -d
   zcx-zookeeper-img
   docker exec -it dummy bash
   cp -a /conf/* /conf1/
   cp -a /data/* /data1/
   apt-get update
   apt install -y vim
   ```

   Edit `conf1/zoo.cfg` file by using the `vi` command with the following instructions.

   a. Delete the following line:
      ```
      server.1=localhost:2888:3888:2181
      ```

   b. Update the following values:
initLimit=10
syncLimit=5
standaloneEnabled=false
c. Add the following lines:
server.1=0.0.0.0:12888:13888;12181
server.2=129.40.23.76:22888:23888;22181
server.3=129.40.23.76:32888:33888;32181

After the updates, the zoo.cfg file should look similar to that shown in Example 1-4:

```
Example 1-4   /conf1/zoo.cfg file

dataDir=/data
dataLogDir=/datalog
tickTime=2000
initLimit=10
syncLimit=5
autopurge.snapRetainCount=3
autopurge.purgeInterval=0
maxClientCnxns=60
standaloneEnabled=false
admin.enableServer=true
server.1=0.0.0.0:12888:13888;12181
server.2=129.40.23.76:22888:23888;22181
server.3=129.40.23.76:32888:33888;32181
```
d. Edit /data1/myid file using vi and update with the number 1 for our first ZooKeeper instance.
e. Execute the `exit` command to leave the dummy container
f. Execute the command `docker rm dummy --force` to delete the dummy container.

2. For zk-app-2, execute the commands shown in Example 1-5:

```
Example 1-5   Populate the volumes for zk-app-2

docker run --name dummy -v zoo2-conf:/conf2 -v zoo2-data:/data2 -d zcx-zookeeper-img
docker exec -it dummy bash
cp -a /conf/* /conf2/
cp -a /data/* /data2/
apt-get update
apt install -y vim
```

Edit /conf2/zoo.cfg file by using the `vi` command with the following instructions:
a. Delete the following line:
server.1=localhost:2888:3888;2181 line
b. Update the following values:
initLimit=10
syncLimit=5
standaloneEnabled=false
c. Add the following lines:

```plaintext
server.1=129.40.23.76:12888:13888;12181
server.2=0.0.0.0:22888:23888;22181
server.3=129.40.23.76:32888:33888;32181
```

After the updates, zoo.cfg file should look similar to that shown in Example 1-6:

```plaintext
Example 1-6   /conf2/zoo.cfg file

dataDir=/data
dataLogDir=/datalog
tickTime=2000
initLimit=10
syncLimit=5
autopurge.snapRetainCount=3
autopurge.purgeInterval=0
maxClientCnxns=60
standaloneEnabled=false
admin.enableServer=true
server.1=129.40.23.76:12888:13888;12181
server.2=0.0.0.0:22888:23888;22181
server.3=129.40.23.76:32888:33888;32181
```

d. Edit `/data2/myid` file using the command `vi` and update with the number 2 for our second ZooKeeper instance.

e. Execute the `exit` command to leave the dummy container.

f. Execute the command `docker rm dummy --force` to delete the dummy container.

3. For zk-app-3, execute the commands shown in Example 1-7:

```plaintext
Example 1-7   Populate the volumes for zk-app-3

docker run --name dummy -v zoo3-conf:/conf3 -v zoo3-data:/data3 -d zcx-zookeeper-img
docker exec -it dummy bash
cp -a /conf/* /conf3/
cp -a /data/* /data3/
apt-get update
apt install -y vim
```


i. Delete the following line:

```plaintext
server.1=localhost:2888:3888;2181 line
```

ii. Update the following values:

```plaintext
initLimit=10
syncLimit=5
standaloneEnabled=false
```

iii. Add the following lines:

```plaintext
server.1=129.40.23.76:12888:13888;12181
```
After the updates zoo.cfg file should look similar to that shown in Example 1-8.

```
Example 1-8 /conf3/zoo.cfg file

dataDir=/data
dataLogDir=/datalog
tickTime=2000
initLimit=10
syncLimit=5
autopurge.snapRetainCount=3
autopurge.purgeInterval=0
maxClientCnxns=60
standaloneEnabled=false
admin.enableServer=true
server.1=129.40.23.76:12888:13888;12181
server.2=129.40.23.76:22888:23888;22181
server.3=0.0.0.0:32888:33888;32181
```

b. Edit the /data3/myid file using the command `vi` and update with the number 3 for our third ZooKeeper instance.

c. Execute the `exit` command to leave the dummy container

d. Execute the command `docker rm dummy --force` to delete the dummy container.

4. Start your three ZooKeeper Instances. Since Zookeeper "fails fast" it's better to always restart it, we appended `--restart always` in the docker run commands shown in Example 1-9.

```
Example 1-9 Docker run commands


docker run --name zk-app-3 -p 32181:32181 -p 32888:32888 -p 33888:33888 -v zoo3-conf:/conf:rw -v zoo3-data:/data:rw --restart always -d zcx-zookeeper-img
```

5. Issue the following commands to verify the startup of each container:

```
docker logs zk-app-1
docker logs zk-app-2
docker logs zk-app-3
```

For each command, verify that you receive a message such as that shown in Example 1-10.

```
Example 1-10 Example messages of successful start of each container

2020-06-09 16:41:30,773 [myid:1] - INFO [main:ContextHandler@825] - Started o.e.j.s.ServletContextHandler@33ecda92{/,null,AVAILABLE}
2020-06-09 16:41:30,778 [myid:1] - INFO [main:AbstractConnector@330] - Started ServerConnector@4c60d6e9[HTTP/1.1,[http/1.1]]{0.0.0.0:8080}
2020-06-09 16:41:30,778 [myid:1] - INFO [main:Server@399] - Started @975ms
```
6. For zk-app-1 instance, execute the following commands to confirm ZooKeeper is operational.

```bash
docker exec -it zk-app-1 bash -c "wget -O -
http://localhost:8080/commands/stats"
```

You should receive output similar to that shown in Example 1-11. Highlighted in red is the state of each instance. One instance should be a leader and the others followers.

**Example 1-11 Confirmation of follower instance**

```bash
admin@16bbba5088fb:~/kafka$ docker exec -it zk-app-1 bash -c "wget -O -
http://localhost:8080/commands/stats"
```

```
Resolving localhost (localhost)... 127.0.0.1, ::1
Connecting to localhost (localhost)|127.0.0.1|:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 923 [application/json]
Saving to: 'STDOUT'

-                     0%
0   --.-KB/s
```

```json
{
    "version": "3.6.1--104dcb3e3fb464b30c5186d229e00af9f332524b, built on 04/21/2020 15:01 GMT",
    "read_only": false,
    "server_stats": {
        "packets_sent": 0,
        "packets_received": 0,
        "fsync_threshold_exceed_count": 0,
        "client_response_stats": {
            "last_buffer_size": -1,
            "min_buffer_size": -1,
            "max_buffer_size": -1
        },
        "uptime": 19944686,
        "provider_null": false,
        "server_state": "follower",
        "outstanding_requests": 0,
        "min_latency": 0,
        "avg_latency": 0.0,
        "max_latency": 0,
        "data_dir_size": 67108880,
        "log_dir_size": 459,
        "last_processed_zxid": 4294967388,
        "num_alive_client_connections": 0
    },
    "client_response": {
        "last_buffer_size": -1,
        "min_buffer_size": -1,
        "max_buffer_size": -1
    },
    "node_count": 55,
    "connections": [],
    "secure_connections": []
}
```
7. For zk-app-2 instance, execute the following commands to confirm ZooKeeper is operational.

```
docker exec -it zk-app-2 bash -c "wget -O - http://localhost:8080/commands/stats"
```

You should receive similar output as shown in Example 1-12. Highlighted in red is the state of each instance. One instance should be a leader and the others followers.

Example 1-12 Confirmation of leader instance

```
admin@16bbba5088fb:~/kafka$ docker exec -it zk-app-2 bash -c "wget -O - http://localhost:8080/commands/stats"
Resolving localhost (localhost)... 127.0.0.1, ::1
Connecting to localhost (localhost)|127.0.0.1|:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1223 (1.2K) [application/json]
Saving to: 'STDOUT'

  100%[===================================>] 923 --.-KB/s in 0s
2020-06-09 22:13:55 (356 MB/s) - written to stdout [923/923]
```

```
"command" : "stats",
"error" : null
- 100%[===================================>] 923 --.-KB/s in 0s
```

```
7. For zk-app-2 instance, execute the following commands to confirm ZooKeeper is operational.

docker exec -it zk-app-2 bash -c "wget -O - http://localhost:8080/commands/stats"
```

You should receive similar output as shown in Example 1-12. Highlighted in red is the state of each instance. One instance should be a leader and the others followers.

Example 1-12 Confirmation of leader instance

```
admin@16bbba5088fb:~/kafka$ docker exec -it zk-app-2 bash -c "wget -O - http://localhost:8080/commands/stats"
Resolving localhost (localhost)... 127.0.0.1, ::1
Connecting to localhost (localhost)|127.0.0.1|:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1223 (1.2K) [application/json]
Saving to: 'STDOUT'

- 0%[===================================>] 923 --.-KB/s
```

```
"version" : "3.6.1--104dcb3e3fb464b30c5186d229e00af9f332524b, built on
04/21/2020 15:01 GMT",
"read_only" : false,
"server_stats" : {
  "packets_sent" : 24877,
  "packets_received" : 24877,
  "fsync_threshold_exceed_count" : 0,
  "client_response_stats" : {
    "last_buffer_size" : 16,
    "min_buffer_size" : 16,
    "max_buffer_size" : 196
  },
  "provider_null" : false,
  "uptime" : 112174649,
  "server_state" : "leader",
  "outstanding_requests" : 0,
  "min_latency" : 0,
  "avg_latency" : 0.4014,
  "max_latency" : 6009,
  "data_dir_size" : 67108880,
  "log_dir_size" : 459,
  "last_processed_zxid" : 4294967388,
  "num_alive_client_connections" : 1
},
"client_response" : {
  "last_buffer_size" : 16,
  "min_buffer_size" : 16,
  "max_buffer_size" : 196
```

8. For zk-app-3 instance, execute the following commands to confirm ZooKeeper is operational.

   `docker exec -it zk-app-3 bash -c "wget -O - http://localhost:8080/commands/stats"`

   You should receive output shown in Example 1-13. Highlighted in red is the state of each instance. One instance should be a leader and the others followers.

   **Example 1-13  Confirmation of second follower instance**

   ```
   admin@16bbba5088fb:~/kafka$ docker exec -it zk-app-3 bash -c "wget -O - http://localhost:8080/commands/stats"
   Resolving localhost (localhost)... 127.0.0.1, ::1
   Connecting to localhost (localhost)|127.0.0.1|:8080... connected.
   HTTP request sent, awaiting response... 200 OK
   Length: 1274 (1.2K) [application/json]
   Saving to: 'STDOUT'
   -                     0%[                    ]       0  --.-KB/s
   100%[======================================================> 1.19K --.-KB/s in 0s
   2020-06-09 22:21:46 (29.1 MB/s) - written to stdout [1223/1223]
   ```

   ```
   "version" : "3.6.1--104dcb3e3fb464b30c5186d229e00af9f332524b, built on 04/21/2020 15:01 GMT",
   "read_only" : false,
   "server_stats" : {
     "packets_sent" : 9858,
     "packets_received" : 9858,
     "fsync_threshold_exceed_count" : 0,
     "client_response_stats" : {
       "last_buffer_size" : 16,
       "min_buffer_size" : 16,
       "max_buffer_size" : 16
     }
   }
   ```
Zookeeper installation is completed. In the next section, “Kafka Docker images”, you will start the installation of Kafka.

**Kafka Docker images**

In this section, we provide the steps to create images in order to install Kafka. For the most current Dockerfile examples for Kafka, see:

https://github.com/linux-on-ibm-z/dockerfile-examples

1. Use the following commands to create a folder for the Kafka image.

   ```bash
   mkdir /home/admin/kafka
   cd /home/admin/kafka
   ```

2. Download the Kafka Dockerfile using the following command:

   ```bash
   curl -k -o Dockerfile.kafka
   ```
3. Edit the file `Dockerfile.kafka` to look similar to that shown in Example 1-14. Highlighted in red is what is required to update, add or remove.

```bash
Example 1-14  Dockerfile.kafka

########## dockerfile for Apache Kafka version 2.5.0 ##########
#
# This Dockerfile builds a basic installation of Apache Kafka.
#
# Kafka is run as a cluster on one or more servers. The Kafka cluster stores
# streams of records in categories called topics.
# Each record consists of a key, a value, and a timestamp.
# In Kafka the communication between the clients and the servers is done with a
# simple, high-performance, language agnostic TCP protocol.
#
# To build this image, from the directory containing this Dockerfile
# (assuming that the file is named Dockerfile):
# docker build -t <image_name> .
#
# To Start Apache Kafka run the below command:
# docker run --name <container_name> -d <image>
#
# To check Apache kafka is running, Enter below command:
# docker exec <container_id of kafka> <any kafka related command>
# Eg. To list topic and message files:
#   docker exec <container_id of kafka> bin/kafka-topics.sh --list --zookeeper
# localhost:2181
#
# Reference:
# http://kafka.apache.org/
# https://kafka.apache.org/quickstart
#
# Base Image
FROM s390x/ubuntu:16.04
#
# The author
```

Note: If you run into problems using the curl commands to download the required files, download these files from a Linux server and then upload to your zCX instance by using the `scp` or `ssh` commands. Use the following instructions:

a. Download the Dockerfile and put into `/tmp/` and name it as `Dockerfile.kafka`
b. Download `docker-entrypoint.sh` and put into `/tmp/`
c. Issue the following commands to upload

```
cat /tmp/Dockerfile.kafka | ssh -p 8022 admin@<zcx IP address> "cat > /home/admin/Dockerfile.kafka"
```

Dockerfile.kafka file is written to: `/home/admin/Dockerfile.kafka`
LABEL maintainer="LoZ Open Source Ecosystem
(https://www.ibm.com/developerworks/community/groups/community/lozopensource)"

ENV SOURCE_DIR=/home/
ENV JAVA_HOME=/home/jdk-11.0.5+10
ENV PATH=$PATH:$SOURCE_DIR:$JAVA_HOME/bin
ENV VERSION=2.12-2.5.0

WORKDIR $SOURCE_DIR

# Install dependencies
RUN apt-get update && apt-get -y install \
    git \
    unzip \
    wget \
    vim \
    net-tools \
    kafkacat \
    # Download Adopt JDK
    && wget https://github.com/AdoptOpenJDK/openjdk11-binaries/releases/download/jdk-11.0.5 \
       %2B10/OpenJDK11U-jdk_s390x_linux_hotspot_11.0.5_10.tar.gz \ 
    && tar -xvzf OpenJDK11U-jdk_s390x_linux_hotspot_11.0.5_10.tar.gz \ 
    && rm OpenJDK11U-jdk_s390x_LINUX_hotspot_11.0.5_10.tar.gz \ 
    # Download the Apache Kafka Binary
    && wget http://mirrors.estointernet.in/apache/kafka/2.5.0/kafka_${VERSION}.tgz \
    && tar -xvzf kafka_${VERSION}.tgz \ 
    && rm kafka_${VERSION}.tgz \ 
    && mv kafka_${VERSION} kafka
    # Expose ports for Apache ZooKeeper and kafka
    # EXPOSE 2181 9092

WORKDIR $SOURCE_DIR/kafka/

# start zookeeper and kafka server
CMD bin/kafka-server-start.sh config/server.properties

# End of Dockerfile

Note: Writing EXPOSE in Dockerfile, is merely a hint that a certain port is useful. Docker
won't do anything with that information by itself.

4. To build a new Kafka image and tag it as zcx-kafka-img, issue the following command:
   docker build -t zcx-kafka-img -f Dockerfile.kafka .
   You will receive the follow messages if build completes successfully.
   Successfully built 3ac1c714ee08
   Successfully tagged zcx-kafka-img:latest

5. Define Kafka volume groups for each Kafka container instance and logs using the
   following commands:
   docker volume create kafka1-conf
   docker volume create kafka1-logs
docker volume create kafka2-conf
docker volume create kafka2-logs
docker volume create kafka3-conf
docker volume create kafka3-logs

Doing this will make the configuration files persistent for Kafka, allowing your system to survive during an upgrade. For additional information about persistent volumes, please see the IBM Redbooks publication, *Getting started with z/OS Container Extensions and Docker*, SG24-8457.

Use the following steps to create the configuration file for each Kafka container image.

1. For kafka-app-1, execute the commands shown in Example 1-15:

   **Example 1-15  Create the configuration file for kafka-app1**
   
   ```
   docker run --name dummy -it --entrypoint /bin/bash -v
   kafka1-conf:/home/kafka/config1 -d zcx-kafka-img
   docker exec -it dummy bash
   cp -a /home/kafka/config/* /home/kafka/config1/
   apt-get update
   apt install -y vim
   ```

   a. Edit the `/home/kafka/config1/server.properties` file with the `vim` command by using the following steps:

   **Note:** 129.40.23.76 is the IP address of our zCX host

   i. Update the following values:

   ```
   broker.id=1
   zookeeper.connect=129.40.23.76:12181,129.40.23.76:22181,129.40.23.76:32181
   ```

   ii. Add these lines (under Socket Server Settings)

   ```
   listeners=PLAINTEXT://0.0.0.0:19092
   advertised.listeners=PLAINTEXT://129.40.23.76:19092
   ```

   After the updates, `/home/kafka/config1/server.properties` file should look like those shown in Example 1-16

   **Example 1-16  `/home/kafka/config1/server.properties`**

   ```
   # The id of the broker. This must be set to a unique integer for each broker.
   broker.id=1

   # The address the socket server listens on. It will get the value returned from
   # java.net.InetAddress.getCanonicalHostName() if not configured.
   # FORMAT:
   # listeners = listener_name://host_name:port
   # EXAMPLE:
   # listeners = PLAINTEXT://your.host.name:9092
   ```
# listeners=PLAINTEXT://:9092
listeners=PLAINTEXT://0.0.0.0:9092
advertised.listeners=PLAINTEXT://129.40.23.76:9092

*** OUTPUT OMITTED ***

############################# Zookeeper #############################

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=129.40.23.76:12181,129.40.23.76:22181,129.40.23.76:32181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=18000

*** OUTPUT OMITTED ***

iii. Issue `exit` command to leave the dummy container

iv. Issue `docker rm dummy --force` command to delete the dummy container.

2. For kafka-app-2, execute the commands shown in Example 1-17:

   ```bash
   Example 1-17  Create the configuration file for kafka-app-2
   
   docker run --name dummy -it --entrypoint /bin/bash -v
   kafka2-conf:/home/kafka/config2 -d zcx-kafka-img
   docker exec -it dummy bash
   cp -a /home/kafka/config/* /home/kafka/config2/
   apt-get update
   apt install -y vim
   
   a. Edit the `/home/kafka/config2/server.properties` file by using the `vi` command to perform
   the following steps.
   
   Note: 129.40.23.76 is the IP address of our zCX host.
   
   i. Update the following values:
   
   broker.id=2
   zookeeper.connect=129.40.23.76:12181,129.40.23.76:22181,129.40.23.76:32181
   
   ii. Add these lines (under Socket Server Settings)
   
   listeners=PLAINTEXT://0.0.0.0:29092
   advertised.listeners=PLAINTEXT://129.40.23.76:29092
   
   After the updates, `/home/kafka/config2/server.properties` file should look like those
   shown in Example 1-18.
   
   Example 1-18  `/home/kafka/config2/server.properties`
   
   ####################################################################
   
   Chapter 1. Apache Kafka and ZooKeeper
# The id of the broker. This must be set to a unique integer for each broker.
broker.id=2

### Socket Server Settings

# The address the socket server listens on. It will get the value returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://0.0.0.0:29092
advertised.listeners=PLAINTEXT://129.40.23.76:29092

*** OUTPUT OMMITED ***

### Zookeeper

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=129.40.23.76:12181,129.40.23.76:22181,129.40.23.76:32181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=18000

*** OUTPUT OMMITED ***

iii. Issue the `exit` command to leave the dummy container

iv. Issue the `docker rm dummy --force` command to delete the dummy container.

3. For kafka-app-3, execute the commands shown in Example 1-19:

   **Example 1-19   Create the configuration file for kafka-app-3**

   ```
   docker run --name dummy -it --entrypoint /bin/bash -v
   kafka3-conf:/home/kafka/config3 -d zcx-kafka-img
   docker exec -it dummy bash
   cp -a /home/kafka/config/* /home/kafka/config3/
   apt-get update
   apt install -y vim
   ```

   a. Edit the `/home/kafka/config3/server.properties` file using the `vi` command with the following instructions.

   **Note:** 129.40.23.76 is the IP address of our zCX host

   i. Update the following values:

   broker.id=3
zookeeper.connect=129.40.23.76:12181,129.40.23.76:22181,129.40.23.76:32181

ii. Add the following lines (under Socket Server Settings)

`listeners=PLAINTEXT://0.0.0.0:39092`
`advertised.listeners=PLAINTEXT://129.40.23.76:39092`

After the updates, `/home/kafka/config3/server.properties` file should look like that shown in Example 1-20.

```
Example 1-20  /home/kafka/config3/server.properties

# The id of the broker. This must be set to a unique integer for each broker.
broker.id=3

# Socket Server Settings

# The address the socket server listens on. It will get the value returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
# FORMAT:
#   listeners = listener_name://host_name:port
# EXAMPLE:
#   listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://0.0.0.0:39092
advertised.listeners=PLAINTEXT://129.40.23.76:39092

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=129.40.23.76:12181,129.40.23.76:22181,129.40.23.76:32181
# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=18000
```

iii. Issue the `exit` command to leave the dummy container

iv. Issue the `docker rm dummy --force` command to delete the dummy container

4. Now, bring your kafka instances up by issuing the following commands:

```
docker run --name kafka-app-1 -p 19092:19092 -v kafka1-conf:/home/kafka/config -v kafka1-logs:/home/kafka/logs -d zcx-kafka-img
docker run --name kafka-app-2 -p 29092:29092 -v kafka2-conf:/home/kafka/config -v kafka2-logs:/home/kafka/logs -d zcx-kafka-img
docker run --name kafka-app-3 -p 39092:39092 -v kafka3-conf:/home/kafka/config -v kafka3-logs:/home/kafka/logs -d zcx-kafka-img
```

Hint: The Dockerfile.kafka file was updated to install the `kafkacat` utility that you can use to test and debug Apache Kafka.
5. Check the Kafka container logs to see if Kafka was successfully started by using the following command:

`docker logs kafka-app-1`

You should receive output that shows `brokerID=1` was started, as shown in Example 1-21.

```
Example 1-21 Logs for kafka-app-1
[2020-06-10 11:56:17,434] INFO [Transaction Marker Channel Manager 1]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2020-06-10 11:56:17,744] INFO Kafka version: 2.5.0 (org.apache.kafka.common.utils.AppInfoParser)
[2020-06-10 11:56:17,746] INFO Kafka commitId: 66563e712b0b9f84 (org.apache.kafka.common.utils.AppInfoParser)
```

6. Check the second Kafka container with the following command:

`docker logs kafka-app-2`

You should receive output that shows `brokerID=2` was started, as shown in Example 1-22.

```
Example 1-22 Logs for kafka-app-2
[2020-06-10 11:56:18,003] INFO [Transaction Marker Channel Manager 2]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
[2020-06-10 11:56:18,087] INFO Kafka version: 2.5.0 (org.apache.kafka.common.utils.AppInfoParser)
[2020-06-10 11:56:18,090] INFO Kafka commitId: 66563e712b0b9f84 (org.apache.kafka.common.utils.AppInfoParser)
```

7. Now check the third Kafka container with the following command:

`docker logs kafka-app-3`

You should receive output that shows `brokerID=3` was started as shown in Example 1-23.
Example 1-23  Logs for kafka-app-3

[2020-06-10 11:56:18,242] INFO [Transaction Marker Channel Manager 3]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)

All Kafka container instances are up and running with the correct broker IDs.

Use the following steps to optionally perform a high availability test to ensure everything is fine.

1. To do this, first check if the cluster is healthy. Check the first Kafka instance by using the following command:

   `docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.76:19092 -L"`

   You should receive the output shown in Example 1-24.

Example 1-24  Kafka-app-1 check

```
admin@sc74cn09:~$ docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.76:19092 -L"
```

```
Metadata for all topics (from broker -1: 129.40.23.76:19092/bootstrap):
  3 brokers:
    broker 2 at 129.40.23.76:29092
    broker 3 at 129.40.23.76:39092
    broker 1 at 129.40.23.76:19092
  0 topics:
```

2. Check if the second Kafka instance is accessible by using the following command:

   `docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.76:29092 -L"`

   You should receive the output shown in Example 1-25.

Example 1-25  Kafka-app-2 check

```
admin@sc74cn09:~$ docker exec -it kafka-app-2 bash -c "kafkacat -b 129.40.23.76:29092 -L"
```

```
Metadata for all topics (from broker -1: 129.40.23.76:29092/bootstrap):
  3 brokers:
    broker 2 at 129.40.23.76:29092
    broker 3 at 129.40.23.76:39092
    broker 1 at 129.40.23.76:19092
  0 topics:
```
3. Check if the third Kafka instance is accessible by using the following command:

```
docker exec -it kafka-app-3 bash -c "kafkacat -b 129.40.23.76:39092 -L"
```

You should receive the output shown in Example 1-26.

```
Example 1-26 Kafka-app-3 check
admin@sc74cn09:~$ docker exec -it kafka-app-3 bash -c "kafkacat -b 129.40.23.76:39092 -L"
Metadata for all topics (from broker -1: 129.40.23.76:39092/bootstrap):
  3 brokers:
    broker 2 at 129.40.23.76:29092
    broker 3 at 129.40.23.76:39092
    broker 1 at 129.40.23.76:19092
  0 topics:
admin@sc74cn09:~$
```

All of the output from each check shows that all three of the Kafka instances can be accessed via the zCX IP address.

To verify that the cluster can detect the state of the Kafka instances, take one of the instances down. To take kafka-app-2 down, issue the following command:

```
docker stop kafka-app-2
```

To check if the cluster detected a problem on kafka-app-2 and removed this broker from the cluster, issue the `kafkacat` command on kafka-app-1 or kafka-app-3 to check the cluster. Use the following command:

```
docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.76:19092 -L"
```

You should receive the output shown in Example 1-27.

```
Example 1-27 Querying Kafka cluster
admin@sc74cn09:~$ docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.76:19092 -L"
Metadata for all topics (from broker -1: 129.40.23.76:19092/bootstrap):
  2 brokers:
    broker 3 at 129.40.23.76:39092
    broker 1 at 129.40.23.76:19092
  0 topics:
```

To bring kafka-app-2 up again, issue the following command:

```
docker start kafka-app-2
```

Verify if kafka-app-2 re-joined the cluster by issuing the following command:

```
docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.76:19092 -L"
```

You should receive the output shown in Example 1-28.

```
Example 1-28 Querying the Kafka cluster
admin@sc74cn09:~$ docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.76:19092 -L"
Metadata for all topics (from broker -1: 129.40.23.76:19092/bootstrap):
  3 brokers:
    broker 2 at 129.40.23.76:29092
```

If you received output similar to that shown in Example 1-28, the Kafka cluster is fully operational.

### 1.6 Kafka configuration on three zCX instances

This section provides information on how to build a Kafka cluster to achieve high availability. The process is similar to that of deploying the containers in one zCX instance. Three zCX instances that will host another Kafka cluster having three ZooKeeper containers and three Kafka containers. This will illustrate a high availability solution.

Our example in this section uses the following container names: `zk-app-1`, `zk-app-2`, `zk-app-3` for ZooKeeper and `kafka-app-1`, `kafka-app-2` and `kafka-app-3` for Kafka. We appended the container instance number to help to identify where each instance will be running, as shown in Figure 1-3.

![Kafka cluster for high availability](image)

Table 1-5 provides an overview of our Kafka cluster information. Please note the column called Instance Location (zCX).
To easily identify which zCX host a container will be running on, the container names, persistent volumes and application ports all have the instance number appended to them.

In our example:
- Containers with instance name #1 will be deployed on sc74cn03 zCX server
- Containers with instance name #2 will be deployed on sc74cn04 zCX server
- Containers with instance name #3 will be deployed on sc74cn09 zCX server

Perform the following steps while connected to all zCX servers:

**Confirm containers do not exist**
If you have followed the steps to define a Kafka cluster previously, it is good to perform a cleanup before you define a new cluster. Issue:

```
docker rm --force zk-app-1
docker rm --force zk-app-2
docker rm --force zk-app-3
docker rm --force kafka-app-1
docker rm --force kafka-app-2
docker rm --force kafka-app-3
```

**Create ZooKeeper file**
No official ZooKeeper Docker image is available on the Docker hub at this time. The steps described in this section will build a ZooKeeper image to run on the IBM Z platform. For the latest Dockerfile examples for ZooKeeper, see:

https://github.com/linux-on-ibm-z/dockerfile-examples

1. Create a folder and download the two required files from github, Use the commands shown in Example 1-29 on your zCX instance.

```
Example 1-29  Create the ZooKeeper file

mkdir -p /home/admin/zookeeper
```
cd zookeeper

curl -k -o Dockerfile.zookeeper

curl -k -o docker-entrypoint.sh

chmod 755 docker-entrypoint.sh

Note: If you run into problems using the curl commands to download the required files, download these files from a Linux server and then upload to your zCX instance by using the `scp` or `ssh` commands. Use the following instructions:

a. Download the Dockerfile and put into /tmp/ and name it as `Dockerfile.zookeeper`
b. Download `docker-entrypoint.sh` and put into /tmp/
c. Issue the following commands to upload

```
cat /tmp/Dockerfile.zookeeper | ssh -p 8022 admin@<zcx IP address> "cat > /home/admin/Dockerfile.zookeeper"
cat /tmp/docker-entrypoint.sh | ssh -p 8022 admin@<zcx IP address> "cat > /home/admin/Docker-entrypoint.sh"
```

The Dockerfile.zookeeper file is written to: /home/admin/Dockerfile.zookeeper
Docker-entrypoint.sh file is written to: /home/admin/Docker-entrypoint.sh

2. Edit the `Dockerfile.zookeeper` file with your details as shown Example 1-30 and highlighted in red.

```
Example 1-30 Dockerfile.zookeeper file

FROM s390x/ubuntu:18.04

ENV JAVA_HOME=/usr/lib/jvm/java-11-openjdk-s390x

ENV ZOO_CONF_DIR=/conf 
  ZOO_DATA_DIR=/data 
  ZOO_DATA_LOG_DIR=/datalog 
  ZOO_LOG_DIR=/logs 
  ZOO_TICK_TIME=2000 
  ZOO_INIT_LIMIT=5 
  ZOO_SYNC_LIMIT=2 
  ZOO_AUTOPURGE_PURGEINTERVAL=0 
  ZOO_AUTOPURGE_SNAPRETAINECOUNT=3 
  ZOO_MAX_CLIENT_CNXNS=60 
  ZOO_STANDALONE_ENABLED=true 
  ZOO_ADMINSERVER_ENABLED=true
```
# Add a user with an explicit UID/GID and create necessary directories
RUN set -eux; \
    groupadd -r zookeeper --gid=1000; \
    useradd -r -g zookeeper --uid=1000 zookeeper; \
    mkdir -p "$ZOO_DATA_LOG_DIR" "$ZOO_DATA_DIR" "$ZOO_CONF_DIR" "$ZOO_LOG_DIR"; \
    chown zookeeper:zookeeper "$ZOO_DATA_LOG_DIR" "$ZOO_DATA_DIR" "$ZOO_CONF_DIR" "$ZOO_LOG_DIR"

# Install required packges
RUN set -eux; \
    apt-get update; \
    DEBIAN_FRONTEND=noninteractive \
    apt-get install -y --no-install-recommends \
        openjdk-11-jre \
        tar \
        wget \
        vim \
        ca-certificates \
        dirmngr \
        gosu \
        gnupg \
        netcat \
    rm -rf /var/lib/apt/lists/*; \
# Verify that gosu binary works
    gosu nobody true
ARG GPG_KEY=BBE7232D7991050B54C8EA0ADC08637CA615D22C
ARG SHORT_DISTRO_NAME=zookeeper-3.6.1
ARG DISTRO_NAME=apache-zookeeper-3.6.1-bin

# Download Apache Zookeeper, verify its PGP signature, untar and clean up
RUN set -eux; \
    ddist() { \n        local f="$1"; shift; \n        local distFile="$1"; shift; \n        local success=; \n        local distUrl=; \n        for distUrl in \n            'https://www.apache.org/dyn/closer.cgi?action=download&filename=' \n            https://www-us.apache.org/dist/ \n            https://www.apache.org/dist/ \n            https://archive.apache.org/dist/ \n        ; do \n            if wget -q -O "$f" "$distUrl$distFile" && [ -s "$f" ]; then \n                success=1; \n                break; \n            fi; \n        done; \n        [ -n "$success" ]; \n    }; \n    ddist "$DISTRO_NAME.tar.gz" 
"zookeeper/$SHORT_DISTRO_NAME/$DISTRO_NAME.tar.gz"; \
    ddist "$DISTRO_NAME.tar.gz.asc" 
"zookeeper/$SHORT_DISTRO_NAME/$DISTRO_NAME.tar.gz.asc"; \
    export GNUPGHOME="$(mktemp -d)"; \

gpg --keyserver ha.pool.sks-keyservers.net --recv-key "$GPG_KEY" || \ 
gpg --keyserver pgp.mit.edu --recv-keys "$GPG_KEY" || \ 
gpg --keyserver keyserver.pgp.com --recv-keys "$GPG_KEY"; \ 
gpg --batch --verify "$DISTRO_NAME.tar.gz.asc" "$DISTRO_NAME.tar.gz"; \ 
tar -zxf "$DISTRO_NAME.tar.gz"; \ 
rm -rf "$GNUPGHOME" "$DISTRO_NAME.tar.gz" "$DISTRO_NAME.tar.gz.asc"; \ 
chown -R zookeeper:zookeeper "/$DISTRO_NAME"

WORKDIR $DISTRO_NAME
VOLUME ["$ZOO_DATA_DIR", "$ZOO_DATA_LOG_DIR", "$ZOO_LOG_DIR"]

#EXPOSE 2181 2888 3888 8080

ENV PATH=$PATH:/$DISTRO_NAME/bin \ 
ZOOCFGDIR=$ZOO_CONF_DIR

COPY docker-entrypoint.sh /
ENTRYPOINT ["/docker-entrypoint.sh"]
CMD ["zkServer.sh", "start-foreground"]

Note: Writing EXPOSE in the Dockerfile is merely a hint that a certain port is useful. Docker won’t do anything with that information by itself.

3. Issue the following command to build the ZooKeeper image.

    docker build -t zcx-zookeeper-img -f Dockerfile.zookeeper .

    We named the new image zcx-zookeeper-img.

4. Create two persistent volumes for each ZooKeeper instance by using the following commands:

    For zk-app-1, issue commands below connected to sc74cn03 server:

    docker volume create zool-conf
    docker volume create zool-data

    For zk-app-2, issue commands below connected to sc74cn04 server:

    docker volume create zoo2-conf
    docker volume create zoo2-data

    For zk-app-3, issue commands below connected to sc74cn09 server:

    docker volume create zoo3-conf
    docker volume create zoo3-data

Use the following steps to populate the volumes for each container instance. We use a dummy temporary image to help with it.

1. For zk-app-1, execute the commands shown in Example 1-31 on sc74cn03 server:

    Example 1-31 Populate the volumes for zk-app-1

    docker run --name dummy -v zool-conf:/conf1 -v zool-data:/data1 -d 
    zcx-zookeeper-img

    docker exec -it dummy bash
cp -a /conf/* /conf1/

cp -a /data/* /data1/

apt-get update

apt install -y vim

Edit /conf1/zoo.cfg file by using the vi command with the following instructions.

a. Delete the following line:
   server.1=localhost:2888:3888:2181

b. Update the following values:
   initLimit=10
   syncLimit=5
   standaloneEnabled=false

c. Add the following lines:
   server.1=0.0.0.0:12888:13888:12181
   server.2=129.40.23.71:22888:23888:22181
   server.3=129.40.23.76:32888:33888:32181

After the updates, the zoo.cfg file should look similar to that shown in Example 1-32:

```
Example 1-32  /conf1/zoo.cfg file

dataDir=/data
dataLogDir=/datalog
tickTime=2000
initLimit=10
syncLimit=5
autopurge.snapRetainCount=3
autopurge.purgeInterval=0
maxClientCnxns=60
standaloneEnabled=false
admin.enableServer=true
server.1=0.0.0.0:12888:13888:12181
server.2=129.40.23.71:22888:23888:22181
server.3=129.40.23.76:32888:33888:32181
```

d. Edit the /data1/myid file by using the vi command and update with the number 1 for our first ZooKeeper instance.

e. Execute the exit command to leave the dummy container

f. Execute the command docker rm dummy --force to delete the dummy container.

2. For zk-app-2, execute the commands shown in Example 1-33 on the sc74cn04 server:

```
Example 1-33  Populate the volumes for zk-app-2

docker run --name dummy -v zoo2-conf:/conf2 -v zoo2-data:/data2 -d
  zcx-zookeeper-img

docker exec -it dummy bash

cp -a /conf/* /conf2/

cp -a /data/* /data2/
```
apt-get update

apt install -y vim

Edit the `/conf2/zoo.cfg` file by using the `vi` command with the following instructions:

a. Delete the following line:
   ```
   server.1=localhost:2888:3888;2181
   ```

b. Update the following values:
   ```
   initLimit=10
   syncLimit=5
   standaloneEnabled=false
   ```

c. Add the following lines:
   ```
   server.1=129.40.23.70:12888:13888;12181
   server.2=0.0.0.0:22888:23888;22181
   server.3=129.40.23.76:32888:33888;32181
   ```

After the updates, the zoo.cfg file should look similar to that shown in Example 1-34:

```
Example 1-34 /conf2/zoo.cfg file

dataDir=/data
dataLogDir=/datalog
tickTime=2000
initLimit=10
syncLimit=5
autopurge.snapRetainCount=3
autopurge.purgeInterval=0
maxClientCnxns=60
standaloneEnabled=false
admin.enableServer=true
server.1=129.40.23.70:12888:13888;12181
server.2=0.0.0.0:22888:23888;22181
server.3=129.40.23.76:32888:33888;32181
```

d. Edit `/data2/myid` file by using the `vi` command and update with the number 2 for our second ZooKeeper instance.

e. Execute the `exit` command to leave the dummy container.

f. Execute the command `docker rm dummy --force` to delete the dummy container.

3. For zk-app-3, execute the commands shown in Example 1-35 on the `sc74cn09` server.

```
Example 1-35 Populate the volumes for zk-app-3

docker run --name dummy -v zoo3-conf:/conf3 -v zoo3-data:/data3 -d zcx-zookeeper-img

docker exec -it dummy bash

cp -a /conf/* /conf3/

cp -a /data/* /data3/
```

apt-get update
apt install -y vim

   i. Delete the following line:
      server.1=localhost:2888:3888;2181 line
   ii. Update the following values:
      initLimit=10
      syncLimit=5
      standaloneEnabled=false
   iii. Add the following lines:
      server.1=129.40.23.70:12888:13888;12181
      server.2=129.40.23.71:22888:23888;22181
      server.3=0.0.0.0:32888:33888;32181

   After the updates, the zoo.cfg file should look similar to that shown in Example 1-8.

Example 1-36  /conf3/zoo.cfg file

| dataDir=/data |
| dataLogDir=/datalog |
| tickTime=2000 |
| initLimit=10 |
| syncLimit=5 |
| autopurge.snapRetainCount=3 |
| autopurge.purgeInterval=0 |
| maxClientCnxns=60 |
| standaloneEnabled=false |
| admin.enableServer=true |
| server.1=129.40.23.70:12888:13888;12181 |
| server.2=129.40.23.71:22888:23888;22181 |
| server.3=0.0.0.0:32888:33888;32181 |

b. Edit the /data3/myid file using the vi command and update with the number 3 for our third ZooKeeper instance.

c. Execute the exit command to leave the dummy container

d. Execute the command docker rm dummy --force to delete the dummy container.

4. Start your three ZooKeeper instances. Since Zookeeper "fails fast" it's better to always restart it. We appended --restart always in the docker run commands shown in Example 1-37.

Example 1-37  Docker run commands

# On sc74cn03 server:

# On sc74cn04 server:

# On sc74cn09 server:
docker run --name zk-app-3 -p 32181:32181 -p 32888:32888 -p 33888:33888 -v zoo3-conf:/conf:rw -v zoo3-data:/data:rw --restart always -d zcx-zookeeper-img

5. Issue the following commands to verify the startup of each container:

   For the sc74cn03 server:
   
   ```bash
   docker logs zk-app-1
   ```

   For the sc74cn04 server:
   
   ```bash
   docker logs zk-app-2
   ```

   For the sc74cn09 server:
   
   ```bash
   docker logs zk-app-3
   ```

   For each command, verify that you receive a message such as that shown in Example 1-38.

   **Example 1-38 Example messages of successful start of each container**

   ```
   *** OUTPUT OMITTED ***
   2020-06-09 16:41:30,773 [myid:1] - INFO [main:ContextHandler@825] - Started
   o.e.j.s.ServletContextHandler@33ecda92{/,null,AVAILABLE}
   2020-06-09 16:41:30,778 [myid:1] - INFO [main:AbstractConnector@330] - Started
   ServerConnector@4c60d6e9{HTTP/1.1,[http/1.1]}{0.0.0.0:8080}
   2020-06-09 16:41:30,778 [myid:1] - INFO [main:Server@399] - Started @975ms
   2020-06-09 16:41:30,778 [myid:1] - INFO [main:JettyAdminServer@178] - Started
   AdminServer on address 0.0.0.0, port 8080 and command URL /commands
   *** OUTPUT OMITTED ***
   ```

6. For zk-app-1 instance on the sc74cn03 server, execute the following commands to confirm ZooKeeper is operational.

   ```bash
   docker exec -it zk-app-1 bash -c "wget -O - http://localhost:8080/commands/stats"
   ```

   You should receive output similar to that shown in Example 1-39. Highlighted in red is the state of each instance. One instance should be a leader and the others followers.

   **Example 1-39 Confirmation of follower instance**

   ```
   admin@sc74cn03:$ docker exec -it zk-app-1 bash -c "wget -O - http://localhost:8080/commands/stats"
   --2020-08-18 18:04:55--  http://localhost:8080/commands/stats
   Resolving localhost (localhost)... 127.0.0.1, ::1
   Connecting to localhost (localhost)|127.0.0.1|:8080... connected.
   HTTP request sent, awaiting response... 200 OK
   Length: 904 [application/json]
   Saving to: 'STDOUT'
   - 0 --.--KB/s  0%
   [ ]
   "version": "3.6.1--104dcb3e3fb464b30c5186d229e00af9f332524b, built on
   04/21/2020 15:01 GMT",
   "read_only": false,
   "server_stats": {
       "packets_sent": 0,
       "packets_received": 0,
       "fsync_threshold_exceed_count": 0,
       "client_response_stats": {
   ```
7. For zk-app-2 instance on the `sc74cn04` server, execute the following commands to confirm ZooKeeper is operational.

```
docker exec -it zk-app-2 bash -c "wget -O - http://localhost:8080/commands/stats"
```

You should receive similar output as shown in Example 1-40. Highlighted in red is the state of each instance. One instance should be a leader and the others followers.

```
admin@sc74cn04:$ docker exec -it zk-app-2 bash -c "wget -O - http://localhost:8080/commands/stats"
--2020-08-18 18:06:28--  http://localhost:8080/commands/stats
Resolving localhost (localhost)... 127.0.0.1, ::1
Connecting to localhost (localhost)|127.0.0.1|:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1023 [application/json]
Saving to: 'STDOUT'

100%[..................................................................................................] 904 --.-KB/s
in 0s

2020-08-18 18:04:56 (484 MB/s) - written to stdout [904/904]
```

Example 1-40  Confirmation of leader instance
"server_stats": {
    "packets_sent": 0,
    "packets_received": 0,
    "fsync_threshold_exceed_count": 0,
    "client_response_stats": {
        "last_buffer_size": -1,
        "min_buffer_size": -1,
        "max_buffer_size": -1
    },
    "uptime": 521767,
    "provider_null": false,
    "server_state": "leader",
    "outstanding_requests": 0,
    "min_latency": 0,
    "avg_latency": 0.0,
    "max_latency": 0,
    "data_dir_size": 0,
    "log_dir_size": 459,
    "last_processedzxid": 4294967296,
    "num_alive_client_connections": 0
},
"client_response": {
    "last_buffer_size": -1,
    "min_buffer_size": -1,
    "max_buffer_size": -1
},
"proposal_stats": {
    "last_buffer_size": -1,
    "min_buffer_size": -1,
    "max_buffer_size": -1
},
"node_count": 5,
"connections": [],
"secure_connections": [],
"command": "stats",
"error": null

8. For zk-app-3 instance on sc74cn09 server, execute the following commands to confirm ZooKeeper is operational.

docker exec -it zk-app-3 bash -c "wget -O - http://localhost:8080/commands/stats"

You should receive output shown in Example 1-41. Highlighted in red is the state of each instance. One instance should be a leader and the others followers.

Example 1-41 Confirmation of second follower instance

admin@sc74cn09:~/zookeeper$ docker exec -it zk-app-3 bash -c "wget -O - http://localhost:8080/commands/stats"
--2020-08-18 18:10:27-- http://localhost:8080/commands/stats

-----------[======================================================>] 1023 --.-KB/s
in 0s

2020-08-18 18:06:28 (481 MB/s) - written to stdout [1023/1023]
Resolving localhost (localhost)... 127.0.0.1, ::1
Connecting to localhost (localhost)|127.0.0.1|:8080... connected.
HTTP request sent, awaiting response... 200 OK
Length: 914 [application/json]
Saving to: 'STDOUT'

```
- 0 --.-KB/s
   "version" : "3.6.1--104dcb3e3fb464b30c5186d229e00af9f332524b, built on 04/21/2020 15:01 GMT",
   "read_only" : false,
   "server_stats" : {
     "packets_sent" : 0,
     "packets_received" : 0,
     "fsync_threshold_exceed_count" : 0,
     "client_response_stats" : {
       "last_buffer_size" : -1,
       "min_buffer_size" : -1,
       "max_buffer_size" : -1
     },
     "provider_null" : false,
     "uptime" : 603836,
     "server_state" : "follower",
     "outstanding_requests" : 0,
     "min_latency" : 0,
     "avg_latency" : 0.0,
     "max_latency" : 0,
     "data_dir_size" : 0,
     "log_dir_size" : 1100,
     "last_processed_zxid" : 4294967296,
     "num_alive_client_connections" : 0
   },
   "client_response" : {
     "last_buffer_size" : -1,
     "min_buffer_size" : -1,
     "max_buffer_size" : -1
   },
   "node_count" : 5,
   "connections" : [ ],
   "secure_connections" : [ ],
   "command" : "stats",
   "error" : null
- 100%[===============================================>] 914 --.-KB/s
in 0s
```

2020-08-18 18:10:27 (383 MB/s) - written to stdout [914/914]

Zookeeper installation is completed. In the next section, “Kafka Docker images”, you will start the installation of Kafka in a high availability (HA) use case.

### Kafka Docker images

In this section, we provide the steps to create images in order to install Kafka.

Complete these steps on sc74cn03, sc74cn04 and sc74cn09.
1. Use the following commands to create a folder for the Kafka image.
   
   ```bash
   mkdir /home/admin/kafka
cd /home/admin/kafka
   ```

2. Download the Kafka Dockerfile using the following command:
   
   ```bash
   curl -k -o Dockerfile.kafka
   https://raw.githubusercontent.com/linux-on-ibm-z/dockerfile-examples/master/ApacheKafka/Dockerfile
   ```

   **Note:** You can check latest Dockerfile example for Kafka in https://github.com/linux-on-ibm-z/dockerfile-examples

   ```bash
   ```

3. Edit the file `Dockerfile.kafka` to look similar to that shown in Example 1-42. Highlighted in red is what is required to update, add or remove.

   ```bash
   # LICENSE: Apache License, Version 2.0
   (http://www.apache.org/licenses/LICENSE-2.0)

   ###### dockerfile for Apache Kafka version 2.5.0 ######
   
   # This Dockerfile builds a basic installation of Apache Kafka.
   # Kafka is run as a cluster on one or more servers. The Kafka cluster stores
   # streams of records in categories called topics.
   # Each record consists of a key, a value, and a timestamp.
   # In Kafka the communication between the clients and the servers is done with a
   # simple, high-performance, language agnostic TCP protocol.
   # To build this image, from the directory containing this Dockerfile
   # (assuming that the file is named Dockerfile):
   # docker build -t <image_name> .
   # To Start Apache Kafka run the below command:
   # docker run --name <container_name> -d <image>
   # To check Apache kafka is running, Enter below command:
   # docker exec <container_id of kafka> <any kafka related command>
   # Eg. To list topic and message files:
   ```
# docker exec <container_id of kafka> bin/kafka-topics.sh --list --zookeeper localhost:2181
#
# Reference:
# http://kafka.apache.org/
# https://kafka.apache.org/quickstart
#
# Base Image
FROM s390x/ubuntu:20.04

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

ENV SOURCE_DIR=/home/
ENV JAVA_HOME=/home/jdk-11.0.5+10
ENV PATH=$PATH:$SOURCE_DIR/:$JAVA_HOME/bin
ENV VERSION=2.12-2.5.0
WORKDIR $SOURCE_DIR

# Install dependencies
RUN apt-get update && apt-get -y install \
    git \
    unzip \
    wget \
    vim \
    net-tools \
    kafkacat \
# Download Adopt JDK
    && wget https://github.com/AdoptOpenJDK/openjdk11-binaries/releases/download/jdk-11.0.5 \%
    2B10/OpenJDK11U-jdk_s390x_linux_hotspot_11.0.5_10.tar.gz \
    && tar -xvzf OpenJDK11U-jdk_s390x_linux_hotspot_11.0.5_10.tar.gz \
    && rm OpenJDK11U-jdk_s390x_linux_hotspot_11.0.5_10.tar.gz \
# Download the Apache Kafka Binary
    && wget http://mirrors.estointernet.in/apache/kafka/2.5.0/kafka_${VERSION}.tgz \
    && tar -xvzf kafka_${VERSION}.tgz \
    && rm kafka_${VERSION}.tgz \
    && mv kafka_${VERSION} kafka
# Expose ports for Apache ZooKeeper and kafka
#EXPOSE 2181 9092
WORKDIR $SOURCE_DIR/kafka/

# start zookeeper and kafka server
CMD bin/zookeeper-server-start.sh -daemon config/zookeeper.properties && sleep 20 && bin/kafka-server-start.sh config/server.properties > /dev/null

# End of Dockerfile
4. To build a new Kafka image and tag it as zcx-kafka-img, issue the following command:

   ```bash
docker build -t zcx-kafka-img -f Dockerfile.kafka .
   ```

You will receive the following messages if build completes successfully.

   ```
   Successfully built 3ac1c714ee08
   Successfully tagged zcx-kafka-img:latest
   ```

**Note:** Writing EXPOSE in Dockerfile, is merely a hint that a certain port is useful. Docker won't do anything with that information by itself.

5. Define kafka volume groups for each Kafka container instance using the following commands:

   On sc74cn03:
   ```bash
docker volume create kafka1-conf
   docker volume create kafka1-logs
   ```

   On c74cn04:
   ```bash
docker volume create kafka2-conf
   docker volume create kafka2-logs
   ```

   On sc74cn09:
   ```bash
docker volume create kafka3-conf
   docker volume create kafka3-logs
   ```

Doing this will make the configuration files persistent for Kafka, allowing your system to survive during an upgrade. For additional information about persistent volumes, please see the IBM Redbooks publication, *Getting started with z/OS Container Extensions and Docker*, SG24-8457.

Use the following steps to create the configuration file for each Kafka container image.

1. For kafka-app-1, execute the commands shown in Example 1-43 on sc74cn03:

   ```bash
   Example 1-43   Create the configuration file for kafka-app1 on sc74cn03
   docker run --name dummy -it --entrypoint /bin/bash -v
   kafka1-conf:/home/kafka/config1 -v kafka1-logs:/home/kafka/logs1 -d
   zcx-kafka-img
   docker exec -it dummy bash
   cp -a /home/kafka/config/* /home/kafka/config1/
   cp -a /home/kafka/logs/* /home/kafka/logs1/
   apt-get update
   apt install -y vim
   ```

   a. Edit `/home/kafka/config1/server.properties` file using the `vi` command by using instructions listed here.
i. Update the following values:

```
broker.id=1
zookeeper.connect=129.40.23.70:12181,129.40.23.71:22181,129.40.23.76:32181
```

ii. Add these lines (under Socket Server Settings)

```
listeners=PLAINTEXT://0.0.0.0:19092
advertised.listeners=PLAINTEXT://129.40.23.70:19092
```

After the updates, `/home/kafka/config1/server.properties` file should look like those shown in Example 1-16

```
Example 1-44   /home/kafka/config1/server.properties file

###############################################################################
# Server Basics
###############################################################################

# The id of the broker. This must be set to a unique integer for each broker.
broker.id=1

###############################################################################
# Socket Server Settings
###############################################################################

# The address the socket server listens on. It will get the value returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://0.0.0.0:19092
advertised.listeners=PLAINTEXT://129.40.23.70:19092

###############################################################################
# Zookeeper
###############################################################################

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. E.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify the
# root directory for all kafka znodes.
zookeeper.connect=129.40.23.70:12181,129.40.23.71:22181,129.40.23.76:32181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=18000
```

Note:
129.40.23.70 is the IP address of sc74cn03
129.40.23.71 is the IP address of sc74cn04
129.40.23.76 is the IP address of sc74cn09
iii. Issue `exit` command to leave the dummy container

iv. Issue `docker rm dummy --force` command to delete the dummy container.

2. For kafka-app-2, execute the commands shown in Example 1-45 on sc74cn04:

Example 1-45  Create the configuration file for kafka-app-2 on sc74cn04

```
docker run --name dummy -it --entrypoint /bin/bash -v kafka2-conf:/home/kafka/config2 -v kafka2-logs:/home/kafka/logs2 -d zcx-kafka-img

docker exec -it dummy bash

cp -a /home/kafka/config/* /home/kafka/config2/

cp -a /home/kafka/logs/* /home/kafka/logs2/

apt-get update

apt install -y vim
```

a. Edit `/home/kafka/config2/server.properties` file using the `vi` command with instructions below.

Note:

129.40.23.70 is the IP address of sc74cn03

129.40.23.71 is the IP address of sc74cn04

129.40.23.76 is the IP address of sc74cn09

i. Update the following values:

   broker.id=2

   zookeeper.connect=129.40.23.70:12181,129.40.23.71:22181,129.40.23.76:32181

ii. Add these lines (under Socket Server Settings)

   listeners=PLAINTEXT://0.0.0.0:29092

   advertised.listeners=PLAINTEXT://129.40.23.71:29092

After the updates, `/home/kafka/config2/server.properties` file should look like those shown in Example 1-46.

Example 1-46  `/home/kafka/config2/server.properties` file

```
############################# Server Basics ############################

# The id of the broker. This must be set to a unique integer for each broker.
broker.id=2

############################## Socket Server Settings ######################

# The address the socket server listens on. It will get the value returned from
```
# java.net.InetAddress.getCanonicalHostName() if not configured.
# FORMAT:
# listeners = listener_name://host_name:port
# EXAMPLE:
# listeners = PLAINTEXT://your.host.name:9092
#listeners=PLAINTEXT://:9092
listeners=PLAINTEXT://0.0.0.0:29092
advertised.listeners=PLAINTEXT://129.40.23.71:29092


*** OUTPUT OMITTED ***

### Zookeeper ###

# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify
# root directory for all kafka znodes.
zookeeper.connect=129.40.23.70:12181,129.40.23.71:22181,129.40.23.76:32181

# Timeout in ms for connecting to zookeeper
zookeeper.connection.timeout.ms=18000


*** OUTPUT OMITTED ***

iii. Issue the **exit** command to leave the dummy container

iv. Issue the **docker rm dummy --force** command to delete the dummy container.

3. For kafka-app-3, execute the commands shown in Example 1-47 on sc74cn09:

   Example 1-47 Create the configuration file for kafka-app-3 on sc74cn09
   
docker run --name dummy -it --entrypoint /bin/bash -v
   kafka3-conf:/home/kafka/config3 -v kafka3-logs:/home/kafka/logs3 -d
   zcx-kafka-img

docker exec -it dummy bash

cp -a /home/kafka/config/* /home/kafka/config3/

cp -a /home/kafka/logs/* /home/kafka/logs3/

apt-get update

apt install -y vim

a. Edit the /home/kafka/config3/server.properties file using the **vi** command with the
following instructions.
i. Update the following values:

   broker.id=3
   zookeeper.connect=129.40.23.70:12181,129.40.23.71:22181,129.40.23.76:32181

ii. Add the following lines (under Socket Server Settings)

   listeners=PLAINTEXT://0.0.0.0:39092
   advertised.listeners=PLAINTEXT://129.40.23.76:39092

After the updates, /home/kafka/config3/server.properties file should look like that shown in Example 1-48.

Example 1-48 /home/kafka/config3/server.properties

# The id of the broker. This must be set to a unique integer for each broker.
broker.id=3

# The address the socket server listens on. It will get the value returned from java.net.InetAddress.getCanonicalHostName() if not configured.
listeners=PLAINTEXT://0.0.0.0:9092
advertised.listeners=PLAINTEXT://129.40.23.76:9092

---

Note:

129.40.23.70 is the IP address of sc74cn03
129.40.23.71 is the IP address of sc74cn04
129.40.23.76 is the IP address of sc74cn09
iii. Issue the exit command to leave the dummy container
iv. Issue the docker rm dummy --force command to delete the dummy container.

4. Now, bring your kafka instances up by issuing the following commands:
   On sc74cn03:
   ```
   docker run --name kafka-app-1 -p 19092:19092 -v kafka1-conf:/home/kafka/config
   -v kafka1-logs:/home/kafka/logs -d zcx-kafka-img
   ```
   On sc74cn04:
   ```
   docker run --name kafka-app-2 -p 29092:29092 -v kafka2-conf:/home/kafka/config
   -v kafka2-logs:/home/kafka/logs -d zcx-kafka-img
   ```
   On sc74cn09:
   ```
   docker run --name kafka-app-3 -p 39092:39092 -v kafka3-conf:/home/kafka/config
   -v kafka3-logs:/home/kafka/logs -d zcx-kafka-img
   ```

   **Hint:** The Dockerfile.kafka file was updated to install the kafkacat utility that you can use to test and debug Apache Kafka.

5. Check the Kafka container logs to see if kafka was successfully started by using the following command:
   On sc74cn03:
   ```
   docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.70:19092 -L"
   ```
   On sc74cn04:
   ```
   docker exec -it kafka-app-2 bash -c "kafkacat -b 129.40.23.71:29092 -L"
   ```
   On sc74cn09:
   ```
   docker exec -it kafka-app-3 bash -c "kafkacat -b 129.40.23.76:39092 -L"
   ```

   Output should be:
   ```
   Metadata for all topics (from broker 1: 129.40.23.70:19092/1):
   3 brokers:
   broker 2 at 129.40.23.71:29092
   broker 3 at 129.40.23.76:39092
   broker 1 at 129.40.23.70:19092 (controller)
   0 topics:
   ```

   Use the following steps to optionally perform a high availability test to ensure everything is fine.

1. To do this, first check if the cluster is healthy. Check the first Kafka instance by using the following command on sc74cn03:
   ```
   docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.70:19092 -L"
   ```
   You should receive the output shown in Example 1-49.

   **Example 1-49  Kafka-app-1 check**
   ```
   Metadata for all topics (from broker 1: 129.40.23.70:19092/1):
   3 brokers:
   broker 2 at 129.40.23.71:29092
   broker 3 at 129.40.23.76:39092
   broker 1 at 129.40.23.70:19092 (controller)
   ```
To verify that the cluster can detect the state of the Kafka instances, take one of the instances down. To take kafka-app-2 down, issue the following command on sc74cn04:

`docker stop kafka-app-2`

To check if the cluster detected a problem on kafka-app-2 and removed this broker from the cluster, re-issue the `kafkacat` command on kafka-app-1 to check the cluster. Use the following command on sc74cn03:

`docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.70:19092 -L"`

You should receive the output shown in Example 1-27.

**Example 1-50  Querying Kafka cluster**

```bash
admin@sc74cn03:~/kafka$ docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.70:19092 -L"
Metadata for all topics (from broker 1: 129.40.23.70:19092/1):
  2 brokers:
    broker 3 at 129.40.23.76:39092
    broker 1 at 129.40.23.70:19092 (controller)
  0 topics:
```

To bring kafka-app-2 up again, issue the following command on sc74cn04:

`docker start kafka-app-2`

Verify if kafka-app-2 re-joined the cluster by issuing the following command on sc74cn03:

`docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.70:19092 -L"`

You should receive the output shown in Example 1-51.

**Example 1-51  Querying the Kafka cluster**

```bash
admin@sc74cn03:~/kafka$ docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.70:19092 -L"
Metadata for all topics (from broker 1: 129.40.23.70:19092/1):
  3 brokers:
    broker 2 at 129.40.23.71:29092
    broker 3 at 129.40.23.76:39092
    broker 1 at 129.40.23.70:19092 (controller)
  0 topics:
```

If you received the output similar to that shown in Example 1-51, the Kafka cluster is fully operational.

### 1.7 Diagnostic commands

You may come across issues that need to be diagnosed and corrected. This section is a basic list of commands for ZooKeeper and Kafka that will assist you in diagnosing and correcting these issues.
1.7.1 ZooKeeper

These commands must be run inside one of the ZooKeeper instances (in our example, zk-app-1, 2 or 3). Issue the command `docker exec -it zk-app-1 bash` to connect to the zk-app-1 container and then issue the commands shown in this section.

- To list active brokers:
  ```sh
  /apache-zookeeper-3.6.1-bin/bin/zkCli.sh -server 129.40.23.76:12181 ls /brokers/ids
  ```

- To get detailed information of a specific broker id “1”: (Broker numbers can be found with the command above)
  ```sh
  /apache-zookeeper-3.6.1-bin/bin/zkCli.sh -server 129.40.23.76:12181 get /brokers/ids/1
  ```

Repeat each command for each of the brokers (broker ID 2 and broker ID 3).

- To list topics:
  ```sh
  /apache-zookeeper-3.6.1-bin/bin/zkCli.sh -server 129.40.23.76:12181 ls /brokers/topics
  ```

Another very helpful command is `docker logs` that provides information logged by a running container.

Note: Replace IP address and port (i.e. 129.40.23.76:12181) with information of any other Zookeeper instance. (129.40.23.76:12181, 129.40.23.76:22181 or 129.40.23.76:32181)

1.8 Integrating IBM Z applications and Kafka

In this section, we describe different types of applications and their integration with Kafka.

1.8.1 Event driven application and Apache Kafka

Competition and disrupters are causing companies to become more customer-centric. This has led to a surge in event driven solutions and applications. Event driven solutions require different thinking than data-centric solutions. Events form the nervous system of the digital business.

Application infrastructure needs to provide event stream processing capabilities and support emerging event-driven programming models. This is an event-driven journey and will underpin the next generation of digital customer experiences.

Let us look at the components of an event driven application as shown in Figure 1-4 on page 61.
The four components of an event driven application are

1. **Event Sources**: They are application triggers, data in existing databases, sensors, web traffic etc. Event sources can also be transactions on the mainframe that we will discuss in our use case later in this chapter. These event sources publish event data to the event backbone.

2. **Stream Processing**: They are applications that process streams of data that are in the event backbone in real time.

3. **Event Archive**: They are applications or tools that store the events from the event backbone for future reference.

4. **Notifications**: These are applications that trigger notifications based on some events that are stored in the event backbone.

All of the above four components are connected by an event backbone. Apache Kafka is a perfect match for this event backbone.

### 1.8.2 Key usage patterns of Apache Kafka and IBM Z

Companies are facing growing competition and want to modernize their customer experience to improve customer retention. Companies run their mission critical business applications and store their data on the mainframe. Companies want to build and offer services to customers who can be alerted in real-time when “noteworthy” events occur. Those events vary by applications and industries. Customers do not want to disrupt or add load to their core systems which handle millions of transactions a day and are costly and complex to change. To stay competitive, companies want to unlock the data flowing through its transactional systems. Let us look at one of the key usage pattern of unlocking events from the mainframe using Apache Kafka.

One of the most common usage patterns for Kafka with IBM Z is shown in Figure 1-5 on page 62. Multiple teams responsible for innovation, application and microservice are demanding data from the systems of record, often hosted on an IBM Z. The teams running the systems of record need to ensure that they are stable, but this new demand is unpredictable, and often not well defined at the outset. Kafka offers a way to expose the data so that it can be consumed by multiple readers at their own pace. The load on the backend systems is well defined and predicatable.
Kafka offers the data as a persistent read-only buffer or cache of the data which is continually refreshed and updated.

We can progress that scenario to the next level by pushing that data out to multiple consumers on multiple clouds. Allowing each set of consumers on each cloud to have its own buffer or stream of the data allows them to have a consistent, rapid read of the data, while minimizing the amount of data that is transferred.

Figure 1-5  Key Usage Pattern

1.8.3 Unlocking data from IBM Z to Kafka

Companies run their mission critical core transactions on the mainframe with CICS, IMS and DB2. We can unlock the data on those transactional systems and databases using one of the various options as shown in Figure 1-6 on page 63
The following products have the ability to integrate with Apache Kafka:

- IBM MQ: Use of the IBM MQ Kafka Connectors
- IBM App Connect Enterprise: Use of the Kafka nodes
- IBM z/OS Connect EE: Outbound REST API call to Kafka
- CICS: Using Liberty Kafka client running in CICS
- IBM InfoSphere® Data Replication - Change Data Capture.

1.9 Integration of IBM MQ with Apache Kafka

Many organizations use both IBM MQ and Apache Kafka for their messaging needs. Although they’re generally used to solve different kinds of messaging problems, users often want to connect them together for various reasons. For example, IBM MQ can be integrated with systems of record while Apache Kafka is commonly used for streaming events from web applications. The ability to connect the two systems together enables scenarios in which these two environments intersect. It is easy to connect from IBM MQ to Apache Kafka. IBM has created a pair of connectors, available as source code or as part of IBM Event Streams. The two connectors are:

- Kafka Connect source connector for IBM MQ
  - Kafka-connect-mq-source is a Kafka Connect source connector for copying data from IBM MQ into Apache Kafka. The following link provides more information on the MQ source connector: https://github.com/ibm-messaging/kafka-connect-mq-source

- Kafka Connect sink connector for IBM MQ
  - You can use the MQ sink connector to copy data from Apache Kafka into IBM MQ. The connector copies messages from a Kafka topic into a MQ queue. The following link provides more information on the MQ source connector: https://github.com/ibm-messaging/kafka-connect-mq-sink
In this section we will discuss the very popular MQ Connectors for integration with Apache Kafka.

1.9.1 Setting up MQ connectors to run on IBM z/OS

In this section, we will discuss how to build and run the MQ connectors. The connectors can be deployed and run in a Kafka Connect runtime, for example, in z/OS Unix System Services. In this section we will build and configure the connectors running on z/OS Unix System Services as shown in Figure 1-7.

![Figure 1-7 Running connectors on Unix System Services](image)

The following are the steps to set up the connectors on IBM z/OS Unix System Services

1. Setting up Apache Kafka to run on IBM z/OS
   - Download Apache Kafka 2.0.0 or later to a non z/OS system from the following link: https://kafka.apache.org/downloads
   - On the system that you downloaded the file, extract the downloaded .tgz file to a .tar using gunzip as shown in Example 1-52
     
     **Example 1-52 Extract to tar file**
     ```
     gunzip -k kafka_2.12-2.5.0.tgz
     ```

   - Transfer the resulting kafka_2.12-2.5.0.tar file to the directory /u/maitra/kafkadownl/ on z/OS Unix System Services as shown in Example 1-53.
     
     **Example 1-53 Transfer the tar file**
     ```
     MAITRA@SC59:/u/maitra/kafkadownl> ls kafka*
     kafka_2.12-2.5.0.tar
     ```

2. Extract the Apache Kafka distribution.
   - Create a new directory and navigate to the new directory as shown in Example 1-54.
     
     **Example 1-54 Create directory for kafka**
     ```
     MAITRA@SC59:/u/maitra>
     ```
===> mkdir kafka
MAITRA@SC59:/u/maitra> cd kafka
MAITRA@SC59:/u/maitra/kafka>

– Copy the `kafka_2.12-2.5.0.tar` file from `/u/maitra/kafka/download/` to the new directory
– Extract the `kafka_2.12-2.5.0.tar` file, for as shown in Example 1-55.

```
Example 1-55  Extract tar file
tar -xvf kafka_2.12-2.5.0.tar
```

– Navigate to the newly created directory as shown in Example 1-56.

```
Example 1-56
MAITRA@SC59:/u/maitra/kafka>
===> cd kafka_2.12-2.5.0
```

3. Convert the shell scripts to run in z/OS Unix System Services.
– We are going to run the connectors in distributed mode and will do a distributed setup.
Copy the `connect-distributed.sh` shell script into the current directory as shown in Example 1-57.

```
Example 1-57  Copy connect-distributed.sh
MAITRA@SC59:/u/maitra/kafka/kafka_2.12-2.5.0>
===> cp bin/connect-distributed.sh ./connect-distributed.sh.orig
```

– Determine the codeset on the IBM z/OS system by running the following locale command as shown in Example 1-58.

```
Example 1-58
MAITRA@SC59:/u/maitra/kafka/kafka_2.12-2.5.0>
locale -k codeset
codeset="IBM-1047"
```

– Convert the script to EBCDIC encoding and replace the original, for codeset IBM-1047 using the following command as shown in Example 1-59.

```
Example 1-59
iconv -f ISO8859-1 -t IBM-1047 ./connect-distributed.sh.orig > bin/connect-distributed.sh
```

– Ensure the file permissions are set so that the script is executable by running the following command as shown in Example 1-60.

```
Example 1-60  Change file permission for distributed.sh
chmod +x bin/connect-distributed.sh
```
– We are now going to convert the kafka-run-class.sh shell script. Copy the kafka-run-class.sh shell script into the current directory as shown in Example 1-61.

Example 1-61  Copy the shell script

```bash
    cp bin/kafka-run-class.sh ./kafka-run-class.sh.orig
```

– Convert the script to EBCDIC encoding and replace the original, for codeset IBM-1047 using the following command as shown in Example 1-62.

Example 1-62  Convert kafka-run-class.sh

```bash
    iconv -f ISO8859-1 -t IBM-1047 ./kafka-run-class.sh.orig > bin/kafka-run-class.sh
```

– Ensure the file permissions are set so that the script is executable by running the following command as shown in Example 1-63.

Example 1-63  Change permissions

```bash
    chmod +x bin/kafka-run-class.sh
```

4. Download IBM MQ Source connectors and configuration files.

– In our use case, we will download and configure the IBM MQ Source Connector. If you are licensed to run IBM Event Streams, you can download the connector from the IBM Event Streams UI using the following steps:

• Log in to the IBM Event Streams UI from a supported web browser

• Click Toolbox in the primary navigation as shown in Figure 1-8 on page 67.
Scroll to the Connectors section and click on “Connecting to IBM MQ” as shown in Figure 1-9

Ensure the MQ Source tab is selected and download the connector file and configuration file as shown in Figure 1-10 on page 68. Two files are downloaded to the local system and need to be transferred to z/OS Unix System Services:

i. kafka-connect-mq-source-1.1.1-jar-with-dependencies.jar
ii. mq-source.json
Edit the mq-source.json file and the content should look like as shown in Example 1-64. Transfer the mq-source.json in binary format to z/OS Unix System Services to the following directory /u/maitra/kafka/kafka_2.12-2.5.0. The mq-source.json file should remain in ASCII format.

Example 1-64  mq-source.json

```json
{
    "name": "mq-source",
    "config": {
        "connector.class": "com.ibm.eventstreams.connect.mqsource.MQSourceConnector",
        "tasks.max": "1",
        "mq.queue.manager": "MQR1",
        "mq.connection.mode": "bindings",
        "mq.queue": "FROM.MQ",
        "mq.record.builder": "com.ibm.eventstreams.connect.mqsource.builders.DefaultRecordBuilder",
        "topic": "fromMQ",
        "key.converter": "org.apache.kafka.connect.storage.StringConverter",
        "value.converter": "org.apache.kafka.connect.converters.ByteArrayConverter"
    }
}
```

Transfer the .jar file to z/OS Unix System Services to the following directory: /u/maitra/kafka/kafka_2.12-2.5.0.

If you are not entitled to IBM Event Streams, you can follow the steps outlined in the following links to build the .jar file and get the sample configuration file:

https://github.com/ibm-messaging/kafka-connect-mq-source#building-the-connector
5. Convert the properties file to EBCDIC format to run in z/OS Unix System Services.
   - Copy the connect-distributed.properties file into the current directory as shown in Example 1-65.

   **Example 1-65  Copy connect-distributed.properties**
   ```
   cp config/connect-distributed.properties ./connect-distributed.properties.orig
   ```

   - Convert the properties file to EBCDIC encoding and replace the original as shown in Example 1-66.

   **Example 1-66  Convert property file to EBCDIC**
   ```
   iconv -f ISO8859-1 -t IBM-1047 ./connect-distributed.properties.orig > config/connect-distributed.properties
   ```

6. Update the Kafka Connect configuration
   - The connect-distributed.properties file must include the correct bootstrap.server for your Apache Kafka install. Our bootstrap server is 129.40.23.76:19092. Update the connect-distributed.properties file as shown in Example 1-67.

   **Example 1-67  Updated connect-distributed.properties**
   ```
   bootstrap.servers=129.40.23.76:19092
   group.id=connect-cluster
   key.converter=org.apache.kafka.connect.json.JsonConverter
   value.converter=org.apache.kafka.connect.json.JsonConverter
   key.converter.schemas.enable=true
   value.converter.schemas.enable=true
   offset.flush.interval.ms=10000
   offset.storage.topic=connect-offsets
   offset.storage.replication.factor=3
   offset.storage.partitions=25
   status.storage.topic=connect-status
   status.storage.replication.factor=3
   status.storage.partitions=5
   config.storage.topic=connect-configs
   config.storage.replication.factor=3
   ```

7. Configure the environment to run the connector.
   - The IBM MQ connectors use the JMS API to connect to MQ. We must set the environment variables required for JMS applications before running the connectors on IBM z/OS.

   We must ensure that we set CLASSPATH to include com.ibm.mq.allclient.jar, and also set the JAR file for the connector we are using - this is the connector JAR file we downloaded from the Event Streams UI or built after cloning the GitHub project, for example, kafka-connect-mq-source-1.1.1-jar-with-dependencies.jar as shown in Example 1-68.

   **Example 1-68  Set CLASSPATH**
   ```
   CP1=/usr/lpp/mqm/V9R1MQ/java/lib/com.ibm.mq.allclient.jar
   ```
We are using the bindings connection mode for the connector to connect to the queue manager, so you must also set the following environment variables:

- The STEPLIB used at run time must contain the IBM MQ SCSQAUTH and SCSQANLE libraries. Specify this library using the .profile file. We will add the IBM MQ SCSQAUTH and SCSQANLE libraries using a line in our .profile file as shown in Example 1-69, replacing the high level qualifier with the high-level data set qualifier that we chose when installing IBM MQ as shown in Example 1-69.

Example 1-69   Add MQ STEPLIB

```
export STEPLIB=MQ910.SCSQAUTH:MQ910.SCSQANLE
```

- The connector needs to load a native library. Set LIBPATH to include the directory of the MQ installation as shown in Example 1-70

Example 1-70   Set LIBPATH

```
L1=/usr/lpp/mqm/V9R1M0/java/lib
export LIBPATH=$L1
```

### 1.9.2 Starting Kafka Connect on z/OS

Kafka Connect is started using a bash script. To run the Kafka connect in distributed mode, use the following steps:

- Start Kafka Connect in distributed mode
  - Navigate to the Kafka directory and run the connect-distributed.sh script, passing in the connect-distributed.properties file as shown in Example 1-71

Example 1-71   Run connect-distributed

```
cd /u/maitra/kafka/kafka_2.12-2.5.0
./bin/connect-distributed.sh connect-distributed.properties
```

- To start an individual mq-source connector, use the Kafka Connect REST API as shown in

Example 1-72   Start source connector

```
curl -X POST -H "Content-Type: application/json"
http://localhost:8083/connectors -d @mq-source.json
```
1.9.3 Status of plug-ins and connectors

To see if the MQ connectors are installed properly, open a browser and navigate to the following URL with your own host name:

http://wtsc59.pbm.ihost.com:8083/connector-plugins

You would see the components that are installed as shown in Example 1-73. This shows that the connectors are installed correctly.

Example 1-73   Installed plug-ins

0:
  class: com.ibm.eventstreams.connect.mqsink.MQSinkConnector"
  type: sink"
  version: 1.1.1"
1:
  class"com.ibm.eventstreams.connect.mqsource.MQSourceConnector"
  type"source"
  version"1.1.1"
2:
  class"org.apache.kafka.connect.file.FileReaderSinkConnector"
  type"sink"
  version"2.5.0"
3:
  class"org.apache.kafka.connect.file.FileReaderSourceConnector"
  type"source"
  version"2.5.0"

To see the status of the mq-source connector, open the browser and point to the following URL with your own host name:

http://wtsc59.pbm.ihost.com/connectors/mq-source/status

You will see the status of the mq-source connector as shown in Figure 1-11.

Figure 1-11   Status of MQ Source Connector
1.9.4 Sending messages from MQ to Kafka

In the previous steps, we have setup the MQ connectors on z/OS Unix System Services and also started the mq-source connector. We are going to send messages from an MQ Queue on z/OS to a Kafka topic on zCX. We have the infrastructure in place that is shown in Figure 1-12 on page 72.

We have the following running:
- Queue Manager on z/OS: MQR1
- Local Queue on MQR1: FROM.MQ
- MQ-Source connector worker running on Unix System Services
- Kafka Cluster running on zCX: bootstrap server -> 129.40.23.76:19092
- Kafka Topic: fromMQ

We are going to put a message on queue FROM.MQ on Queue Manager MQR1 using the MQ Explorer tool as shown in Figure 1-13.

We will now put the message shown in Figure 1-14 on page 73.
Let us now check the current depth of the queue FROM.MQ. We will see that the current depth of the queue is 0. It means that the mq-source connector has consumed the message as shown in Figure 1-15.

If the mq-source connector has worked properly, we will see the message in the Kafka topic from MQ on the Kafka cluster running on zCX. We will use the kafkacat utility to browse the content of the topic. Logon to zCX and run the command shown in Example 1-74.

```
docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.76:19092 -t fromMQ"
```

The output of the command is shown in Example 1-75. It shows that the message we put in the queue FROM.MQ has landed on the Kafka topic from MQ. This concludes our test of the connector.

```
docker exec -it kafka-app-1 bash -c "kafkacat -b 129.40.23.76:19092 -t fromMQ"
```

For more information on any of the topics found in this chapter, see our references in “Related publications” on page 235.
1.10 Sending CICS events to Kafka

In this section we show how CICS events can be sent to Kafka servers running in zCX.

1.10.1 Why CICS events?

CICS is used by many businesses to run core applications. When these applications are executed many events occur which can be of interest to other parts of the business. For example a business may want to send an SMS message to a customer when a large amount of money is withdrawn from their bank account.

One way to achieve that is to modify the program, put it through testing and then implement in production.

This can take some time to implement and require a lot of testing.

CICS events provides a way to quickly implement the capture of an event like this and its subsequent processing, without requiring any program changes. The advantage is that implementing a CICS Event can be done far quicker then a comparable program change with less risk of disruption.

The following link provides a starting point for further detailed information about CICS Events:

https://www.ibm.com/support/knowledgecenter/SSGMCP_5.6.0/fundamentals/event-processing/dfhep_definition.html

For more information on CICS Events, see Event Processing with CICS, SG24-7792.

1.10.2 CICS to Kafka overview

Figure 1-16 shows the configuration we set up to demonstrate using CICS Events to capture a business event and sending the captured data to a topic in Kafka. We used a CICS sample application to demonstrate sending a CICS event to Kafka in zCX. A copy of this example can be downloaded by downloading the CicsToKafkaProjects.zip file. The instructions to download this file can be found in Appendix A, “Additional material” on page 233. The file contains the following projects:

CatalogEvents Contains the CICS event
CicsToKafkaDemo Contains the ZKAFKA Java program
CicsToKafkaBundle Bundle definition used to install the ZKAFKA Java program.
Flow of the example
The following steps explain how our example functions. Section 1.10.3, “Components of the example” on page 76 describes each component in this flow.

1. The CICS transaction EGUI is run in a 3270 screen
2. The display in Figure 1-17 shows an order being placed.

3. When Enter is pressed on the above display, the program issues the following CICS API:

```cics
EXEC CICS REWRITE FILE(EXMPCAT) FROM(Data)
```

4. The CICS event detects the above API and at the capture point it collects data from that event, in this case it collects all the updated data being written to the VSAM file called EXMPCAT.

5. The CICS EGUI transaction completes as normal.
6. The CICS Event then starts a new CICS transaction called ZOEV, passing the collected data via a CICS Container.

7. The started transaction invokes a Java program called ZKAFKA which is a Java program running in a Liberty server in CICS.

8. The Java program reads the data from the CICS container and then sends it to a topic called fromCICSEVENT in the Kafka server running in zCX.

9. A Kafka client program running on MS Windows is used to echo the message that was sent to the Kafka topic by CICS.

### 1.10.3 Components of the example

In this section, we explain the key components that we used to set up this example.

**CICS sample application**

CICS supplies a sample application called the Catalog Manager application. Details on this application and how to set it up in CICS can be found here:

https://www.ibm.com/support/knowledgecenter/SSGMCP_5.6.0/reference-samples/web-services/dfhxa_t100.html

We set this application up in a CICS region called CD54000.

The application is invoked using a transaction called EGUI which is supplied in the sample Catalog Manager application.

**CICS Event bundle**

We created a CICS Event to capture when the Catalog Manager Application updates the EXMPCAT file. The update occurs when an order is placed using the 3270 screen. The event traps on the program performing this command:

```
EXEC CICS REWRITE FILE(EXMPCAT) FROM(Data)
```

When the CICS Event is triggered by the above command it is configured to start a CICS transaction called ZOEV running a program called ZKAFKA.

**ZKAFKA Java program**

A Java program called ZKAFKA runs in a Liberty server in CICS. It reads the data passed to it in CICS containers and sends the data to the Kafka server running in zCX.

**MS Windows Kafka client**

We used a supplied Kafka client program to subscribe to the Kafka topic, to display the messages written to Kafka by the ZKAFKA program in CICS.

### 1.10.4 ZKAFKA Java program

We used a Java program called ZKAFKA that runs in a Liberty server in CICS to send the data captured by the CICS Event to the Kafka server running in zCX.

Example 1-76 on page 77 shows the Java code used to obtain the data passed in the CICS containers.
Example 1-76  Java code to read contents of the CICS Containers

```java
// Get data from all containers

Container contContext = eventChlData.getContainer("DFHEP.CCECONTEXT");
System.out.println(cTime + ">>>" + contContext.getName() + ": " + contContext.getString());

Container contName = eventChlData.getContainer("DFHEP.NAME.00001");
System.out.println(cTime + ">>>" + contName.getName() + ": " + contName.getString());

Container contData = eventChlData.getContainer("DFHEP.DATA.00001");
System.out.println(cTime + ">>>" + contData.getName() + ": " + contData.getString());
```

Example 1-77 shows the Java code used to send data to topic in Kafka.

Example 1-77  Java code to send data to Topic in Kafka

```java
// Specify TCP/IP address and port where Kafka server is located
String KAFKA_BROKER_LIST = "129.40.23.77:19092";
String CLIENT_ID = "zzz";
// Specify name of Kafka topic to send to
String TOPIC = "fromCICSEVENT";
Properties properties = new Properties();
properties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, KAFKA_BROKER_LIST);
properties.put(ProducerConfig.CLIENT_ID_CONFIG, CLIENT_ID);
properties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringSerializer");
properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringSerializer");

KafkaProducer producer = new KafkaProducer<>(properties);
ProducerRecord data = new ProducerRecord(TOPIC, eventData);

// Set up for call back after data has been sent to Kafka
TestCallback callback = new TestCallback();

// Send the data to Kafka
producer.send(data, callback);
```

We coded the Java program to write a number of messages to the Liberty server STDOUT log.
1.10.5 Testing the example

Figure 1-17 on page 75 showed how we ran the CICS Catalog Sample Application to place an order.

On our system, the STDOUT log for the Liberty server running in CICS could be viewed in the following directory:

/u/edmcar/CD54000/WLP54000/CURRENT.STDOUT

When Enter was pressed, the output from the ZKAFKA Java program can be seen in the above file and is shown in Example 1-78.

Example 1-78  Output from ZKAFKA Java program

```
2020/07/15 01:13:31>>>cont: DFHEP.CCECONTEXT
2020/07/15 01:13:31>>>cont: DFHEP.NAME.00001
2020/07/15 01:13:31>>>cont: DFHEP.DATA.00001
2020/07/15 01:13:31>>>DFHEP.CCECONTEXT: EPFE0002zOfficeOrdersEvents
2020/07/15 01:13:31>>>DFHEP.NAME.00001: OrderData
2020/07/15 01:13:31>>>DFHEP.DATA.00001: 0010Ball Pens Black 24pk
2020/07/15 01:13:31>>>Data sending to Kafka: EPFE0002zOfficeOrdersEvents
2020/07/15 01:13:31>>>Time for first send of data of length: 320 to Kafka: 381ms
2020/07/15 01:13:31>>>Time for second send of data of length: 320 to Kafka: 0ns
2020/07/15 01:13:31>>>CICS sent message to topic: fromCICSEVENT partition:1 offset:14
```

We set up the ZKAFKA Java program to send the data twice to the topic in Kafka.

You can see that the first send of the data took 381ms. This is to be expected as the first send requires classes to be loaded, a TCPIP connection to be established between CICS and the Kafka server running in ZCX.

When the ZKAFKA Java program then does a second send of the data, as the TCPIP connection has been established and classes loaded, the second send took such a small time, it showed as zero nanoseconds.

In a real production environment, you could consider setting up an Object Pool design pattern.

The approach would be to create an object pool of KafkaProducer objects, with these objects being in effect already created connections to the Kafka sever. The Java program could get a Kafka Producer object from the pool, use it and then return it to the pool.

This would save the Java program from having to create a connection to the Kafka server on each call and should improve performance.

One open source project that could be used for this is the Apache Commons Pool project (org.apache.commons.pool2).

On a Windows PC, we installed the Kafka product code, and ran a supplied Kafka consumer using this command:

```
kafka-console-consumer.bat --bootstrap-server 129.40.23.77:19092 --topic fromCICSEVENT --from-beginning
```
When we placed the order in Windows, we saw the output shown in Example 1-79 on page 79.

Example 1-79  Data from topic in Kafka

<table>
<thead>
<tr>
<th>EPFE0002zOfficeOrdersEvents</th>
<th>zOfficeOrder</th>
</tr>
</thead>
<tbody>
<tr>
<td>1910E4E2C9C2D4E2C34BE3C3D7F5F5F0F2F9380F7BF57AE2000100USIBM itemName</td>
<td>0030Ball Pens Red 24pk</td>
</tr>
<tr>
<td>2020-07-15T01:17:12.803+00:00</td>
<td>010002.900105000</td>
</tr>
</tbody>
</table>

This showed that the CICS event had captured the updated data, and that the Java program in CICS had successfully sent that data to the Kafka server.

1.10.6 Developing the ZKAFKA Java program

The following shows how we developed the ZKAFKA Java program:

- We used IBM Rational® Application Developer for WebSphere Software V9.7.0 to do the development.
- We used z/OS Explorer to develop the Java program.

To obtain the z/OS Explorer, we went to the following link:


Figure 1-18 shows the download screen for z/OS Explorer.
Download and install

These instructions will show you how to install a new instance of Eclipse with the selected products, using Eclipse p2, from an online IBM repository.

Step 1 - Downloading and extracting the platform executable

1. Download and extract the package to a location of your choosing.
2. Launch the platform application by running "zosexplorer.exe". Why is the executable from z/OS Explorer?

   We clicked on the 64-bit button, after which we were presented with a display asking to agree to the licensing agreement. After accepting the agreement, a file called cicsexplorer-aqua-5.5.9-win32-x86_64.zip of 601MB was downloaded to our PC.

   We uncompressed this file into a parent directory called C:\zProducts\zosExp559. Then from the directory C:\zProducts\zosExp559\IBM Explorer for z/OS, we started the program zosexplorer.exe.

   Once z/OS Explorer has started, we selected Help, then 'Install New Software...'. In the displayed window, select the drop down for 'Work with:' and select the entry that starts with IBM Explorer for z/OS Update Site as shown in Figure 1-19.
Select IBM CICS Explorer® from the list of available products as shown in Figure 1-20 on page 81.

Once installed, you are then able to use IBM Explorer for z/OS and its suite of tools to develop applications.
IBM App Connect Enterprise

This chapter discusses the following topics:

- “Technical and architectural concepts of ACE” on page 84
- “Installing IBM App Connect Enterprise” on page 89
- “Configuration details” on page 98
- “Deploying an application to ACE to integrate with CICS” on page 98
2.1 Technical and architectural concepts of ACE

Introduced in 1999, MQ Series integrator was rebranded as WebSphere Business Integration Message Broker and then WebSphere Message Broker. In 2013, IBM Integration Bus (IIB) v9 enabled users to import and convert existing integration assets from WebSphere Enterprise Service Bus, converging these capabilities into IIB.

In 2018, IBM released App Connect Enterprise (ACE) which combined the existing, industry-trusted technologies of IBM Integration Bus (IIB) with IBM App Connect capabilities (professional software and Cloud Connectors via IBM Cloud® managed service). The runtime of IIB was also updated in this release to make it cloud native so the term “Bus” in the name didn’t accurately reflect the possibilities that this new architecture opened up.

IBM App Connect Enterprise re-architected the core IBM Integration Bus technology, to make it amenable for deployment in container based architectures, while also continuing support for older design paradigms such as the Enterprise Service Bus pattern. This makes IBM App Connect Enterprise v11 an excellent choice for production systems both for users wishing to embrace the benefits of containers as part of an Agile Integration Architecture, and also users looking to make minimal changes to their IBM Integration Bus architecture yet benefit from the new features and product lifespan of v11. Figure 2-1 depicts the evolution of IBM App Connect Enterprise (ACE).

Before we discuss IBM App Connect Enterprise, we discuss what we mean by the term “integration”. There are three concepts that are involved in connecting applications together:
ENDPOINTS: The different communication protocols
INPUTS: The data exchanged between endpoints in different formats
INTEGRATION: The mediation patterns for interoperation between endpoints – message transformation, enrichment, audit, aggregation and scaling

Applications need to talk with each other over a communications protocol. Typical protocols in use today include TCP/IP, and higher level protocols such as MQ, FTP, SMTP and HTTP.

Applications exchange data over the communications protocol, typically in discrete structures known as messages. The format of these messages can be defined from JSON, C structures or COBOL copybooks (for example), or simply use a standard format such as XML.

In order to connect applications together so that their protocols and message formats interoperate, mediation patterns need to be applied to one or both systems you’re trying to connect. These mediation patterns can be relatively straightforward, for example, routing messages from one place to another, or the transformation of one message format into another to relatively complex patterns such as aggregating multiple outputs from an application into a single message for a target system.

IBM App Connect Enterprise enables “universal connectivity” by integrating protocols, message formats and mediation patterns.

ACE provides the ability to be an endpoint and to connect to other endpoints. It can do this over a variety of protocols and using a variety of message formats, sometimes with more than one in use at each time.

ACE also supports a wide range of mediation patterns, helping to support the use of the various message formats and protocols in many ways. Figure 2-2 shows the positioning of ACE in an enterprise.

![Figure 2-2 ACE in an enterprise](image)

### 2.1.1 Key concepts of ACE

We have discussed some of the qualities and capabilities of the ACE product, Figure 2-3 on page 86 shows some of the key concepts of ACE.
The image in the middle of Figure 2-3 represents the IBM App Connect Enterprise runtime. It can connect multiple different applications and systems together. Included in the applications that can be connected are the following:

- MQ applications
- Mainframe applications (like CICS / IMS) or Enterprise Information Systems like SAP etc
- Databases
- Files
- REST API’s
- Web Services

In ACE, we can define specific integrations between one system and one or more other systems. For example, an MQ applications can be integrated with CICS and writing to a file based application.

In ACE, the logic of this integration is described by a message flow. A message flow is fancy way to write a program where reusable blocks called nodes are dragged to the canvas and wired together with some transformation code and integration logic. Message flows are general purpose, reusable integration applications.

The message flow is a key concept in IBM App Connect Enterprise/IBM Integration Bus.

### 2.1.2 Runtime Components of ACE

In this section, we provide an overview of the ACE product architecture. Figure 2-4 shows the runtime components of ACE.
The main components of ACE are the following:

- **The Integration Toolkit**
  The Integration Toolkit is the development environment. Based on the Eclipse platform, all the objects required to perform application integration using ACE are developed, deployed and tested here. It provides standard ways to build integration applications and perform version control.

- **The Integration Node and Integration Server**
  The Integration node (or broker) is the container that hosts Integration servers (or execution groups). Each integration server is an operating system process that contains a pool of threads responsible for running the message flows that are deployed to it.

  Message flows are deployed to integration servers in applications which may contain reusable sets of resources. The integration servers directly interact with the endpoints that are being integrated.

- **The Web UI**
  The web user interface (Web UI) provides administration capability, including monitoring of deployed objects and the ability to start, stop, delete, deploy and manage workload.

### 2.1.3 ACE runtime in zCX

In IBM App Connect Enterprise integration servers can be deployed in one of two ways – one or more integration servers under the management of an integration node or as independent integration servers.
In zCX, the ACE runtime is running as Independent integration servers. They are started directly through an external framework within a Docker container. The independent integration runtimes are used for container-based and micro services-aligned architectures.

Figure 2-5 shows the ACE runtime integration server in zCX.

![Integration Server in zCX](image)

This lightweight, cloud-native runtime (integration server) can be used in the following ways:

- As part of a DevOps / agile approach - the lightweight container spins up in seconds.
- In a microservices model - managed and deployed by the microservice teams close to the microservice.
- Across multiple clouds – private, public, or as a fully managed cloud service.

Some other key features of the independent Integration Server are as follows:

- Deploy an application to an Integration Server from Toolkit, REST Commands, and Web UI.
- Display one of more local/remote independent Integration Servers which can be deployed to.
- Toolkit communications with servers via an administrative REST API. Bar files are deployed to servers using this REST API.
- View and administer independent Integration Servers in the Web user interface.
2.1.4 Reasons to run ACE on zCX

The three main reasons to run ACE on zCX are as follows.

- **Server Consolidation**
  - If you are currently running ACE integration servers off platform, you can now move them to z/OS and manage them along with applications and data they serve such as CICS or IMS applications.
  - It also means that the z/OS qualities of service such as scalability, availability, integrated disaster recovery, backup, WLM and security are available to the integration servers.
- **Save money**
  - If you are currently running IIB v10 on z/OS and have any particularly MIPs heavy message flows, you may consider moving those flows into an environment where majority of processing can be offloaded to ZIIP processors to reduce costs.
- **Skills**
  - If deep IBM integration Bus experience lies in z/OS system programmers, then you should run integration servers where knowledge and experience lies.

For more information on running ACE on zCX, see the IBM Statement of Direction found at the following website:


2.2 Installing IBM App Connect Enterprise

There is currently no prebuilt image available at present to install IBM ACE in a zCX image. However it is possible to install ACE with a step-by-step approach. This section will explain in detail how to install and get ACE up and running. Some of the key points to note are as follows:

- You can run v11 (Fix Pack 8 onwards) ACE integration server on z/OS!
- No prebuilt image is available at the time of writing of this IBM Redbooks publication.

2.2.1 Create an intermediate image to pull data from GitHub

In this section we guide you in creating an intermediate image to pull data from GitHub.

1. Log on to the zCX instance where the ACE Image should be built. After logging on, the default path is `~/home/admin`. Execute the following command:

   ```bash
   mkdir redbook && cd redbook
   ```

2. Build an intermediate image to download further data from the ace-docker Git repository found here: [https://github.com/ot4i/ace-docker](https://github.com/ot4i/ace-docker)

3. Create the intermediate image is shown in Example 2-1.

   **Example 2-1** Start the dummy git container

   ```bash
   admin@sc74cn11:~$ redbook$ docker run --name dummy-git -it --entrypoint /bin/bash -d ubuntu
   ```
4. Once the image is up and running, log on to the image as shown in Example 2-2.

Example 2-2  Logon to the dummy git container

```
admin@sc74cn11:~$ redbook$ docker exec -it dummy-git bash
root@c0423231dc39:~#
```

5. Install the net-tools package which includes important tools for controlling the network subsystem of the Linux kernel. This includes arp, hostname, ifconfig, netstat, rarp and route, as shown in Example 2-3.

Example 2-3  Install the required tools in the dummy git container

```
root@c0423231dc39:~# apt update
Get:2 http://ports.ubuntu.com/ubuntu-ports bionic-updates InRelease [88.7 kB]
Fetched 15.7 MB in 5s (3485 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
4 packages can be upgraded. Run 'apt list --upgradable' to see them.

root@c0423231dc39:~# apt install -y git net-tools vim wget
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ca-certificates file git-man krb5-locales less libasn1-8-heimdal libbsd0
  libcurl3-gnutls libedit2 liberror-perl libexpat1 libgdbm-compat4 libgdbm5
  libgpm2 libgssapi-krb5-2 libgssapi3-heimdal libcrypto4-heimdal
  libheimibase1-heimdal libheimntlm1-0-heimdal libhx509-5-heimdal libk5crypto3
  libkeyutils1

Setting up xauth (1:1.0.10-1) ...
Setting up libdlap-2.4-2-s390x (2.4.45+dfsg-1ubuntu1.5) ...
Setting up libcurl13-gnuTLSs390x (7.58.0-2ubuntu3.8) ...
Setting up git (1:2.17.1-lubuntu0.7) ...
Processing triggers for libcurl13-gnuTLSs390x ...
Processing triggers for libcrypto4-heimdal ...
Processing triggers for ca-certificates (20190110-18.04.1) ...
Updating certificates in /etc/ssl/certs...
  0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
```

6. Once prerequisite tools are installed, continue to download (clone) the Git repository. Example 2-4 shows how to clone the repository.
Example 2-4  Clone the git repository

```bash
root@c0423231dc39:/# git clone https://github.com/ot4i/ace-docker.git
Cloning into 'ace-docker'...
remote: Enumerating objects: 2340, done.
remote: Total 2340 (delta 0), reused 0 (delta 0), pack-reused 2340
Receiving objects: 100% (2340/2340), 3.22 MiB | 19.86 MiB/s, done.
Resolving deltas: 100% (897/897), done.
```

7. Once the download has completed successfully, change into the newly created directory.

```bash
cd ace-docker
```

Example 2-5 lists all the files that should be downloaded from the Git repository.

Example 2-5  Verify the downloaded git files

```bash
root@c0423231dc39:/ace-docker# ls -l
total 208
-rw-r--r-- 1 root root 1881 Jun 14 11:17 CHANGELOG.md
-rw-r--r-- 1 root root 83670 Jun 14 11:17 LICENSE
-rw-r--r-- 1 root root 22365 Jun 14 11:17 README.md
-rw-r--r-- 1 root root 533 Jun 14 11:17 ace_compile_bars.sh
-rw-r--r-- 1 root root 1439 Jun 14 11:17 ace_config_agent.sh
-rw-r--r-- 1 root root 563 Jun 14 11:17 ace_config_bars.sh
-rw-r--r-- 1 root root 704 Jun 14 11:17 ace_config_extensions.sh
-rw-r--r-- 1 root root 2392 Jun 14 11:17 ace_config_keystore.sh
-rw-r--r-- 1 root root 1504 Jun 14 11:17 ace_config_logging.sh
-rw-r--r-- 1 root root 733 Jun 14 11:17 ace_config_odbcini.sh
-rw-r--r-- 1 root root 1020 Jun 14 11:17 ace_config_policy.sh
-rw-r--r-- 1 root root 676 Jun 14 11:17 ace_config_serverconf.sh
-rwxr-xr-x 1 root root 1837 Jun 14 11:17 ace_config_setdbparms.sh
-rw-r--r-- 1 root root 724 Jun 14 11:17 ace_config_ssl.sh
-rw-r--r-- 1 root root 1409 Jun 14 11:17 ace_config_truststore.sh
-rw-r--r-- 1 root root 3912 Jun 14 11:17 ace_config_webusers.sh
-rw-r--r-- 1 root root 1030 Jun 14 11:17 ace_discover_port_overrides.sh
-rw-r--r-- 1 root root 378 Jun 14 11:17 ace_env.sh
-rwxr-xr-x 1 root root 740 Jun 14 11:17 ace_integration_port_overrides.sh
-rw-r--r-- 1 root root 1454 Jun 14 11:17 ace_license_check.sh
-rw-r--r-- 1 root root 386 Jun 14 11:17 ace_mqsicommand.sh
-rw-r--r-- 1 root root 11691 Jun 14 11:17 app_connect_light_256x256.png
-rw-r--r-- 1 root root 4383 Jun 14 11:17 appconnect_enterprise_logo.svg
drwxr-xr-x 1 root root 36 Jun 14 11:17 asoc
drwxr-xr-x 1 root root 1638 Jun 14 11:17 build-rhel.sh
drwxr-xr-x 1 root root 72 Jun 14 11:17 cmd
drwxr-xr-x 1 root root 42 Jun 14 11:17 deps
drwxr-xr-x 1 root root 152 Jun 14 11:17 experimental
drwxr-xr-x 1 root root 56 Jun 14 11:17 internal
drwxr-xr-x 1 root root 44 Jun 14 11:17 licenses
drwxr-xr-x 1 root root 186 Jun 14 11:17 sample
drwxr-xr-x 1 root root 188 Jun 14 11:17 ubi
drwxr-xr-x 1 root root 56 Jun 14 11:17 vendor
```

The downloaded files will be used for input to built the ACE Docker image. The files must be copied into the underlying zCX instance by using Docker commands. To do this, leave the intermediate dummy-git container with the `exit` command.

8. Use the following command from the zCX image to copy the files into the local zCX image.
docker cp dummy-git:/ace-docker .

9. After copying the files from the dummy Git image to the local zCX instance, a docker volume is needed to store the data from the ACE Image. Create the docker image with the following command:

```
docker volume create ace-only-data
```

Verify if the volume has been created as shown in Example 2-6.

**Example 2-6  List the created ace-only-data volume**

```
admin@sc74cn11:~/redbook$ docker volume ls | grep ace-only-data
local ace-only-data
```

10. Download additional binaries from the Docker hub.

**Important Note:** To download files directly from the Docker hub you must have a Docker ID. An ID can be created at this website: https://hub.docker.com.

If the download is attempted without a valid logon an error is shown, as shown in Example 2-7.

**Example 2-7  Error when attempting a build directly from Docker without a valid logon ID.**

```
admin@sc74cn11:~/redbook$ docker build -t aceonly-img -f ace-docker/ubi/Dockerfile.aceonly .
```

Sending build context to Docker daemon 32.34MB

Step 1/43 : FROM golang:1.10.3 as builder

Get https://registry-1.docker.io/v2/library/golang/manifests/1.10.3: unauthorized: incorrect username or password

To build the image without the authentication error, log on before the Docker build command is executed, as shown in Example 2-8.

**Example 2-8  Successful download from docker**

```
admin@sc74cn11:~/redbook$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: <your_user_name>
Password:
WARNING! Your password will be stored unencrypted in /home/admin/.docker/config.json.
Configure a credential helper to remove this warning. See https://docs.docker.com/engine/reference/commandline/login/#credentials-store
```

Login Succeeded

11. After logging into Docker, the dockerfile that was downloaded from the Git repository needs to be adjusted for zCX, as shown in Example 2-4. Edit the file with the vi editor and change the content to that which is shown in Example 2-9. The old lines are highlighted red and the new lines are highlighted in blue. The red lines must be replaced with the blue lines.

**Example 2-9  Adjust the downloaded file, dockerfile.aceonly**

```
admin@sc74cn11:~/redbook$ vi ace-docker/ubi/Dockerfile.aceonly
FROM golang:1.10.3 as builder
```
WORKDIR /go/src/github.com/ot4i/ace-docker/
ARG IMAGE_REVISION="Not specified"
ARG IMAGE_SOURCE="Not specified"
COPY cmd/ ./cmd
COPY internal/ ./internal
COPY vendor/ ./vendor
RUN go build -ldflags "-X \"main.ImageCreated=$(date --iso-8601=seconds)\" -X "main.ImageRevision=$IMAGE_REVISION\" -X "main.ImageSource=$IMAGE_SOURCE\"" ./cmd/runaceserver/
RUN go build ./cmd/chkaceready/
RUN go build ./cmd/chkacehealthy/
# Run all unit tests
RUN go test -v ./cmd/runaceserver/
RUN go test -v ./internal/...
RUN go vet ./cmd/... ./internal/...

ARG ACE_INSTALL=ace-11.0.0.2.tar.gz
ARG ACE_INSTALL=11.0.0-ACE-LINUXZ64-FP0009.tar.gz
WORKDIR /opt/ibm
COPY deps/$ACE_INSTALL .
RUN mkdir ace-11
RUN tar -xzf $ACE_INSTALL --absolute-names --exclude ace-11/*.tools --strip-components 1 --directory /opt/ibm/ace-11

FROM registry.access.redhat.com/ubi7/ubi-minimal
FROM registry.access.redhat.com/ubi8/ubi-minimal

ENV SUMMARY="Integration Server for App Connect Enterprise" \
DESCRIPTION="Integration Server for App Connect Enterprise" \
PRODNAME="AppConnectEnterprise" \
COMPNAME="IntegrationServer"

LABEL summary="$SUMMARY" \
description="$DESCRIPTION" \
io.k8s.description="$DESCRIPTION" \
io.k8s.display-name="Integration Server for App Connect Enterprise" \
io.openshift.tags="$PRODNAME,$COMPNAME" \
com.redhat.component="$PRODNAME-$COMPNAME" \
name="$PRODNAME/$COMPNAME" \
vendor="IBM" \
version="11.0.0.6" \
release="1" \
license="IBM" \
maintainer="Hybrid Integration Platform Cloud" \
io.openshift.expose-services=""

# Add required license as text file in Liceses directory (GPL, MIT, APACHE, 
Partner End User Agreement, etc)
COPY /licenses/ /licenses/

# Create OpenTracing directories, and copy in any library or configuration files 
available
RUN mkdir /etc/ACEOpenTracing /opt/ACEOpenTracing /var/log/ACEOpenTracing
COPY deps/OpenTracing/library/* ./opt/ACEOpenTracing/
COPY deps/OpenTracing/config/* ./etc/ACEOpenTracing/

WORKDIR /opt/ibm

RUN microdnf install --nodocs openssl util-linux unzip python2 && microdnf clean all
RUN microdnf install findutils tar git wget vim openssl util-linux unzip python2 && microdnf clean all
COPY --from=builder /opt/ibm/ace-11 /opt/ibm/ace-11
RUN /opt/ibm/ace-11/ace make registry global accept license silently

# Copy in PID1 process
COPY --from=builder /go/src/github.com/ot4i/ace-docker/runaceserver /usr/local/bin/
COPY --from=builder /go/src/github.com/ot4i/ace-docker/chkace* /usr/local/bin/

# Copy in script files
COPY *.sh /usr/local/bin/

ADD https://storage.googleapis.com/kubernetes-release/release/v1.16.0/bin/linux/amd64/kubectl /usr/local/bin/kubectl
RUN wget https://storage.googleapis.com/kubernetes-release/release/v1.18.0/kubernetes-client-linux-s390x.tar.gz
    && tar xvfp kubernetes-client-linux-s390x.tar.gz
    && cp kubernetes/client/bin/kubectl /usr/local/bin/kubectl

RUN chmod +x /usr/local/bin/kubectl && 
    /usr/local/bin/kubectl version --client

# Create a user to run as, create the ace workdir, and chmod script files
RUN useradd -u 1000 -d /home/aceuser -G mqbrkrs,wheel aceuser 
    && su - aceuser -c '. /opt/ibm/ace-11/server/bin/mqsiprofile && 
    mqsicreateworkdir /home/aceuser/ace-server' 
    && chmod 755 /usr/local/bin/*

# Set permissions for OpenTracing directories
RUN chown aceuser:aceuser /etc/ACEOpenTracing /var/log/ACEOpenTracing

12. After editing the file, close the edit session with the :wq command.

### 2.2.2 Get the ACE installation binaries

Download the ACE binaries from IBM Support into the newly created redbook folder that will be used when building the image. In this book, we used V11.0.0 with fixpack FP0009 with Release date of the 28th of May 2020. Further details, with the most recent fixes, can be found at the following website:


Select the ‘Fix Central Download’ Image for the Linux on zSeries platform.
To download the image directly into the zCX instance, select FTPS/SFTP as the download option as shown in Example 2-6.

![Available download options](image)

**Important Note:** You must have an IBM account to download ACE from the IBM Support server as well as a valid licence coupled to the IBM ID used to download the image.

Use the provided User ID and Password to download the file as shown in Example 2-10. Before initiating the download, change the directory by using the following command:

```
cd ace-docker/deps
```

**Example 2-10  Download ACE image from IBM Support**

```
admin@sc74cn11:~/redbook/ace-docker/deps$ sftp <your_user>@delivery04-bld.dhe.ibm.com
The authenticity of host 'delivery04-bld.dhe.ibm.com (170.225.15.104)' can't be established.
RSA key fingerprint is SHA256:QRJpOHdTFuPmP2NLOQHTpB+IrDSNrqve7RadzKcFyFc.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'delivery04-bld.dhe.ibm.com,170.225.15.104' (RSA) to the list of known hosts.
ASMbggPF@delivery04-bld.dhe.ibm.com's password:
sftp> mget *
Fetching /11.0.0-ACE-LINUXZ64-FP0009.tar.gz to 11.0.0-ACE-LINUXZ64-FP0009.tar.gz
/11.0.0-ACE-LINUXZ64-FP0009.tar.gz
100% 690MB 10.1MB/s 01:08
2.2.3 Build the ACE docker image

Once the dockerfile has been edited and saved successfully, the Docker image is downloaded from IBM Support and can be built, as shown in Example 2-11.

Example 2-11 Build the ACE Docker image

```bash
admin@sc74cn11:~/redbook/ace-docker/deps$ cd ..
admin@sc74cn11:~/redbook/ace-docker$ docker build -t aceonly-img -f ubi/Dockerfile.aceonly .
```

Sending build context to Docker daemon 32.34MB
Step 1/43 : FROM golang:1.10.3 as builder
  ---> 1311442b6183
Step 2/43 : WORKDIR /go/src/github.com/ot4i/ace-docker/
  ---> Using cache
  ---> 24565e9ee58d
<output omitted intentionally>

Step 43/43 : ENTRYPOINT ["runaceserver"]
  ---> Running in 461a644cf95a
Removing intermediate container 461a644cf95a
  ---> bd26c4fa8b62
Successfully built bd26c4fa8b62
Successfully tagged aceonly-img:latest

After building the image, the ACE server can be started with the following command.

```bash
docker run --name aceserver-app-1 -p 7600:7600 -p 7800:7800 -p 7843:7843 -v ace-only-date:/home/aceuser/ace-server --env LICENSE=accept --env ACE_SERVER_NAME=ACESERVER1 aceonly-img
```

You will know the server is up and running as indicated by the server that it is listening on port 7600. See Example 2-12 how to start the ACE image and to verify it is up and running.

Example 2-12 Starting the ACE docker container

```bash
admin@sc74cn11:~acemq/ace-docker$ docker run --name aceserver-app-1 -p 7600:7600 -p 7800:7800 -p 7843:7843 -v ace-only-date:/home/aceuser/ace-server --env LICENSE=accept --env ACE_SERVER_NAME=ACESERVER1 aceonly-img
```

2020-06-14T13:10:53.191Z Image created: 2020-06-14T12:44:29+00:00
2020-06-14T13:10:53.192Z Image revision: Not specified
2020-06-14T13:10:53.313Z ACE version: 11009
2020-06-14T13:10:53.313Z ACE level: S000-L200527.16701
2020-06-14T13:10:53.313Z ACE build type: Production, 64 bit, s390x_linux_2
2020-06-14T13:10:53.313Z Checking for valid working directory
2020-06-14T13:10:53.313Z Checking if work dir is already initialized
2020-06-14T13:10:53.313Z Checking for contents in the work dir
2020-06-14T13:10:53.314Z Work dir initialization complete
2020-06-14T13:10:53.314Z Performing initial configuration of integration server
2020-06-14T13:10:53.314Z No content server url available
2020-06-14T13:10:53.314Z Initial configuration of integration server complete
2020-06-14T13:10:53.314Z Discovering override ports
2020-06-14T13:10:53.320Z Successfully discovered override ports
2020-06-14T13:10:53.320Z Starting integration server
2020-06-14T13:10:53.320Z No default application name supplied. Using the integration server name instead.
2020-06-14T13:10:53.321Z Waiting for integration server to be ready
2020-06-14T13:10:53.323Z Integration server not ready yet
......2020-06-14 13:10:53.684216:2020-06-14 13:10:53.684388: Integration server 'ACESERVER1' starting initialization; version '11.0.0.9' (64-bit)
2020-06-14T13:10:58.326Z Integration server not ready yet
2020-06-14 13:10:58.797108: The HTTP Listener has started listening on port '7600' for 'RestAdmin http' connections.
2020-06-14 13:10:58.812824: Integration server has finished initialization.
2020-06-14T13:11:03.332Z Integration server is ready
2020-06-14T13:11:03.332Z Metrics are disabled

The ACE WebUI is accessible via http://<your_zcx_instance_name>:7600.

Figure 2-7 shows the web user interface for the ACE Server.

![Figure 2-7   UI from the running ACE instance](image)

To stop the running container, use the following command:

docker stop aceserver-app-1

The container will shut down cleanly, stopping the interrogation server.
2.3 Configuration details

After installing ACE on zCX, our configuration appears as shown in Figure 2-8.

![Figure 2-8  zCX configuration](image)

The following are the details of our configuration:

- Integration server name: ACESERVER1
- Listener port for administration (including Web UI): 7600
- HTTP listener port: 7800
- HTTPS listener port: 7843

We will use this integration server for our use case scenario. We will deploy an application that will receive a REST API call and invoke a CICS program to query a customer and send the response back to the REST client.

2.4 Deploying an application to ACE to integrate with CICS

The intent of the use case in this section is to show the deployment of an application to the ACE integration server in zCX. The application is a message flow that accesses a CICS application on the z/OS system. The CICS application is a very simple one that returns a fictional customer address based on a customer number as input, between 1 and 10.

The use case is shown in Figure 2-9.
The key points of this use case show the following:

- The message flow is built in the ACE toolkit.
- The message flow accesses a CICS COBOL program, CUSTINQ.
- The message flow is exposed as a REST API in port 7800 for external clients.
- The CICS program expects data in COPYBOOK format.
- The incoming REST input data is in JSON format.
- The output to the REST client is in JSON format.

**The Message Flow**

The message flow is built in the ACE toolkit. We do not describe how to build the message flow in this book. The message flow looks similar to that shown in Figure 2-10.
The message flow does the following

1. Receives the REST call

   - Identifies the flow as a REST API with a query parameter accepting customer number. The input data to the flow from the REST client is in JSON format.

2. Converts JSON input data to COPYBOOK format

   - This step converts the JSON format to a copybook format and passes the input customer number to the CustNo field of the record. The copybook is what the CICS program expects in the COMMAREA. The copybook format is shown in Example 2-13.

   **Example 2-13  Copybook for CICS program**

   ```
   01 DFHCOMMAREA.
   02 CustNo          PIC S9(9) COMP-5 .
   02 LastName       PIC A(25).
   02 FirstName      PIC A(15).
   02 Address1       PIC X(20).
   02 City           PIC A(20).
   02 State          PIC A(5).
   02 Country        PIC X(15).
   02 RetCode        PIC S9.
   ```

3. Invoke CICS program

   - The message flow makes a call to the CICS program CUSTINQ from the CICS node with the following parameters as shown in Figure 2-11. Port 3001 is the IPIC listener port in CICS.
4. Convert from COPYBOOK to JSON format.
   In this step, the response from CICS is converted from COPYBOOK format to JSON format.

5. Reply with response back to the REST client.
   The JSON response is sent back to the REST client that made the request.

### 2.4.1 Deploy to ACE runtime in zCX

To deploy a solution to an environment, you can package the resources into a BAR (Broker Archive) file. You can add applications and libraries to the BAR file. The BAR file is the deployment artifact to the ACE integration server. To do this, follow these steps:

1. Create BAR file
   Use the ACE Toolkit to build the BAR file. The BAR file will contain the REST API application we built and the message model of the COPYBOOK structure as shown in Figure 2-12.
Prepare

Select deployable resources to include in the BAR

**Deployable Resources**

Select an application to package all its contained resources. Resources within an application are isolated from other applications.

- Applications, shared libraries, services, and REST APIs
- Policies
- Message flows, static libraries and other message flow dependencies

- Text filter: cust

(*) Resource types marked with * will be automatically added to the BAR if referenced by another selected artifact.

![Image of build and save BAR file](image)

**Figure 2-12  Build and save BAR file**

The generated BAR file is saved as `getCustomerApp.bar` on the desktop as shown in Figure 2-13.

![Image of generated BAR file](image)

**Figure 2-13  Generated BAR file**

2. To open the Web UI in your web browser and use listener port 7600 as follows:

   http://<your zcx instance>:7600

   This will bring up the Web UI with no deployments as shown in Figure 2-14.
3. Deploy the BAR file that was created to ACESERVER1 by clicking on the Deploy button.
4. Add the BAR file, `getCustomerApp.bar` that was created, as shown in Figure 2-15.

Deploy the BAR file by clicking on the Deploy button as shown in Figure 2-16.
5. After successful deployment, the API and the message model are visible on the Web UI as shown in Figure 2-17.
2.4.2 Using the Web UI to test deployed REST API’s

The REST API is exposed via port 7800. Starting from ACEv11.0.0.9, ACE is providing a new dedicated component for helping test your deployed REST APIs. Follow these steps to perform the test.

1. On the Web UI, click on the getCustomer API.

2. We will see that the REST API has been deployed to an integration server. Click on the available operation “GET /getCustomerInfo” and switch to the Try it tab. Enter the value of 9 in the custno* field and click on the Send button as shown in Figure 2-18.

![Figure 2-18 Try the REST API](image)

We are using an integration server with default settings. When we attempted to invoke the REST API we received the error shown in Figure 2-19. This was most likely due to a need to enable the integration server to permit cross-origin requests from a web browser by enabling Cross-Origin Resource Sharing (CORS). There are quite a few control options we can select, depending on how open we would like our CORS settings to be.

![Figure 2-19 No Response from REST API](image)

We set one of the open set of configuration options in the integration server’s server.conf.yaml file, CORSEnabled, to true and restarted the server.
3. We executed an interactive bash shell on the ACE container by running the command shown in Example 2-14.

Example 2-14  Executing interactive bash
admin@sc74cn09:~$ docker exec -it aceserver-app-1 bash

4. We dynamically configured the ACE integration server. We created a directory called serverconf and made a copy of the server.conf.yaml file as shown in Example 2-15

Example 2-15
[aceuser@c4679b717b3b ~]$ cd initial-config/
[aceuser@c4679b717b3b initial-config]$ mkdir serverconf
[aceuser@c4679b717b3b initial-config]$ cd serverconf
[aceuser@c4679b717b3b serverconf]$ cp /home/aceuser/ace-server/server.conf.yaml server.conf.yaml

5. Edit the server.conf.yaml file and uncomment and set CORSEnabled to true in the HTTPConnector section as shown in Example 2-16.

Example 2-16  Set CORSEnabled
HTTPConnector:
  #ListenerPort: 0              # Set non-zero to set a specific port, defaults to 7800
  #ListenerAddress: '0.0.0.0'  # Set the IP address for the listener to listen on. Default is
  #AutoRespondToHTTPHEADRequests: false # Automatically respond to HTTP HEAD
  #ServerName: ''               # Set the value to be returned in the 'Server' HTTP header.
  CORSEnabled: true            # Set the value to true to make the listener respond to valid
  #CORSAllowOrigins: '*'      # CORSAllowCredentials: false
  #CORSExposeHeaders: 'Content-Type'
  #CORSMaxAge: -1
  #CORSAllowMethods: 'GET,HEAD,POST,PUT,PATCH,DELETE,OPTIONS'
  #CORSAllowHeaders: 'Accept,Accept-Language,Content-Language,Content-Type'

6. Exit from the bash shell from by typing the exit command as shown in Example 2-17.

Example 2-17  Exit from container bash shell
[aceuser@c4679b717b3b ~]$ exit

7. Stop and start the container to make the configuration take effect as shown in Example 2-18.

Example 2-18  Stop and start container
admin@sc74cn09:~$ docker stop aceserver-app-1
aceserver-app-1
admin@sc74cn09:~$ docker start aceserver-app-1
aceserver-app-1
8. Open the browser and point to this URL: http://<your
zcx>:7800/getcustomer/v1/getCustomerInfo?custno=9

The response from CICS giving information about customer number 9 is shown in
Figure 2-20.

![Figure 2-20  Response from CICS](image-url)
Chapter 3. IBM Aspera fasp.io Gateway

The IBM Aspera® fasp.io Gateway is a lightweight TCP/IP tunnel for high-speed bidirectional data transport. It uses the Fast and Secure Protocol (FASP®) which is particularly efficient when transferring data over high latency networks, or those that have a tendency to loose packets.

From MQ Version 9.2.0 (Long Term Release), customers with entitlement to MQ Advanced for z/OS VUE are provided with free entitlement and access to the gateway for transferring MQ messages between MQ Advanced for z/OS VUE queue managers. Alternatively customers can purchase a separate entitlement to the gateway.

This chapter provides information on how to deploy the IBM Aspera fasp.io Gateway with IBM MQ on a zCX container. It includes:

3.1, “Introduction to Aspera FASP.io Gateway” on page 110

3.2, “Aspera configuration details” on page 111

3.3, “Integration with MQ Advanced for z/OS, VUE” on page 121
  – IBM Aspera fast.io Gateway introduction
  – Installing IBM Aspera fasp.io Gateway on zCX
  – Configuration Details
  – Integration with MQ on z/OS
3.1 Introduction to Aspera FASP.io Gateway

In this age of digital economy era, fast services and connectivity become an important factor to cloud and native applications today.

The IBM Aspera fasp.io Gateway is a software solution that provides significant improvements in performance and service quality when transferring data between highly remote or dispersed locations in unfavorable network conditions, such as high latency and packet loss. Fast and Secure Protocol (FASP) provides significant improvements in performance and service quality. MQ queue managers can take advantage of it and provide better service.

This technology was designed to deliver 100 percent bandwidth efficient transport of bulk data over any IP network. The IBM Aspera fasp.io Gateway uses an efficient algorithm for fast data transfer to retransmit only needed data, obtaining good performance independent of network delay or packet loss. It achieves speeds up to hundreds of times faster than FTP and HTTP that enables maximum speed without network saturation.

To ensure exceptional security for business-critical digital assets, Aspera uses open standards cryptography for user authentication, data encryption and data-integrity verification.

The IBM Aspera File Transfer Calculator can be used to estimate the gains of using Aspera versus TCP. The calculator is available at the following link:


The IBM Aspera fasp.io Gateway delivers the following key features and capabilities

- Extraordinary bandwidth control
  The intelligent adaptive transmission-rate control mechanism in the FASP protocol enables fast, automatic discovery of available bandwidth to fully utilize capacity while being fair to other traffic.

- Flexible and open architecture
  Aspera supports interoperable file and directory transfers between all major operating systems and cloud platforms, and provides a complete, modern software API.

- Software-only design
  Aspera is a software-based technology that can run on commodity hardware and over standard, unmodified IP networks, making for easier, more scalable deployments and lower total cost of ownership.

The Aspera Gateway component can be integrated quickly and easily with existing applications that use a TCP connection for data flow. By achieving per-process aggregate bandwidths as high as 2.5 GBPS, regardless of distance and network conditions, Aspera fasp.io Gateway outperforms TCP-based data flows over wide-area networks that exhibit high round-trip times and high packet loss.

The Gateway uses standard UDP in the transport layer and achieves decoupled congestion and reliability control in the application layer through a theoretically optimal approach that retransmits precisely the real packet loss on the channel.

Figure 3-1 shows a case where a messaging service like MQ or Event Streams can be used with fasp.io.
3.2 Aspera configuration details

In our environment we defined MQ sender-receiver channels between two queue managers, MQZ1 and MQZ2, each running on separate z/OS LPARs - wtsc74 and wtsc75.

We used the information shown in Table 3-1 in our configuration.

Table 3-1  Lab configuration

<table>
<thead>
<tr>
<th>LPAR</th>
<th>Queue Manager</th>
<th>FQDN / IP Address</th>
<th>Queue Manager Listener Port</th>
<th>zCX Name</th>
<th>zCX IP Address</th>
<th>Fasp.IO container name</th>
<th>Fasp.IO port (UDP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wtsc74</td>
<td>MQZ1</td>
<td>wtsc74.pbm.ihost.com</td>
<td>1414</td>
<td>sc74cn04.pbm.ihost.com</td>
<td>129.40.23.71</td>
<td>faspio-app-sc74cn04</td>
<td>2414</td>
</tr>
<tr>
<td>wtsc75</td>
<td>MQZ2</td>
<td>wtsc75.pbm.ihost.com</td>
<td>2414</td>
<td>sc74cn09.pbm.ihost.com</td>
<td>129.40.23.76</td>
<td>faspio-app-sc74cn09</td>
<td>1414</td>
</tr>
</tbody>
</table>

As noted in Table 3-1, we preferred to expose different TCP/UDP ports on each zCX container. You may want to do the same or define your own ports.

Note: In our instructions, we appended the zCX name to the container name, for example:
- For the sc74cn04 zCX server, the fasp.io container name is faspio-app-sc74cn04
- For the sc74cn09 zCX server, the fasp.io container name is faspio-app-sc74cn09

We installed fasp.io in two parts:
- “Configuration part 1 - fasp.io on the first zCX (sc74cn09) instance.” on page 112

Note: The connection between the fasp.io gateways containers that use the FASP protocol is UDP (User Datagram Protocol), not TCP (Transmission Control Protocol).
3.2.1 Configuration part 1 - fasp.io on the first zCX (sc74cn09) instance.

In this configuration, we will obtain the Aspera Gateway from an IBM MQ installation. Note that use of the Aspera gateway provided with MQ is limited to IBM MQ messages unless the gateway is separately entitled. To use the IBM MQ provided Aspera gateway, you must have one or more of the following entitlements:

- IBM MQ Advanced for Multiplatforms
- IBM MQ Appliance
- IBM MQ Advanced for z/OS VUE

For IBM MQ Advanced for z/OS VUE entitlement, you can get the Aspera gateway from the Connector Pack component that is part of the SMP/E installation. The Connector Pack component is identified by the following:

- FMID: HAV9110
- COMPID: 5655AV100
- Component Name: IBM MQ Connector Pack for z/OS

**Note:** See the IBM MQ Advanced for z/OS VUE program directory PDF files for more information in the following link:
https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.pro.doc/q001040_.htm?view=kc#q001040___pdnozos

In this section, we outline the steps we took to configure our first zCX instance.

1. Once the SMP/E installation, which includes the Aspera gateway from the Connector Pack component, is complete the file CSQ8FSP1.tar.Z is available in the following USS directory: /usr/lpp/mqm/V9R1MX
2. Download the CSQ8FSP1.tar.Z as a file named CSQ8FSP1.tar to a directory of your choice.
3. On the sc74cn09 zCX server, create the /home/admin/faspio directory.
   ```bash
   mkdir -p /home/admin/faspio
   cd /home/admin/faspio
   ```
4. Upload the file CSQ8FSP1.tar into the /home/admin/faspio directory.
5. To extract the required installation files, perform the following steps:
   a. Create a new dummy container to decompress the files using the following command:
      ```bash
      docker run --name dummy -it --entrypoint /bin/bash -d fedora
      ```
   b. Copy the CSQ8FSP1.tar file to the dummy container:
      ```bash
      docker cp CSQ8FSP1.tar dummy:.
      ```
   c. Enter the following command in the dummy container:
      ```bash
      docker exec -it dummy bash
      ```
   d. Execute the following commands to extract the fasp.io binary and config files:
      ```bash
      yum install -y unzip
tar xf CSQ8FSP1.tar
cd fasp
unzip ASP_FASP.IO_GW_2.5GBPSV1.0.1_ZLNX.zip
      ```
e. Install `ibm-fasp.io-gateway` by using the following command:
   
rpm -ivh ibm-fasp.io-gateway-1.0.1-1.s390x.rpm

f. Type `exit` to leave the dummy container

6. Use the following commands to copy the `/usr/bin/fasp.io-gateway` file and `/etc/fasp.io/` directory. They will be copied to the new fasp container.
   
docker cp dummy:/usr/bin/fasp.io-gateway
   docker cp dummy:/etc/fasp.io/ .

7. Create the file, `Dockerfile.faspio`, with the content shown in Example 3-1.

   **Example 3-1   Dockerfile.faspio**

   ```
   FROM fedora:latest
   
   # ENV
   ENV PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
   
   # Copy Entrypoint file
   COPY docker-entrypoint.sh /bin
   
   # The author
   LABEL maintainer="IBM Redbooks - Sample Dockerfile"
   
   # Installing Linux Packages
   RUN yum install -y envsubst telnet net-tools vim
   
   # Copying fasp.io files
   RUN mkdir /etc/fasp.io/
   COPY fasp.io-gateway /bin/
   COPY fasp.io/* /etc/fasp.io/
   
   # Start Fasp.io
   CMD ["fasp.io-gateway","start-foreground"]
   
   # Entrypoint Fasp.io
   ENTRYPOINT ["fasp.io-gateway"]
   ```

   **Note:** We added telnet, net-tools and vim packages into the container because they provide base networking utilities for Linux and allow us to query information about the services.

8. Build the new faspio container image using the following command:
   
docker build -t faspio-img-new -f Dockerfile.faspio .

9. Once the image is built, issue the following command to create a new fasp.io container:
   
docker run --name faspio-app-sc74cn09 -p2414:2414 -p1414:1414/udp -d faspio-img-new
Table 3-2 shows the options we used along with their descriptions.

**Table 3-2  Docker run options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>faspio-app-sc74cn09</td>
<td>Name of the fasp.io container. We appended the zCX name (sc74cn09) to make it easier to identify the container on each zCX instance.</td>
</tr>
<tr>
<td>-p2414:2414</td>
<td>Expose 2414 TCP port</td>
</tr>
<tr>
<td>-p1414:1414/udp</td>
<td>Expose 1414 UDP port</td>
</tr>
<tr>
<td>-d</td>
<td>Run container in background and print container ID</td>
</tr>
<tr>
<td>faspio-img-new</td>
<td>Name of the fasp.io Docker image</td>
</tr>
</tbody>
</table>

Architecturally, an Aspera gateway, configured to use the FASP protocol, sends and receives UDP traffic on the fasp port. To allow the UDP session to start, the fasp.io container on the zCX server side must expose the port UDP. During our environment creation, we neglected to specify /udp when exposing the FASP port and FASP gateway and received the error shown in Example 3-2.

**Example 3-2  Output of Docker logs command**

<table>
<thead>
<tr>
<th>Time stamp</th>
<th>Service</th>
<th>Level</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020-06-15 22:10:29</td>
<td>gateway</td>
<td>error</td>
<td>2:Session error: Connection refused</td>
</tr>
<tr>
<td>2020-06-15 22:10:29</td>
<td>s2s</td>
<td>info</td>
<td>2:Stats downstream_bytes=0, upstream_bytes=0</td>
</tr>
<tr>
<td>2020-06-15 22:11:30</td>
<td>gateway</td>
<td>error</td>
<td>3:Session error: Connection refused</td>
</tr>
<tr>
<td>2020-06-15 22:11:30</td>
<td>s2s</td>
<td>info</td>
<td>3:Stats downstream_bytes=0, upstream_bytes=0</td>
</tr>
<tr>
<td>2020-06-15 22:12:32</td>
<td>gateway</td>
<td>error</td>
<td>4:Session error: Connection refused</td>
</tr>
<tr>
<td>2020-06-15 22:12:32</td>
<td>s2s</td>
<td>info</td>
<td>4:Stats downstream_bytes=0, upstream_bytes=0</td>
</tr>
</tbody>
</table>

**Note:** The FASP protocol uses UDP (not TCP). Remember to specify *udp* when exposing the FASP port. If your zCX instance is behind a firewall, remember to open the firewall ports and provide the necessary information to the firewall team.
The communication between our zCX instances did not pass through a firewall. However, ensure you allow TCP and UDP traffic between your zCX instances if they are behind a firewall device.

10. Stop the container

   docker stop faspio-app-sc74cn09

11. Create the file, `gateway.toml` with the contents shown in Example 3-3.

   **Example 3-3  gateway.toml file contents**
   
   ```
   [[bridge]]
   name = "MQZ1_MQZ2"
   [bridge.local]
   protocol = "tcp"
   host = "0.0.0.0"
   port = 2414
   [bridge.forward]
   protocol = "fasp"
   host = "129.40.23.71"
   port = 2414
   
   [[bridge]]
   name = "MQZ2_MQZ1"
   [bridge.local]
   protocol = "fasp"
   host = "0.0.0.0"
   port = 1414
   [bridge.forward]
   protocol = "tcp"
   host = "wtsc74.pbm.ihost.com"
   port = 1414
   ```

12. Copy the file `gateway.toml` into the container with the following command:

   ```
   docker cp gateway.toml faspio-app-sc74cn09:/etc/fasp.io/gateway.toml
   ```

   If you need to enable fasp.io in debug mode for troubleshooting, create the `logging.toml` file with the content shown in Example 3-4.

   **Example 3-4  Logfile for fasp.io**
   
   ```
   # Available Loggers:
   #
   # bridge: High-level logger for the bridge
   # s2s: Stream-2-Stream session class logger
   # fasp.io-cpp: Logger for the Asio/C++ FASP SDK
   # fasp.io-c: Logger for the FASP protocol
   #
   # For more on how to configure logging, see full reference at:
   #     https://github.com/guangie88/spdlog_setup
   #
   # level is optional for both sinks and loggers
   # level for error logging is 'err', not 'error'
   # _st => single threaded, _mt => multi threaded
   ```
# syslog_sink is automatically thread-safe by default, no need for _mt suffix

# check out https://github.com/gabime/spdlog/wiki/3.-Custom-formatting
global_pattern = "%+"

# Async
[global_thread_pool]
queue_size = 8192
num_threads = 1

[[sink]]
  name = "console"
  type = "color_stdout_sink_mt"

[[logger]]
  name = "bridge"
  type = "async"
  sinks = ["console"]
  level = "debug"

[[logger]]
  name = "s2s"
  type = "async"
  sinks = ["console"]
  level = "debug"

[[logger]]
  name = "fasp.io-cpp"
  type = "async"
  sinks = ["console"]
  level = "debug"

[[logger]]
  name = "fasp.io-c"
  type = "async"
  sinks = ["console"]
  level = "debug"

13. Copy the `logging.toml` file into the container with the following command:

   `docker cp logging.toml faspio-app-sc74cn09:/etc/fasp.io/logging.toml`

   **Note:** The logfile generates a high number of messages, only use this if it is really necessary.

14. Start the new container with the following command

   `docker start faspio-app-sc74cn09`

   Use the following steps to confirm that fasp.io is up.

   1. Check the container log to ensure there are no issues by using the following command:

      `docker logs faspio-app-sc74cn09`

      The expected output is shown in Example 3-5.
Example 3-5  Container log file output

Loading gateway config file: /etc/fasp.io/gateway.toml
Loading logging config file: /etc/fasp.io/logging.toml
[2020-06-16 19:21:16.027] [gateway] [info] Gateway version: 1.0.1 5dfd35dde57c7ec251a2fae443f7b05f3e4922b1
[2020-06-16 19:21:16.027] [gateway] [info] Gateway config : /etc/fasp.io/gateway.toml
[2020-06-16 19:21:16.027] [gateway] [info] Logging config : /etc/fasp.io/logging.toml
[2020-06-16 19:21:16.027] [gateway] [info] MQZ1_MQZ2: 0.0.0.0:2414  tcp -> fasp  [129.40.23.71]:2414
[2020-06-16 19:21:16.033] [gateway] [info] MQZ2_MQZ1: 0.0.0.0:1414  fasp -> tcp [129.40.23.1]:1414

2. Check for any exposed container ports with the following command:

   docker port faspio-app-sc74cn09

   Expected output is shown in Example 3-6.

Example 3-6  List specific mapping for the container

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2414/tcp</td>
<td>0.0.0.0:2414</td>
<td></td>
</tr>
<tr>
<td>1414/udp</td>
<td>0.0.0.0:1414</td>
<td></td>
</tr>
</tbody>
</table>

3. Confirm fasp.io ports are opened with the following command:

   docker exec -it faspio-app-sc74cn09 bash -c "netstat -an| grep 0.0.0.0:*

   The expected output is shown in Example 3-7.

Example 3-7  Confirm fasp.io ports are opened

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp</td>
<td>0</td>
<td>0.0.0.0:2414</td>
</tr>
<tr>
<td>udp</td>
<td>0</td>
<td>0.0.0.0:1414</td>
</tr>
</tbody>
</table>

3.2.2  Configuration part 2 - fasp.io on the second zCX (sc74cn04) instance.

In this section, we outline the steps we took to configure our second zCX instance.

1. Repeat step 1 on page 112 through step 8 on page 113 in entirety.

2. Once the image is built, issue the following command to create a new fasp.io container:

   docker run --name faspio-app-sc74cn04 -p1414:1414 -p2414:2414/udp -d faspio-img-new

Note: The FASP protocol uses UDP (not TCP). Remember to specify udp when exposing the FASP port. If your zCX instance is behind a firewall, remember to open the firewall ports and provide the necessary information to the firewall team.

Table 3-3 shows the options we used along with their descriptions for the second zCX instance.
3. Stop the container
   
   ```
docker stop faspio-app-sc74cn04
   ```

4. Create the file, `gateway.toml` with the contents shown in Example 3-8.

   **Example 3-8**  gateway.toml file contents

   ```
   [[bridge]]
   name = "MQZ2_MQZ1"
   [bridge.local]
   protocol = "fasp"
   host = "0.0.0.0"
   port = 2414
   [bridge.forward]
   protocol = "tcp"
   host = "wtsc75.pbm.ihost.com"
   port = 2414
   
   [[bridge]]
   name = "MQZ1_MQZ2"
   [bridge.local]
   protocol = "tcp"
   host = "0.0.0.0"
   port = 1414
   [bridge.forward]
   protocol = "fasp"
   host = "129.40.23.76"
   port = 1414
   ```

5. Copy the file `gateway.toml` into the container with the following command:

   ```
docker cp gateway.toml faspio-app-sc74cn04:/etc/fasp.io/gateway.toml
   ```

   If you need to enable fasp.io in debug mode for troubleshooting, create the `logging.toml` file with the content shown in Example 3-9.

   **Example 3-9**  Logfile for fasp.io

   ```
   ###########################
   ### LOGGING SETTINGS    
   ###########################
   
   # Available Loggers:
   #
   # bridge: High-level logger for the bridge
   ```
# s2s: Stream-2-Stream session class logger
# fasp.io-cpp: Logger for the Asio/C++ FASP SDK
# fasp.io-c: Logger for the FASP protocol

# For more on how to configure logging, see full reference at:
https://github.com/guangie88/spdlog_setup

# level is optional for both sinks and loggers
# level for error logging is 'err', not 'error'
# _st => single threaded, _mt => multi threaded
# syslog_sink is automatically thread-safe by default, no need for _mt suffix

# check out https://github.com/gabime/spdlog/wiki/3.-Custom-formatting
global_pattern = "%+"

# Async
[global_thread_pool]
queue_size = 8192
num_threads = 1

[[sink]]
  name = "console"
  type = "color_stdout_sink_mt"

[[logger]]
  name = "bridge"
  type = "async"
  sinks = ["console"]
  level = "debug"

[[logger]]
  name = "s2s"
  type = "async"
  sinks = ["console"]
  level = "debug"

[[logger]]
  name = "fasp.io-cpp"
  type = "async"
  sinks = ["console"]
  level = "debug"

[[logger]]
  name = "fasp.io-c"
  type = "async"
  sinks = ["console"]
  level = "debug"

6. Copy the logging.toml file into the container with the following command:
   
   docker cp logging.toml faspio-app-sc74cn04:/etc/fasp.io/logging.toml

   
   **Note:** The logfile generates a high number of messages, only use this if it is really necessary.
7. Start the new container with the following command

   `docker start faspio-app-sc74cn04`

Use the following steps to confirm that fasp.io is up.

1. Check the container log to ensure there are no issues by using the following command:

   `docker logs faspio-app-sc74cn04`

   The expected output is shown in Example 3-10.

   **Example 3-10  Container log file output**

   Loading gateway config file: /etc/fasp.io/gateway.toml
   Loading logging config file: /etc/fasp.io/logging.toml
   [2020-06-16 19:22:46.113] [gateway] [info] Gateway version: 1.0.1
   5dfd35dd657c7ec2512f9e443f7b05f3e4922b1
   [2020-06-16 19:22:46.113] [gateway] [info] Gateway config :
   /etc/fasp.io/gateway.toml
   [2020-06-16 19:22:46.113] [gateway] [info] Logging config :
   /etc/fasp.io/logging.toml
   [2020-06-16 19:22:46.116] [gateway] [info] MQZ1_MQZ2: 0.0.0.0:2414  fasp -> tcp
   [129.40.23.3]:2414
   [2020-06-16 19:22:46.117] [fasp.io-c] [debug] 1:setopt port_reuse:1
   [2020-06-16 19:22:46.117] [fasp.io-c] [debug] 1:setopt role:server(1)
   [2020-06-16 19:22:46.117] [fasp.io-c] [debug] 1:dgram sock snd buffer requested
   10485760 bytes, received 524288 bytes
   [2020-06-16 19:22:46.117] [fasp.io-c] [debug] 1:dgram sock rcv buffer requested
   10485760 bytes, received 524288 bytes
   [2020-06-16 19:22:46.117] [fasp.io-c] [debug] 1:bound [0.0.0.0:2414]
   [2020-06-16 19:22:46.117] [fasp.io-c] [debug] 1:detectd timer level: 1
   [2020-06-16 19:22:46.117] [fasp.io-c] [info] 1:FASP Listening [0.0.0.0:2414]
   [2020-06-16 19:22:46.117] [gateway] [info] MQZ2_MQZ1: 0.0.0.0:1414  tcp ->
   fasp  [129.40.23.76]:1414

2. Check for any exposed container ports with the following command:

   `docker port faspio-app-sc74cn04`

   Expected output is shown in Example 3-11.

   **Example 3-11  List specific mapping for the container**

   1414/tcp -> 0.0.0.0:1414
   2414/udp -> 0.0.0.0:2414

3. Confirm fasp.io ports are opened with the following command:

   `docker exec -it faspio-app-sc74cn04 bash -c "netstat -an | grep 0.0.0.0:*"`

   The expected output is shown in Example 3-12.

   **Example 3-12  Confirm fasp.io ports are opened**

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Local Address</th>
<th>Remote Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcp</td>
<td>0 0.0.0.0:1414</td>
<td>0.0.0.0:*</td>
<td>LISTEN</td>
</tr>
<tr>
<td>udp</td>
<td>0 0.0.0.0:2414</td>
<td>0.0.0.0:*</td>
<td></td>
</tr>
</tbody>
</table>

The fasp.io gateways containers are now running without any problem. You should be able to now integrate with z/OS queue managers.
3.3 Integration with MQ Advanced for z/OS, VUE

This section covers the use of fasp.io with MQ Advanced for z/OS, VUE. The assumption is that the reader has a basic knowledge of MQ Advanced for z/OS, VUE.

3.3.1 Topology

In this section, we will build a topology to communicate between two z/OS queue managers using the Aspera fasp.io gateway. Our topology is shown in the Figure 3-2.

![Figure 3-2 MQ and Aspera topology](image)

In this topology, we have the following:
1. MQ V9.1.5 queue manager on SC74 LPAR: MQZ1
2. MQ listener port on MQZ1: 1414
3. FASP.IO gateway running on zCX instance on SC74:
   a. TCP Port: 2414
   b. UDP Port: 1414
4. FASP.IO gateway running on zCX instance on SC75
   a. TCP Port: 1414
   b. UDP Port: 2414
5. MQ V9.1.5 queue manager on SC75 LPAR: MQZ2
6. MQ listener port on MQZ2: 2414

3.3.2 Create transmission queue on MQZ1

Use the MQSC command to define the transmission queue. We will create the transmission queue on queue manager MQZ1 with the name, MQZ2, as shown in Example 3-13.

Example 3-13   Transmission queue on MQZ1

```
DEFINE QLOCAL(MQZ2) USAGE(XMITQ)
```
3.3.3 Create sender channel on MQZ1

Use the MQSC command to define the channel. We will create the sender channel on queue manager MQZ1 with the name, MQZ1_MQZ2_0001, as shown in Example 3-14.

Example 3-14  Sender channel definitions on MQZ1

```
DEF CHL(MQZ1_MQZ2_0001) CHLTYPE(SDR) +
   TRPTYPE(TCP) DISCINT(0) +
   BATCHSZ(50) BATCHINT(1000) +
   NPMSEED(FAST) +
   XMITQ(MQZ2) +
   MAXMSG(2097152) +
   CONNAME('129.40.23.76(2414)') +
   MONCHL(QMGR) +
   COMPMSG(NONE) +
   REPLACE
```

3.3.4 Create receiver channel on MQZ2

Use the MQSC command to define the channel. We will create the receiver channel on MQZ2, as shown in Example 3-15, with the following name: MQZ1_MQZ2_0001.

Example 3-15  Receiver channel definitions on MQZ2

```
DEFINE CHANNEL(MQZ1_MQZ2_0001) CHLTYPE(RCVR) TRPTYPE(TCP) REPLACE
```

3.3.5 Create transmission queue on MQZ2

Use the MQSC command to define the transmission queue. We will create the transmission queue on queue manager MQZ2 with the name, MQZ1, as shown in Example 3-16.

Example 3-16  Transmission queue on MQZ2

```
DEFINE QLOCAL(MQZ1) USAGE(XMITQ)
```

3.3.6 Create sender channel on MQZ2

We will create the sender channel on queue manager MQZ2 as shown in Example 3-17.

Example 3-17  Sender channel definitions on MQZ2

```
DEF CHL(MQZ2_MQZ1_0001) CHLTYPE(SDR) +
   TRPTYPE(TCP) DISCINT(0) +
   BATCHSZ(50) BATCHINT(1000) +
   NPMSEED(FAST) +
   XMITQ(MQZ2) +
   MAXMSG(2097152) +
   CONNAME('129.40.23.71(1414)') +
   MONCHL(QMGR) +
   COMPMSG(NONE) +
   REPLACE
```
3.3.7 Create receiver channel on MQZ1

Use the MQSC command to define the channel. We will create the receiver channel on MQZ2 with the following name: MQZ2_MQZ1_0001, as shown in Example 3-18.

Example 3-18  Receiver channel definitions on MQZ2

```
DEFINE CHANNEL(MQZ2_MQZ1_0001) CHLTYPE(RCVR) TRPTYPE(TCP) REPLACE
```

3.3.8 Verify the infrastructure and configuration

In this section, we create the necessary objects on both queue managers to verify two-way communication.

Create objects on MQZ1

We created a local queue and a remote queue shown in Table 3-4 on queue manager MQZ1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Remote Queue</th>
<th>Remote QMGR</th>
<th>XMIT Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZ1.FASP.MQZ2.QREMOTE</td>
<td>Remote Queue</td>
<td>MQZ1.FASP.MQZ2.QLOCAL</td>
<td>MQZ2</td>
<td>MQZ2</td>
</tr>
<tr>
<td>MQZ2.FASP.MQZ1.QLOCAL</td>
<td>Local Queue</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Create objects on MQZ2

We created a local queue and a remote queue shown in Table 3-5 on queue manager MQZ2.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Remote Queue</th>
<th>Remote QMGR</th>
<th>XMIT Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQZ2.FASP.MQZ1.QREMOTE</td>
<td>Remote Queue</td>
<td>MQZ2.FASP.MQZ1.QLOCAL</td>
<td>MQZ1</td>
<td>MQZ1</td>
</tr>
<tr>
<td>MQZ1.FASP.MQZ2.QLOCAL</td>
<td>Local Queue</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.3.9 Verify two way communication

We are going to use the IBM MQ Explorer tool to PUT and GET messages.

1. Test the communication from MQZ1 to MQZ2. Open MQ Explorer and select the remote queue, MQZ1.FASP.MQZ2.QREMOTE as shown in Figure 3-3.
2. PUT a test message on the selected remote queue shown in Figure 3-3. Figure 3-4 shows the queue, `MQZ1.FASP.MQZ2.QREMOTE`.

3. If our setup is configured correctly, the message should arrive on the local queue named `MQZ1.FASP.MQZ2.QLOCAL` on MQZ2. Select the local queue `MQZ1.FASP.MQZ2.QLOCAL` on MQZ2 and Browse Messages as shown in Figure 3-5.
We see that the message is delivered to the local queue as shown in Figure 3-6.

This verifies that our queue managers and FASP.IO gateways are configured correctly for MQZ1 to MQZ2 using the FASP protocol.

Next, we test the reverse communication from MQZ2 to MQZ1. We are going to PUT a message on the remote queue `MQZ2.FASP.MQZ1.QREMOTE` on MQZ2. To do this, follow these steps:
1. Navigate to the queue manager MQZ1 and Browse Messages on the local queue, MQZ2.FASP.MQZ1.QLOCAL. We will see that a message has been delivered to the local queue as shown in Figure 3-7.

![Message on MQZ1](image)

This shows that we have two-way communication between the two z/OS queue managers using the FASP.IO gateway.

### 3.3.10 Benefits of running the FASP.IO gateway on zCX

By integrating the IBM Aspera FASP protocol, MQ can move messages at high speed over distance for many different real-time processes, for example, financial transactions. The value of having the gateway in zCX is that it makes it as close to the queue manager as possible which minimizes the amount of time TCP/IP is being used.

Integrating the IBM Aspera FASP protocol also simplifies scenarios where z/OS systems are IPLed in alternate locations, as the FASP.IO gateway function moves automatically with the z/OS system and no changes are needed to the MQ channel configuration to connect to an alternate FASP.IO gateway.
Using MQ on zCX as a client concentrator

zCX brings many benefits - but is particularly useful if you just need to run a small number of Linux on IBM Z applications that need to connect into your z/OS applications - there is no need for a dedicated Linux on IBM Z environment, or IFLs. zCX containers can run on zIIPs making them attractive from a pricing perspective.

With zCX comes the ability to connect remote applications directly into IBM MQ on the z/OS queue managers. Running the concentrator on the same z/OS LPAR as the target queue manager is both efficient and and can also exploit the EZA4CXR SAMEHOST interface for high speed communication between the queue managers.

This chapter provides an example use case scenario on how to use MQ on zCX as a client concentrator and includes:

- “Interconnecting MQ on z/OS with MQ in zCX” on page 128
- “Using IBM MQ on zCX as a client concentrator” on page 139
4.1 Interconnecting MQ on z/OS with MQ in zCX

This section describes an IBM MQ integration scenario based on a messaging infrastructure that uses IBM MQ on z/OS and zCX. We review how to build an IBM MQ container on zCX and create the necessary objects for the integration. A messaging-based solution establishes a shared integration layer, enabling the seamless flow of multiple types of data between heterogeneous systems.

Many customers have Systems of Engagement (SoE) applications running as cloud native application microservices inside a container. Systems of Record (SoR) applications run on z/OS with, for example, CICS, IMS and/or DB2 applications. IBM MQ can be used to connect these SoE applications running inside zCX with SoR applications running on z/OS. SoR applications using a request-reply pattern. This section explains how a queue manager running inside zCX can communicate with a queue manager running on z/OS.

Table 4-1 shows how the use of IBM MQ can meet the requirements of SoE and SoR integration.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ability to invoke CICS/IMS applications from zCX</td>
<td>IBM MQ provides channels that can be created and configured to send messages between applications using MQ queues.</td>
</tr>
<tr>
<td>High Availability</td>
<td>IBM MQ can be configured to use a 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.</td>
</tr>
<tr>
<td>High Scalability</td>
<td>The use of a QSG enables a scalable solution that takes advantage of the resources across the parallel sysplex. New instances of CICS regions, IMS regions, or queue managers can be easily introduced into the QSG as business growth dictates.</td>
</tr>
<tr>
<td>Workload Management</td>
<td>Both zCX and IBM MQ can be managed by the WLM.</td>
</tr>
</tbody>
</table>

4.1.1 Architecture

A zCX instance and the IBM MQ queue manager run as started tasks on z/OS. CICS and IMS are also started tasks on z/OS. The IBM z/OS Communication Server is used to establish communication between MQ running inside the container and MQ running on z/OS. Figure 4-1 shows how applications inside zCX communicate with other z/OS applications via the Communication Server.
4.1.2 Scenario

In this section, we discuss an MQ integration scenario where two queue managers communicate with each other via messages on a queue. For simplicity, we will not be building 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 running inside a zCX container can also act as either consumers and/or producers. In our example, we will not build the consumer and producer applications. We will use the tools provided by MQ to put and get messages from MQ queues.

Figure 4-2 shows the infrastructure to support the communication between the queue managers in our environment.
4.1.3 Download an IBM MQ image inside zCX from dockerhub

To download the IBM MQ Advanced image from the dockerhub repository, perform the following steps:

1. Navigate to the dockerhub IBM MQ Advanced web page with the following link. https://hub.docker.com/_/ibm-mq-advanced
   The display will appear as shown Figure 4-3.

   ![IBM MQ Advanced Image on dockerhub](image)

   Figure 4-3  IBM MQ Advanced Image on dockerhub

   As you scroll down through this page, you will see listings for maintained and unmaintained images.

2. Copy the docker pull command as shown in Example 4-1. This example pulls version 9.1.0.0 which is specified as a tag in the command.
Example 4-1   Pull IBM MQ from dockerhub

docker pull store/ibmcorp/mqadvanced-server-dev:9.1.0.0

3. Connect to your previously provisioned zCX instance via your SSH client
   a. Log on to USS with an SSH client tool, such as PuTTY, with the userid you used to
      create an SSH key. In this example, we created the SSH keys with the ZCXADM5
      userid. So, we logged on with the ZCXADM5 userid.
   b. Use the command shown in Example 4-2 to connect 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 4-2   Log in as admin to zCX

ssh admin@129.40.23.76 -p 8022

After successfully logging in, the screen looks as shown Figure 4-4

Figure 4-4   Successful log in

4. Log in to dockerhub with your Docker credentials with the command shown in
   Example 4-3

   Example 4-3   Login to docker

docker login

5. Pull the IBM MQ Advanced image with the command shown in Example 4-4

   Example 4-4   Docker pull command

docker pull store/ibmcorp/mqadvanced-server-dev:9.1.0.0

Figure 4-5 shows the expected output from logging in and running the pull command.
In this section, we provide the Docker commands used to create a Docker volume and the queue manager named QMZCX which runs as a container inside the zCX instance.

1. Create a Docker volume

A new Docker volume must first be created to ensure data persistence. For this example, a volume with the name `mqdataqmzcx` is created by issuing the following command:

```bash
docker volume create mqdataqmzcx
```

The `docker` command, shown in Example 4-5, creates a new MQ Queue Manager named QMZCX that is listening on port 1415 with the Web listener on port 9444 for the MQ Web console. We are also updating to the more current version of IBM MQ, V9.1.2.0, by using a tag for the version.

**Example 4-5  Create Queue Manager on zCX**

```bash
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
```
2. Verify that the docker container has been created and is running by executing the docker command shown in Example 4-6.

**Example 4-6  Display docker process**

```bash
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
```

3. Verify that the Queue Manager is created by logging in to the MQ Web Console. Point a web browser to the MQ Web Console with the URL https://129.40.23.76:9444 where 129.40.23.76 is the IP address assigned to the provisioned zCX instance. Use the default credentials of username=admin and password=passw0rd to log in to the MQ Web Console. You can change these defaults.

Once you have logged in to the web console, you will be presented with a page similar to that shown in Figure 4-6.

![Figure 4-6  MQ Web Console](image)

### 4.1.5 Create MQ channels between the 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 will create message channels between the queue managers. A message channel connects two queue managers. These channels are unidirectional, in that messages are only sent in one direction along the channel. There are subtypes of message channel, with the
different types distinguished by whether they are used for receiving or sending messages, whether they initiate the communication to the partner queue manager, and whether they are used within a cluster or not.

A channel definition on one queue manager has to be matched with a channel definition of the same name with an appropriate subtype on the partner queue manager.

We will define a sender channel on one queue manager with an identically named receiver channel on a second queue manager. We will also create the appropriate transmission queues for the channels. We will not focus on the MQ commands for creating the channels, assuming that the user knows how to create those.

1. Create Message Channels on QMZCX

Table 4-2 lists the channels to create on the queue manager named QMZCX.

<table>
<thead>
<tr>
<th>Channel Name</th>
<th>Type</th>
<th>Conn Name</th>
<th>Transmission Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMZCX.MQZ1</td>
<td>Sender</td>
<td>wtsc74.pbm.ihost.com(1414)</td>
<td>MQZ1</td>
</tr>
<tr>
<td>MQZ1.QMZCX</td>
<td>Receiver</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

Note: The connection named wtsc74.pbm.ihost.com is the z/OS LPAR where the queue manager named MQZ1 is running and listening on port 1414. We also created a transmission queue that is a new local queue named MQZ1 with the usage type of XMITQ.

2. Create message channels on z/OS Queue Manager MQZ1

Table 4-3 lists the channels created on queue manager MQZ1 on z/OS.

<table>
<thead>
<tr>
<th>Channel Name</th>
<th>Type</th>
<th>Conn Name</th>
<th>Transmission Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>QMZCX.MQZ1</td>
<td>Receiver</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>MQZ1.QMZCX</td>
<td>Sender</td>
<td>129.40.23.76(1415)</td>
<td>QMZCX</td>
</tr>
</tbody>
</table>

Note: The connection named 129.40.23.76 is the IP address of the provisioned zCX instance. 1415 is the port that the queue manager, QMZCX, will be use for listening inside the container. We also created a transmission queue that is a new local queue named QMZCX that was created on the queue manager named MQZ1 with the usage type of XMITQ.

This completes the creation of the message channels for queue manager communication.

3. Start sender channels

Channels are usually configured to start automatically when messages are available on the associated transmission queue. We will start the sender channels manually to make sure that all the channels are running. We will start the sender channels on the queue manager named QMZCX (in the container) and MQZ1 (on z/OS) by using the following steps.

a. Log in to the MQ web console, as shown in Figure 4-7. Select the sender channel named QMZCX.MQZ. Start the channel and ensure that it is running.
b. We will start the sender channel on MQZ1 from the MQ Explorer. Select the sender channel named MQZ1.QMZCX and start the channel to ensure that it is running. In our case, the channel did not start and we received an error. To diagnose the error and fix it, we performed the following steps.

i. 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, use the command shown in Example 4-7 to ping the channel.

Example 4-7   Ping Channel

```
-MQZ1 ping chl(MQZ1.QMZCX )
```

ii. The error shown in Example 4-8 is returned from the ping command. The error indicates that the remote channel is not available. This could be because the remote queue manager cannot be reached due to security issues.

Example 4-8   Output of Ping Channel

```
-CQM134I -MQZ1 CSQMPCHL PING CHL(MQZ1.QMZCX) COMMAND ACCEPTED
CSQX558E -MQZ1 CSQXPING Remote channel MQZ1.QMZCX not available
CSQ9023E -MQZ1 CSQXCRPS 'PING CHL' ABNORMAL COMPLETION
```

To inspect the security settings on the QMZCX container queue manager, we preformed the following steps.

i. Find the process ID of the container by running the command shown in Example 4-9.

Example 4-9   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
```
Getting started with z/OS Container Extensions

ii. Run the command shown in Example 4-10 to go inside the container to run MQ commands.

   **Example 4-10  Docker exec command**

```
admin@3ee48347653a:~$ docker exec -it 0c09eade8125 bash
```

iii. Display the attributes of queue manager QMZCX by using the `runmqsc` command to run IBM MQ commands on the queue manager and then the MQSC `display qmgr all` command, as shown in Example 4-11.

   **Example 4-11  Display Queue Manager QMZCX**

```
(mq:9.1.2.0)mqm@0c09eade8125:/$ runmqsc
```

```
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( )               CHAEXIT( )
CHLAUTH(ENABLED)          CHLEV(DISABLED)
CHLADATA( )               CMDEV(DISABLED)
CHLEXIT( )                CMDLEVEL(912)
CHLWLEN(100)              COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)
CHLWLRUC(999999999)       CONFIGEV(DISABLED)
CHLWUSEQ(LOCAL)           CONNAUTH(DEV.AUTHINFO)
CMDATE(2019-09-10)        CRDATE(2019-09-10)
```

iv. We see that **CHLAUTH** is **ENABLED** as a default value and **CONNAUTH** is also set to the default value. For simplicity, we will disable **CHLAUTH** and allow **CONNAUTH** to use the ID of the connecting system. We will use the MQSC `ALTER QMGR` command to do that, as shown in Example 4-12.

   **Example 4-12  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(OFF)
```

```
CONNAUTH(DEV.AUTHINFO)
```

```
qmname(QMZCX)
```

```
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)
AMQ80051: IBM MQ queue manager changed.

v. In MQ Explorer, select the sender channel MQZ1.QMZCX and start the channel as shown in Figure 4-8. It should go into a running state.

![Image of MQ Explorer showing the start of the sender channel MQZ1](image)

### 4.1.6 Test the message flow from both directions

We started the sender-receiver channel pairs on both queue managers in the previous step. We will now test the infrastructure by putting messages in a queue on one queue manager and receiving the messages on the destination queue.

We have created the queues shown in Table 4-4 for sending messages from z/OS to zCX.

<table>
<thead>
<tr>
<th>Table 4-4 From z/OS to zCX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote Queue on z/OS (MQZ1)</td>
</tr>
<tr>
<td>MQZ1.TO.QMZCX.REMOTE</td>
</tr>
</tbody>
</table>

We have created the queues shown in Table 4-5 for sending messages from zCX to z/OS.

<table>
<thead>
<tr>
<th>Table 4-5 From zCX to z/OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote Queue on zCX Container (QMZCX)</td>
</tr>
<tr>
<td>QMZCX.TO.MQZ1.REMOTE</td>
</tr>
</tbody>
</table>

To send messages from z/OS to zCX, perform the following steps.
1. In MQ Explorer, select the remote queue MQZQ.TO.QMZCX and put a test message on the queue as shown in Figure 4-9.

![Figure 4-9 Put Test Message on QMZCX](image)

2. Enter the test message, as shown in Figure 4-10, and click the **Put message** button.

![Figure 4-10 Put Message content](image)

3. Point to the MQ Web Console running the MQ container. Select the queue named MQZ1.TO.QMZCX.LOCAL and browse the queue as shown in Figure 4-11.

![Figure 4-11 Browse messages on QMZCX](image)
The message browsed is displayed in Figure 4-12.

![Figure 4-12 Content of browsed message](image)

4. Sending messages from zCX to z/OS

Similar steps can be performed by putting a message on the remote queue named `QMZCX.TO.MQZ1.REMOTE` on QMZCX that is running inside the container. The message can be browsed using MQ Explorer on the local queue named `QMZCX.TO.MQZ1.LOCAL` on queue manager MQZ1.r

4.2 Using IBM MQ on zCX as a client concentrator

We have built a queue manager on zCX and setup bi-directional connectivity with a z/OS queue manager. We now describe how this queue manager on zCX can be used in an enterprise.

Large numbers of client applications connected directly to an IBM MQ for z/OS queue manager can cause significant CPU load on the channel initiator address space, and therefore increase MLC costs. One way of reducing these costs is to use a client concentrator model where client connections are made to a proxy distributed queue manager (the client concentrator) which then routes messages to and from one or more z/OS queue managers over sender/receiver channels.

We will now demonstrate how a queue manager running in zCX can be used as a client concentrator.

Before we discuss the client concentrator in detail, let's look at the basics of an IBM MQ client.

4.2.1 IBM MQ Clients

The IBM MQ client is a lightweight component of IBM MQ that does not require the queue manager runtime code to reside on the client system. It enables an application, running on a machine where only the client runtime code is installed, to connect to a queue manager that is running on another machine and perform messaging operations with that queue manager. Such an application is called a client and the queue manager is referred to as a server. Using an IBM MQ client is an effective way of implementing IBM MQ messaging and queuing. The following benefits are available by using an IBM MQ client:
- There is no need for a licensed IBM WebSphere MQ server installation on the client machine.
- Hardware requirements on the client system are reduced.
- System administration requirements on the client system are reduced.
- An application that uses an IBM MQ client can connect to multiple queue managers on different machines.

Figure 4-13 shows how MQ clients connect with an IBM MQ server.

![ MQ Clients Diagram ]

A IBM MQ client has the following:
- IBM MQ client libraries installed.
- No queue managers or queues.
- Uses the same MQI calls as an IBM MQ server.
- Flow of messages in MQI channel is bidirectional.

### 4.2.2 MQ client concentrator

The advantages of using a client architecture is that there is no requirement to have servers defined and managed on all of the outlying machines. An enterprise may well have thousands of applications wishing to do messaging with just clients. The advantages of the MQ client is that it makes it easier for growing number of MQ client applications to connect to a z/OS queue manager directly. This is a simple and easy approach and the topology is similar to that shown in Figure 4-14.
Figure 4-14  Client applications connecting directly

However, direct connection to a z/OS queue manager introduces some challenges:

- Badly written MQ client applications can perform a lot of connects and disconnects.
- These frequent connects and disconnects use a very large amount of CPU in the CHINIT address space.

The CPU overhead on the z/OS address space means additional costs. In the client concentrator topology, an additional layer, the client concentrator layer, is introduced as shown in Figure 4-15.
The high level architecture of the client concentrator implementation is as follows:

1. The MQ client layer is comprised of applications that require MQ connectivity. These applications use the IBM MQ client libraries and are configured to connect to one or more queue managers in the MQ concentrator layer.

2. The client concentrator layer is where all the applications from the MQ client layer connect. This concentrator layer connects to the z/OS queue manager by using sender and receiver channels.

3. The MQ server layer on z/OS provides the messaging platform for the MQ applications.

The advantage of using a client concentrator is that it reduces the CHINIT CPU usage. The sender and receiver channels are less CPU intensive than the server-connection channels used by directly connected client apps.

We should keep in mind that the concentrator topology is not suitable for all applications. The concentrator layer introduces some latency and complexity and may not be suitable for applications that require high throughput. Additionally it can be a single point of failure so a high availability solution is typically needed.

### 4.2.3 MQ on zCX as a client concentrator

A queue manager running on zCX can be used as a client concentrator. Figure 4-16 shows the topology of how a queue manager running on zCX can be used as a client concentrator.
There are benefits of using this topology. Some of the benefits are:

- There is no need to provision an additional distributed server or an MQ appliance.
- As the queue manager on zCX and queue manager on z/OS are on the same z/OS LPAR, they use cross memory functionality to implement high-performance network communications.
Part 2

DevOps

In this part, we describe how organizations can benefit from running a DevOps flow in zCX and we describe the set up of necessary components.

This chapter includes the following topics of using DevOps in zCX:

- “DevOps Overview” on page 147
- “Using Gitea as a code repository” on page 151
- “Using Jenkins to automate builds” on page 163
- “Using Ansible to automate deployment and tests” on page 185
- “Putting it all together and run the pipeline” on page 193

Note: To follow along with the instructions in this part of the IBM Redbooks publication, a running zCX instance and an SSH log on to your zCX instance is required.
DevOps Overview

This chapter provides an overview of the DevOps field and describes the benefits of using DevOps within containers and zCX. In this chapter, we discuss the following:

- “What is DevOps?” on page 148
- “Why containerize DevOps?” on page 149
- “Why use DevOps in zCX?” on page 149
5.1 What is DevOps?

DevOps is a combination of the words development and operations. It covers a variety of methods based on the principle that understanding both the technical and the non-technical background is crucial to building a valuable solution.

DevOps is an increasingly common approach that has its roots in agile software development. It enables developers and operations teams to code, build, test and deploy applications with speed, quality and control. The core parts of a DevOps flow are shown in Figure 5-1.

![Figure 5-1 Core DevOps components](image)

DevOps puts the interaction between people over processes and tools. The core goal is to achieve a continuous integration and delivery (CI/CD) to create more room for interactions. Successful DevOps implementations generally rely on an integrated set of solutions or a “toolchain” to remove manual repetitive steps, reduce errors, increase team agility, and to scale beyond small, isolated teams.

The use of DevOps is regardless of architecture, platform or purpose. Common use cases include: cloud-native and mobile applications, application integration, modernization and multi-cloud management.

For more information, see:

5.2 Why containerize DevOps?

Although containers are not mandatory for DevOps, they make it a lot easier and provide some great benefits.

When it comes to resources, containers are best performers. They require fewer system resources than usual machine environments or virtual machines because they do not include operating system images and run directly on the host system.

Containers are real game changers with regards to consistency. While writing, testing and deploying applications inside containers, the environment does not change at different parts of the CI/CD process. That makes collaboration between different teams easier and brings them closer together because they are all working in the same environment.

In the CI/CD pipeline it’s important to roll out application updates continuously. Containers enable you to easily update your applications. When your application consists of, for example, multiple microservices, each one will run in a separate container and you can deploy your updates step by step without stopping the other parts of the application. Therefore, containers allow faster deployment, patching, or scaling of applications.

Using DevOps comes with the benefit of being independent from specific tools. You have the agility to switch between frameworks and platforms easily. Containers are independent from platforms and programming languages and enable you to run nearly every application on every platform. Running your DevOps cycle inside containers guarantees your hosting tools to be self-determined, regardless of third parties.

For more information, see:

https://containerjournal.com/uncategorized/containers-devops-anyway/

5.3 Why use DevOps in zCX?

Using DevOps within zCX offers the ability to combine powerful container technologies as well as agile development within the z/OS and IBM Z infrastructure. This enables you to take advantage of the benefits from all three worlds - DevOps, containers, and the high performance and secure IBM Z.

In the context of modernizing the mainframe, it is so important to run modern applications on a reliable system. Applications hosted in zCX also offer the possibility to dynamically extend your mainframe. An example would be a web application hosted on zCX that can deal as a communications entry point for a mobile application.

zCX enables you to bring existing z/OS enterprise systems, such as CICS and modern applications, to a platform where you can use them both together to benefit from their synergies.

**Note:** Even though DevOps comes with a large cache of tools, only Docker images with s390x architecture can be run in zCX.
Using Gitea as a code repository

Gitea is a self-hosted Git service. It is similar to common services such as GitHub, Bitbucket and GitLab. Gitea’s goal is to provide an easy and fast way to set up a self-hosted Git service for many platforms and architectures.

For more information, see:
https://docs.gitea.io/en-us/

Note: Using Gitea inside a container offers you independence from third parties and their availability. The use case described in this chapter is not dependent on Gitea. You can use any other Git service that provides webhooks. Gitea is our tool of choice in this case because the setup is fast and intuitive.

This chapter describes how to set up Gitea to provide an example of a Git service in zCX. The following topics are included:

Chapter 6.1, “Set up Gitea in zCX” on page 152
Chapter 6.1.1, “Build a Gitea Docker image” on page 152
Chapter 6.1.2, “Set up a private registry” on page 154
Chapter 6.2, “Running a Gitea Docker container” on page 155
Chapter 6.4, “Configure Gitea” on page 157
Chapter 6.4.1, “Upload Code to Gitea” on page 159
6.1 Set up Gitea in zCX

This section includes how to build and run a Gitea Docker container and use it as a code repository. We have provided code samples and instructions for you to follow along with your own set up. Instructions for downloadable materials can be found in Appendix A, “Additional material” on page 233.

6.1.1 Build a Gitea Docker image

Instructions to download the files (including the Dockerfile) required to build and run a Gitea container for the s390x architecture can be found in the Appendix A, “Additional material” on page 233 of this book. Once downloaded, perform the following steps:

1. Download the added material folder for DevOps and navigate from a command prompt to the Gitea folder. Run the following command to copy the file to your z/OS:

   ```
   scp gitea.tar zcxadm3@wtsc74.pbm.ihost.com:/tmp
   ```

   As shown in Example 6-1, you will see the process of copying in your console.

   **Example 6-1  Output of scp gitea.tar command**
   
   base:Gitea maike$ scp gitea.tar zcxadm3@wtsc74.pbm.ihost.com:/tmp
   zcxadm3@wtsc74.pbm.ihost.com's password: 
gitea.tar
   21% 45MB 135.6KB/s 20:17 ETA

2. After completion of the copy process your file is accessible in z/OS. Log in to your z/OS System using the following command:

   ```
   ssh -l zcxadm3  wtsc74.pbm.ihost.com
   ```

3. Navigate to the /tmp folder where the gitea.tar file was copied using the following command:

   ```
   cd /tmp
   ```

4. To verify the file was copied correctly, display the files containing gitea by using the `ls -l | grep gitea` command, which generates the output shown in Example 6-2.

   **Example 6-2  Output of ls -l | grep gitea command**
   
   ZCXADM3 @ SC74:/SC74/tmp> ls -l | grep gitea
   -rw-r--r--  1 ZCXADM3  SYS1   216485197 Jun 26 12:00 gitea.tar

5. To transfer the file securely, log in to your zCX instance via SFTP:

   ```
   sftp -P 8022 admin@129.40.23.72
   ```

   **Note:** In SFTP the `-P` option is reserved for the port.

   You will receive the output shown in Example 6-3 if the SFTP connection was established successfully.

   **Example 6-3  Output of sftp -P 8022 admin@129.40.23.72 command**
   
   ZCXADM3 @ SC74:/SC74/tmp>sftp -P 8022 admin@129.40.23.72
6. To ensure the right file format when copying files to another system, set file encoding to ASCII by using the following command:
```
ascii unix
```

7. Then copy the gitea.tar file from z/OS to your zCX instance by using the following command:
```
put gitea.tar
```
As shown in Example 6-4, you will see the progress in your console.

```
Example 6-4  Output of put gitea.tar command
sftp>put gitea.tar
Uploading gitea.tar to /home/admin/gitea.tar
  gitea.tar
100% 206MB 103.2MB/s   00:02
```

At this point you will have the gitea.tar file stored in your zCX instance and can issue the `exit` command to close the connection.

8. The next step is to log in to your zCX instance by using the `ssh admin@129.40.23.72 -p 8022` command and you will see the welcome message shown in Example 6-5.

```
Example 6-5  zCX login welcome message
ZCXADM3 @ SC74:/u/zcxadm3>ssh admin@129.40.23.72 -p 8022
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
Last login: Fri Jun 26 14:23:27 2020 from 10.1.100.74
Sudo only permits specific User Management functions. See additional documentation for details.
```

9. You can extract the contents of the gitea.tar file by using the following command:
```
tar -xf gitea.tar
```

10. Go to the folder where you extracted Gitea by using the `cd gitea` command and verify that the Dockerfile is there. Use the following command:
```
ls -l | grep Dockerfile
```
You should get output similar to that shown in Example 6-6.

```
Example 6-6  Output of ls -l | grep Dockerfile command
admin@sc74cn05:~$ls -l | grep Dockerfile
-rw-r--r-- 1 admin admin   1298 Jun 24 17:50 Dockerfile
```

11. As you now have the necessary files for building a Gitea Docker image in your zCX instance, you can run the build command shown here:
```
docker build -f Dockerfile -t gitea --build-arg GITEA_VERSION=v1.12.1 .
```
The output you receive should, as shown in Example 6-7, contain the Docker image ID and the name tag.
**Example 6-7  Output of docker build command**

```
admin@sc74cn05:~$docker build -f Dockerfile -t gitea --build-arg GITEA_VERSION=v1.12.1 . 
Successfully built 517f0c538739
Successfully tagged gitea:latest
```

---

### 6.1.2 Set up a private registry

If you want to share the image with people who have access to your zCX instance or want to have a central place for storing your Docker images securely, you can set up a private registry and push your Docker images to that registry.

**Note:** Although it is optional to set up a private registry at this point, it is recommended because you will need it later in the DevOps flow anyway.

As the Docker image for a private registry is officially available in Docker Hub for the IBM Z platform there is no need to build your own image. For more details, see:

https://hub.docker.com/r/ibmcom/registry-s390x

**Note:** If you want to understand how the Dockerfile for the private registry works or do not find a registry image with the version you want to install, you can find the Dockerfile here:


You can copy the file as described for the Gitea Dockerfile in Chapter 6.1.1, “Build a Gitea Docker image” on page 152 or you can follow the instructions in the IBM Redbooks publication, *Getting started with z/OS Container Extensions and Docker*, SG24-8457 pages 114-116.

To avoid dependencies between the registry container runtime and the stored Docker images, it is recommended that you create a volume for storing the registry content outside of the Dockerfile. You can do this by using the following command:

```
docker volume create registry_data
```

You can run a registry container by referencing the image in the Docker Hub. To do this, issue the following command:

```
docker run -d -p 5000:5000 --name registry -v registry_data:/var/lib/registry --rm ibmcom/registry-s390x:2.6.2.5
```

If the container started successfully, the command will return the container ID. The container ID in our case was:

7086f34b2b83a74765a7d04bf40f7ac14be7e0a6532e3584d5b222327035ff2f

Since you now have a running registry container, you can tag and push in the Gitea Docker image you created in Chapter 6.1.1, “Build a Gitea Docker image” on page 152:

```
docker tag gitea localhost:5000/gitea
```

Then run:

```
docker push localhost:5000/gitea
```

**Note:** Although it is optional to set up a private registry at this point, it is recommended because you will need it later in the DevOps flow anyway.

**Note:** If you want to understand how the Dockerfile for the private registry works or do not find a registry image with the version you want to install, you can find the Dockerfile here:


You can copy the file as described for the Gitea Dockerfile in Chapter 6.1.1, “Build a Gitea Docker image” on page 152 or you can follow the instructions in the IBM Redbooks publication, *Getting started with z/OS Container Extensions and Docker*, SG24-8457 pages 114-116.
8471ch05.fm

Draft Document for Review August 31, 2020 2:54 pm

A Docker image consists of different layers. As you can see in Example 6-8 , each layer will
be pushed to the private registry and will get its own ID.
Example 6-8 Output of docker push localhost:5000/gitea command

admin@sc74cn05:~$docker push localhost:5000/gitea
The push refers to repository [localhost:5000/gitea]
c8dfb294bbd4: Pushed
229358ff0258: Pushed
88114cb2d528: Pushed
4752bb9f426e: Pushed
b52e0a82230c: Pushed
d3fde5edeb97: Pushed
latest: digest:
sha256:eaa7c102700dd31a71102625062c5632f3893d879da0bb1813cdbfc03ec9643e size: 1575
If the image was pushed sucessfully into the private registry, you will get the image name and
tag as shown in Example 6-9 by calling its API with the following curl command:
curl 129.40.23.72:5000/v2/gitea/tags/list
Example 6-9 Output of curl 129.40.23.72:5000/v2/gitea/tags/list command

admin@sc74cn05:~$curl 129.40.23.72:5000/v2/gitea/tags/list
{"name":"gitea","tags":["latest"]}

6.2 Running a Gitea Docker container
In order to run a Gitea Docker container, you must first create a volume to make sure the data
stored in the Gitea container is persistent. To do this, run the following command:
docker volume create gitea_data
Example 6-10 shows typical output if the creation of the volume was sucessful.
Example 6-10 Output of Docker volume create gitea_data command

admin@sc74cn05:~$docker volume create gitea_data
gitea_data

In this section, we describe three options you have to run a Gitea Docker container.

6.2.1 Option 1: Run a Gitea Docker container from a Dockerfile
If a Docker image is already built in zCX it can also be run from your local image cache
in your instance, as shown in Example 6-11. It is important to map a volume for
persistent storage to your container in the run command. Also, it can be very useful to
add the --rm option which automatically removes the container if it stopped for some
reason. This way it will not block any space in your zCX instance and can be restarted.

Chapter 6. Using Gitea as a code repository

155


Example 6-11  Run a Gitea Docker container command

docker run -d -p 3008:3000 -p 24:24 -v gitea_data:/data --name gitea --rm gitea

As shown in Example 6-12 the container ID is returned by the run command.

Example 6-12  Output of docker run command

admin@sc74cn05:~$docker run -d -p 3008:3000 -p 24:24 -v gitea:/data --name gitea --rm gitea
415885ef47f99fd4377546cfa5ef4ffe6539787810997ee074b19c2bfb05169f

6.2.2 Option 2: Run a Gitea Docker container from a private registry

If you store the Gitea image in a private registry, as described in Chapter 6.1.2, “Set up a private registry” on page 154, you can start your Gitea container by simply using the image name and registry address, as shown in Example 6-13.

Example 6-13  Start the Gitea container

docker run -d -p 3008:3000 -p 24:24 -v gitea_data:/data --name gitea --rm localhost:5000/gitea

Again, you would see the Container ID as output in the console.

6.2.3 Option 3: Run a Gitea Docker Container from dockerhub

If you want to start very easily with just running a Gitea container without building an image, you can pull and run an image we created from Docker Hub by using the following commands to pull the image and then run it:

    docker pull maikehavemann/gitea-s390x:latest
    docker run -d -p 3008:3000 -p 24:24 -v gitea_data:/data --name gitea_data --rm maikehavemann/gitea-s390x

The image from the pull command will be downloaded and copied to your local system. If the start command was successful, you will see the container ID in the console.

6.3 Check the Gitea Docker container status

By using the docker ps command, you will be able to check if the container runs and is configured correctly. You should see an output similar to what is shown in Example 6-14.

Example 6-14  Output of docker ps command

admin@sc74cn05:~$docker ps
9d7bf05ca731    maikehavemann/gitea-s390x
"/usr/bin/entrypoint..." 4 days ago Up 4 days 22/tcp,
0.0.0.0:24->24/tcp, 0.0.0.0:3008->3000/tcp gitea
6.4 Configure Gitea

To configure your running Gitea container, you will need to open the Gitea user interface in a browser. In our case the, the IP address is **129.40.23.72:3008**.

The user interface in Figure 6-1 should look the same in your browser.

![Gitea Welcome Screen](image)

**Figure 6-1**  Gitea Welcome Screen

Click ‘Register’ in the upper right corner, indicated by the red arrow in Figure 6-1.

The Initial Configuration window shown in Figure 6-2 will pop up. Check your Gitea Base URL and HTTP Listen Port and change them if they are not set correctly already.
Fill out the registration form shown in Figure 6-3 to create an account in Gitea.
6.4.1 Upload Code to Gitea

After the registration, the Gitea overview shown in Figure 6-4 will appear. To create a new code repository, click the plus symbol next to ‘Repositories’.

This will provide you with a form for your new repository where you can name the repository and add a description. Figure 6-5 shows an example of what the form should look like.
Click the “Create Repository” button after you have completed filling it out.

After the creation of a repository, you will get instructions on how to add code to this repository, as shown in Figure 6-6.
In this case, a simple web application written in nodejs is pushed to the master branch of this repository. It is an application that returns the text ‘Hello zCX User!’ when called. The folder containing instructions to download the application files can be found in Appendix A, “Additional material” on page 233. It is included in the DevOps added material folder.

Note: To deploy the application in zCX, a Dockerfile is required. The repository provided in Appendix A, “Additional material” on page 233 already contains the necessary Dockerfile.

After pushing code to Gitea the repository should look similar to what is shown in Figure 6-7.
Using Jenkins to automate builds

This chapter describes how to set up a Docker container for Jenkins server and agent.

The following sections are included:
- “What is Jenkins?” on page 164
- “Build a Jenkins server Docker image” on page 164
- “Run a Jenkins server Docker container” on page 167
- “Check Jenkins server Docker container status” on page 168
- “Configure Jenkins” on page 168
- “Set up a Jenkins build agent” on page 176
- “Check Jenkins build agent Docker container status” on page 184
7.1 What is Jenkins?

Jenkins is a self-contained, open-source server which can be used to set up a robust CI/CD environment to automate build, test and deploy tasks. It is possible to use Jenkins within Docker, through native system packages or as a standalone version with a Java Runtime Environment.

Once Jenkins is installed, you can install default or custom plugins. Plugins in Jenkins are enhancements of functionalities to meet specific task requirements. Jenkins offers thousands of plugins which can be added manually as well as installed on the server’s user interface.

The Jenkins server or so-called master acts to schedule jobs, assign agents and send builds to them to execute the jobs. The official Jenkins documentation recommends to always set up dedicated build nodes than run separately from the master to free up resources to improve the masters performance. The Jenkins master use resources to only handle HTTP requests and manage the environment. The execution of builds will be delegated to the nodes called agents. The advantage is here builds are prevented from being able to modify sensitive data in the masters home directory.

The CI/CD pipeline executed in Jenkins requires a pipeline script. This script defines the build process trough the steps of testing and deployment. It is called Jenkinsfile.

For more information, see:

https://www.jenkins.io/doc/

7.2 Build a Jenkins server Docker image

The Dockerfile for building a Jenkins server container for the s390x architecture can be found in the Appendix A, “Additional material” on page 233 of this book.

If you have not already done so, download the added material folder for DevOps and navigate in the terminal to the folder in which the Docker file of the Jenkins server is located.

Run the following command to copy the file to your z/OS:

```bash
scp Dockerfile maikeh@wtsc74.pbm.ihost.com:/tmp
```

As shown in Example 7-1 you will see the process of copying in your console.

```
Example 7-1 Output of scp Dockerfile ... command
base:Server files maike$scp Dockerfile maikeh@wtsc74.pbm.ihost.com:/tm
zcxadm3@wtsc74.pbm.ihost.com's password: Dockerfile
100% 2037  2.0KB/s  00:01
```

After copying the file is accessible in z/OS. Login to your z/OS system:

```
ssh -l zcxadm3 wtsc74.pbm.ihost.com
```

Then go to the /tmp folder where the Dockerfile was copied:

```
cd /tmp
```
To verify the file was copied correctly, display files containing Dockerfile with the `ls -l | grep Dockerfile` command, which generates the output shown in Example 7-2:

```
Example 7-2  Output of ls -l | grep Dockerfile command
ZCXADM3 @ SC74:/SC74/tmp> ls -l | grep Dockerfile
-rw-r--r--  1 ZCXADM3 SYS1        2037 Jun 26 13:22 Dockerfile
```

Transfer the file securely by logging into your zCX instance via `sftp`:

```
sftp -P 8022 admin@129.40.23.72
```

You will receive the output in Example 7-3 if the `sftp` connection was established successfully.

```
Example 7-3  Output of sftp -P 8022 admin@129.40.23.72 command
ZCXADM3 @ SC74:/SC74/tmp>sftp -P 8022 admin@129.40.23.72
Connected to 129.40.23.72.
```

To ensure the right file format when copying files to another system, set file encoding to ASCII:

```
ascii unix
```

Then copy the Dockerfile from z/OS to your zCX instance:

```
put Dockerfile
```

As shown in Example 7-4 you will see the progress in your console.

```
Example 7-4  Output of put Dockerfile command
sftp>put Dockerfile
Uploading Dockerfile to /home/admin/Dockerfile
Dockerfile
100% 2037     2.0KB/s   00:00
```

At this point you have the Dockerfile file stored in your zCX instance and type the `exit` command to close the connection.

The next step is to log into your zCX instance with the `ssh admin@129.40.23.72 -p 8022` command and you will see the welcome message as shown in Example 7-5.

```
Example 7-5  zCX login welcome message
ZCXADM3 @ SC74:/u/zcxadm3>ssh admin@129.40.23.72 -p 8022
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
Last login: Fri Jun 26 14:23:27 2020 from 10.1.100.74
Sudo only permits specific User Management functions. See additional documentation for details.
```

Double check to ensure the Dockerfile is copied successfully into your zCX instance by using the following command:
ls -l | grep Dockerfile

You should get output similar to what is shown in Example 7-6.

Example 7-6  Output of ls -l | grep Dockerfile command
admin@sc74cn05:~$ls -l | grep Dockerfile
-rw-r--r-- 1 ZCXADM3 SYS1 2037 Jun 26 13:22 Dockerfile

As you now have the necessary files for building a Jenkins server Docker image in your zCX instance, you can run the build command as follows:

docker build -f Dockerfile -t jenkins_server .

The output you receive should, as shown in Example 7-7, contain the Docker image ID and the name tag.

Example 7-7  output of docker build ... jenkins_server . command
admin@sc74cn05:~$docker build -f Dockerfile -t jenkins_server .
Successfully built 48277d1bcefb
Successfully tagged jenkins_server:latest

If you have already have set up a private registry as discussed in Chapter 6.1.2, “Set up a private registry” on page 154, it is now possible to tag and push the Jenkins server Docker image to the registry using the following command:

docker tag jenkins_server localhost:5000/jenkins_server

To push the Docker image to the registry, run the following command:

docker push localhost:5000/jenkins_server

Example 7-8 shows the output for the push command of the Jenkins server Docker image to the registry.

Example 7-8  Output of docker push localhost:5000/jenkins_server command
admin@sc74cn05:~$docker push localhost:5000/jenkins_server
The push refers to repository [localhost:5000/jenkins_server]
a883e664bc9b: Pushed
8af47ae9407e: Pushed
efb58ec2d5b2: Pushed
f5f0f8543c72: Pushed
e6592647728f: Pushed
latest: Digest:
sha256:46e9bd5458f01393c44da9e6c08b692efb6b708d921fe5629b5b56069c12b98d size: 1365

If the image was pushed successfully into the private registry you will get the image name and tag as shown in Example 7-9 by calling its API with the following curl command:

curl 129.40.23.72:5000/v2/jenkins_server /tags/list

Example 7-9  Output of curl 129.40.23.72:5000/v2/jenkins_server /tags/list command
admin@sc74cn05:~$curl 129.40.23.72:5000/v2/jenkins_server /tags/list
{"name":"jenkins_server","tags":["latest"]}
7.3 Run a Jenkins server Docker container

In order to run a Jenkins server Docker container, you must first create a volume to make sure the data is persistent in the Jenkins server Docker container. You would create the volume by issuing the following command:

```bash
docker volume create jenkins_data
```

Example 7-10 shows typical output if the creation of a volume was successful.

```
Example 7-10   Output of docker volume create jenkins_data command
admin@sc74cn05:~$docker volume create jenkins_data
jenkins_data
```

In this section, we describe three options you have to run a Jenkins server Docker container.

7.3.1 Option 1: Run a Jenkins server Docker container from a Dockerfile

If a Docker image is already built in zCX, it can also be run from your local image cache in your instance. It is important to map a volume for persistent storage to your container in the run command. Also, it can be very useful to add the `--rm` option which automatically removes the container if it is stopped for some reason. This way it won’t block any space in your zCX instance and can be restarted. The default port for the Jenkins user interface is set to 8080. Additionally, a port 50000 will be set for the Jenkins server to receive requests from outside its own container. To run a Jenkins server Docker container from a Dockerfile, issue the following command:

```bash
docker run -d -p 3000:8080 -p 50000:50000 -v jenkins_data:/data --name jenkins_server --rm jenkins_server
```

As shown in Example 7-11 the container ID is returned by the run command.

```
Example 7-11   Output of docker run ... jenkins_server command
admin@sc74cn05:~$docker run -d -p 3000:8080 -p 50000:50000 -v jenkins_data:/data --name jenkins_server --rm jenkins_server
5e848255d41ca219524f777a16b0f7a387a57655c207ca1b5407a598f0b4880
```

7.3.2 Option 2: Run a Jenkins server Docker Container from private registry

If the Jenkins server image was pushed to the private registry, as described in Chapter 7.2, “Build a Jenkins server Docker image” on page 164, you are able to run the Jenkins server image from the registry:

```bash
docker run -d -p 3000:8080 -p 50000:50000 -v jenkins_data:/data --name jenkins_server --rm localhost:5000/jenkins_server
```

Again, you would see the container ID as output in the console.

7.3.3 Option 3: Run a Jenkins server Docker Container from dockerhub

If you want to start very easily with just running a Jenkins server container without building an image you can pull and run the image from Docker Hub by using the following command:
7.4 Check Jenkins server Docker container status

Using the docker ps command allows you to check if the container runs and is configured correctly. You should see an output similar to what is shown in Example 7-12.

Example 7-12  output of docker ps | grep jenkins_server command

```
admin@sc74cn05:~$ docker ps | grep jenkins_server
5e848255d41c       localhost:5000/jenkins_server   "/bin/sh -c 'java -j…"   7
minutes ago       Up 7 minutes        0.0.0.0:50000->50000/tcp,
0.0.0.0:3000->8080/tcp     jenkins_server
```

7.5 Configure Jenkins

To configure your running Jenkins server container you need to open the user interface in a browser. Navigating to the adress `129.40.23.72:3000` leads to the Jenkins Getting Started page, shown in Figure 7-1.

![Figure 7-1  Jenkins Start Screen](image)

```
docker run -d -p 3000:8080 -p 50000:50000 -v jenkins_data:/data --name jenkins_server --rm maikehavemann/jenkins-server-s390x
```

The image will be downloaded and copied to your local system. If the start command was successful, the container ID would be displayed as output in the console.
You will get the required initial password by running `docker logs jenkins_server`. The password can be found in the command line output highlighted red in Example 7-13.

**Example 7-13  Output of docker logs jenkins_server command**

```
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:
  e7658f1ece7e4dd58ed796c2791e69c1
This may also be found at: /root/.jenkins/secrets/initialAdminPassword
```

Insert the password into the Administrator password field of the Jenkins Getting Started screen and click continue.

In the next screen, Figure 7-2, you can customize Jenkins and install plugins.
Install the suggested plugins and you will see the progress in the screen, as shown in Figure 7-3.
Figure 7-3  Jenkins Progress Install Plugins

**Note:** Occasionally, the installation of some plugins may fail. It is possible to install plugins later manually. This is going to be demonstrated with the Ansible plugin in Chapter 7.5.1, “Install Plugins manually” on page 174.

Now you can create an administrative user for your Jenkins server, as shown in Figure 7-4.
Figure 7-4  Jenkins Create Admin User

After filling in your credentials, click ‘Save and Continue’.

As shown in Figure 7-5, it is necessary to insert your Jenkins server URL into the configuration.
Click ‘Save and Finish’ and you will get the “Jenkins is ready” message, displayed in Figure 7-6.
The initial Jenkins setup is complete so you can click on the ‘Start using Jenkins’ button.

### 7.5.1 Install Plugins manually

At this point the Ansible plugin will be already be installed, In this section, we show how to install a plugin manually. Ansible is required later on for the use in the application deployment in Chapter 8.2, “Set up Ansible using Jenkins” on page 186.

When the Jenkins setup is complete you will see the Jenkins home screen in Figure 7-7. To install a plugin manually click on ‘Manage Jenkins’.

Click on the ‘Manage Plugins’ menu as shown in Figure 7-8.
Search for the Ansible plugin, check the box and click on ‘Download and install after restart’. This step can be seen in Figure 7-9.

After the installation is complete, click on ‘Restart Jenkins’ you will see the boot screen as shown in Figure 7-10.
7.6 Set up a Jenkins build agent

The purpose of Jenkins is to automate build jobs. Jobs are executed by dedicated agents. To build the image of the node.js example application, an agent is set up. In this section, we describe how to create a Jenkins build node, build the Jenkins agent Docker image and provide two options to run a Jenkins build agent Docker container.

7.6.1 Create a Jenkins build node

From the Jenkins home screen (Figure 7-7) click on ‘Manage Jenkins’ again. As shown in Figure 7-11, instead of ‘Manage plugins’, select ‘Manage nodes and clouds’.
Click on ‘New Node’, as shown in Figure 7-12.

Create the agent as a permanent agent and configure it as shown in Figure 7-13.
You be provided with an overview of your agent. Copy the agent secret highlighted in Figure 7-14. You will need the secret later to run the build agent container.

To make sure the agent container can communicate with the Jenkins server, it is necessary to set the server’s TCP inbound port to 50000. This port was set in section 7.3, “Run a Jenkins server Docker container” as a communication endpoint for the Jenkins server.

Go back to ‘Manage Jenkins’ (Figure 7-7). Then go to the Jenkins security menu, shown in Figure 7-15.

As shown in Figure 7-16, set the TCP port for inbound agents to 5000 and select ‘fixed’.
Enable 'Allow anonymous read access', to allow requests to the Jenkins URL, shown in Figure 7-17.

7.6.2 Build a Jenkins agent Docker image

In this section, we describe how to build the Docker image to run a Jenkins build agent. The files (including the Dockerfile) required to build and run the agent container can be found in Appendix A, “Additional material” on page 233 of this book.
Note: The Jenkins agent Dockerfile provided is customized with an installation of AdoptJDK, a Git client, Python, Ansible, Docker and node.js.

Additionally, a system update and installation of required utility libraries is performed. All these tools are preinstalled in the Docker image so that the agent container created from this image can be used for build or deploy and the tools need not be installed at each pipeline run from scratch.

If you have not already done so, download the additional materials folder for DevOps and navigate in the terminal to the folder in which the Jenkins agent files are located. Run the following command to copy the file to your z/OS:

```
scp jenkins_agent.tar zcxadm3@wtsc74.pbm.ihost.com:/tmp
```

As shown in Example 7-14, you will see the process of copying in your console.

```
Example 7-14   Output of scp jenkins_agent.tar ... command

base:Agent maike$ scp jenkins_agent.tar zcxadm3@wtsc74.pbm.ihost.com:/tmp
zcxadm3@wtsc74.pbm.ihost.com's password:
jenkins_agent.tar
100% 2996KB 129.9KB/s   00:23
```

After completion of the copy process, your file is accessible in z/OS. Log in to your z/OS system:

```
ssh -l zcxadm3  wtsc74.pbm.ihost.com
```

Navigate to the /tmp folder where the jenkins_agent.tar file was copied with the following command:

```
cd /tmp
```

To verify the file was copied correctly, display files containing jenkins_agent with the `ls -l | grep jenkins_agent` command, which generates the output shown in Example 7-15:

```
Example 7-15   Output of ls -l | grep jenkins_agent command

ZCXADM3 @ SC74:/SC74/tmp>ls -l | grep jenkins_agent
-rw-r--r--   1 MAIKEH   SYS1     3067392 Jun 26 14:41 jenkins_agent.tar
```

Transfer the file securely by logging into your zCX instance with `sftp`:

```
sftp -P 8022 admin@129.40.23.72
```

You will receive the output shown in Example 7-16 if an `sftp` connection was established successfully.

```
Example 7-16   Output of sftp -P 8022 admin@129.40.23.72 command

ZCXADM3 @ SC74:/SC74/tmp>sftp -P 8022 admin@129.40.23.72
Connected to 129.40.23.72.
```

To ensure the right file format when copying files to another system, set file encoding to ASCII with the following command:

```
ascii unix
```
Copy jenkins_agent.tar from z/OS to your zCX instance with the following command:

   put jenkins_agent.tar

As shown in Example 7-17, you will see the progress in your console.

Example 7-17  Output of put jenkins_agent.tar command

   sftp>put jenkins_agent.tar
   Uploading jenkins_agent.tar to /home/admin/jenkins_agent.tar
   jenkins_agent.tar
   100% 2996KB   2.9MB/s   00:01

At this point you have the jenkins_agent.tar file stored in your zCX instance and use the exit command to close the connection.

The next step is to log into your zCX instance with the ssh admin@129.40.23.72 -p 8022 command and you will see the Welcome message as shown in Example 7-18.

Example 7-18  zCX login welcome message

   ZCXADM3 @ SC74:/u/zcxadm3>ssh admin@129.40.23.72 -p 8022
   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
   Last login: Fri Jun 26 14:23:27 2020 from 10.1.100.74
   Sudo only permits specific User Management functions. See additional documentation for details.

You can extract the content of jenkins_agent.tar by using the following command:

   tar -xf jenkins_agent.tar

Go to the folder extracted from jenkins_agent.tar by using the cd agent command and check if the Dockerfile is there with the following command:

   ls -l | grep Dockerfile

You should get output similar to that which is shown in Example 7-19.

Example 7-19  Output of ls -l | grep Dockerfile command

   admin@sc74cn05:~$ls -l | grep Dockerfile
   -rw-r--r-- 1 admin admin   1298 Jun 24 17:50 Dockerfile

As you now have the necessary files for building a Jenkins agent Docker image in your zCX instance, you can run the following build command:

   docker build -f Dockerfile -t jenkins_agent .

The output you receive should, as shown in Example 7-20, contain the Docker image ID and the name tag.

Example 7-20  Output of docker build ... jenkins_agent . command

   admin@sc74cn05:~$docker build -f Dockerfile -t jenkins_agent .
   Successfully built 86827336dc16
   Successfully tagged jenkins_agent:latest
If you have already set up a private registry in “Set up a private registry” on page 154 you can tag the Jenkins agent Docker image to the registry with the following command:

```
docker tag jenkins_agent localhost:5000/jenkins_agent
```

Push the Jenkins agent Docker image by using the following command:

```
docker push localhost:5000/jenkins_agent
```

Example 7-21 shows the output for the push command of the Jenkins agent Docker image to the registry.

```
Example 7-21   Output of docker push localhost:5000/jenkins_agent command
admin@sc74cn05:~$ docker push localhost:5000/jenkins_agent
The push refers to repository [localhost:5000/jenkins_agent]
2950e536b743: Pushed
826e5378699f: Pushed
34f2ba2303ab: Pushed
213f288f7dc8: Pushed
32653909e039: Pushed
bb2e8c728fbc: Pushed
969be5474d21: Pushed
dc710a402ff1: Pushed
4a4819c7f7bc: Pushed
b705bfa12379: Pushed
8c3886cb258d: Pushed
b07d92af0b040: Pushed
7d6a83014f9f: Pushed
0c878cd793cb: Pushed
cce965e350f9: Pushed
202aae63ce0b: Pushed
fc9f77e2034f: Pushed
6a155b59d48: Pushed
d3c9262050e0: Pushed
30525bf79200: Pushed
ebaeae86b1e4: Pushed
4c5823ae13d6: Pushed
62d18496ede1: Pushed
latest: digest:
sha256:ff1c26e604eca450dafa2a5f0965247352f11e431abde2a9f59ba8b3d2000623 size: 5151
```

If the image was pushed sucessfully into the private registry, you will get the image name and tag by calling its API with the curl command. The output to this command is shown in Example 7-22.

```
curl 129.40.23.72:5000/v2/jenkins_agent /tags/list
```

```
Example 7-22   Output of curl 129.40.23.72:5000/v2/jenkins_agent /tags/list command
admin@sc74cn05:~$ curl 129.40.23.72:5000/v2/jenkins_agent /tags/list
{"name":"jenkins_agent","tags":["latest"]}
```
7.7 Run a Jenkins build agent Docker container

In this section, we describe three different options you have to run a Jenkins build agent Docker container.

7.7.1 Option 1: Run a Jenkins build agent Docker container from a Dockerfile

If a Docker image is already built in zCX, it can also be run from your local image cache in your instance. The container needs permission to access network interfaces such as a Docker socket. For this reason `--cap-add` needs to be included in the run command, as shown in Example 7-23. Also, it is essential to add the agent name and secret you created in section 7.5, “Configure Jenkins” on page 168.

**Note:** By default zCX does not allow full read/write (rw) access to the Docker socket, the network interface to control the Docker engine. The read only mode (:ro) can be used as an extension to the docker socket. This allows you to create containers without rw access and won't be refused by the zCX engine.

**Example 7-23  Running a Docker image from local image cache**

```
```

As shown in Example 7-24, the container ID is returned by the run command:

**Example 7-24  output of docker run ... j-build-agent command**

```
admin@sc74cn05:~$docker run -d --rm --name j-build-agent --cap-add ALL -v /var/run/docker.sock:/var/run/docker.sock:ro jenkins_agent -url http://129.40.23.72:3000 
```

7.7.2 Option 2: Run a Jenkins build agent Docker container from a private registry

If you pushed your Jenkins agent image to the private registry, you are able to run the Jenkins agent image from the registry, as shown in Figure 7-25.

**Example 7-25  Run the agent from the registry**

```
```

You would see the container ID as output in the console.
7.7.3 Option 3: Run a Jenkins build agent Docker Container from dockerhub

If you want to start very easily with just running a Jenkins agent container without building an image you can pull and run the image from Docker Hub by using the following command:

```
```

The image will be downloaded and copied to your local system. If the start command was successful the Container ID would be displayed as an output in the console.

7.8 Check Jenkins build agent Docker container status

Using the `docker ps` command allows you to check if the container runs and is configured correctly. You should see output similar to what is shown in Example 7-26.

```
Example 7-26    Output of docker ps | grep j-build-agent command

admin@sc74cn05:~$ docker ps | grep j-build-agent
45cc9bb53c4f       maikehavemann/jenkins-agent-s390x:latest
"/usr/local/bin/jenk..." 11 seconds ago  Up 10 seconds
j-build-agent
```
Using Ansible to automate deployment and tests

This chapter provides instructions for writing Ansible playbooks and setting up an agent for deployment.

Included in this chapter:

- “What is Ansible?” on page 186
- “Set up Ansible using Jenkins” on page 186
- “Create an Ansible playbook for deployment to a development environment” on page 186
- “Create an Ansible playbook for integration tests” on page 187
- “Create an Ansible playbook for deployment to a test environment” on page 187
- “Create a deployment agent for Ansible in Jenkins” on page 188
- “Set up a Jenkins deployment agent” on page 188
8.1 What is Ansible?

Ansible is an automation engine that automates multiple tasks like cloud provisioning and multi-tier application deployment. Instead of writing a lot of commands to deploy containers like in Jenkins automation jobs in Ansible are described in YAML playbooks what allows you to create a deployment very easily. It is possible to install Ansible as a standalone tool or as a plugin in your CI/CD pipeline tools as well.

Ansible connects your nodes and pushes out programs, called Ansible modules, to them. It starts these modules over SSH and removes them when the execution is finished.

For more information, see:

https://docs.ansible.com/

8.2 Set up Ansible using Jenkins

This section describes how to create Ansible playbooks for deployment to an environment and for integration tests of a simple node.js app. Ansible itself will be set up in a Jenkins agent container. The playbooks can be found in Appendix A, “Additional material” on page 233.

For more information on how to create Ansible playbooks, see:

https://docs.ansible.com/ansible/latest/network/getting_started/first_playbook.html

8.2.1 Create an Ansible playbook for deployment to a development environment

In the first step, the Ansible framework is used to run the Docker image of the simple node.js application in a development environment. This environment is simulated here by running the application on another port but realistically could be a different network and Docker node.

On the agent which runs the deploy pipeline, you will need the latest version of pip and docker-py.

Once the container for the node.js Docker image is started, the image is pulled from a Docker registry. This step is described later in Chapter 9, “Putting it all together and run the pipeline” on page 193. The Ansible playbook to run the described procedure can be found in Example 8-1.

Example 8-1 Ansible Playbook for deployment to a dev environment

```yaml
- name: build and run the docker container
  hosts: localhost
  tasks:
    - name: Install pip
      apt: name=python-pip state=present
    - name: install docker-py
      pip: name=docker-py
    - name: Creating the container
      docker_container:
```


8.2.2 Create an Ansible playbook for integration tests

To test the deployment, the playbook shown in Example 8-2 executes a simple integration test.

Example 8-2  Ansible Playbook for simple integration test

```yaml
- name: Simple Integration Test
  hosts: localhost
  tasks:
    - name: Running the container
      docker_container:
        name: hello-node-dev
        published_ports:
          - "3004:3000"
        image: localhost:5000/hello-node:latest
        state: started
```

The test sends an HTTP GET request to the application and expects “zCX” to be contained in the response. If the server returns anything else, for example, an error with status code 400 and empty body, the test will fail. For more sophisticated testing, Ansible provides many tools and utilities. For more information, see:

https://docs.ansible.com/ansible/latest/dev_guide/testing.html

8.2.3 Create an Ansible playbook for deployment to a test environment

When integration tests have passed, the third step is creating and starting the application in a test environment using the playbook named 'playbook-test', shown in Example 8-3. The steps of this stage are almost identical with those from the development playbook, except for the port of the running container.

Example 8-3  Ansible Playbook for deployment to a test environment

```yaml
- name: build and run the docker container
  hosts: localhost
  tasks:
    - name: Install pip
      apt: name=python-pip state=present
    - name: install docker-py
      pip: name=docker-py
    - name: Creating the container
```
This test instance can then be used by testers for acceptance and other additional tests before the application in the production environment is updated. For simplicity, we do not show this last step here as it is manual and highly dependent on the specific customer infrastructure and requirements.

8.2.4 Create a deployment agent for Ansible in Jenkins

Jenkins allows you to provision environments and deploy applications with the use of shell scripts. Shell scripts can be cumbersome to maintain and reuse. When Ansible is used in the pipeline, it is possible to easily reuse playbooks for deployment and provisioning. Jenkins can remain as an orchestrator and job scheduler instead of a shell script executor. Also, playbooks can be stored in a versioning system.

Jenkins allows use the of Ansible as a plugin and can therefore be integrated in an existing CI/CD flow. As the Ansible plugin was already installed in section 7.5.1, “Install Plugins manually” on page 174, a Jenkins agent for the Ansible deployment can be set up now.

8.3 Set up a Jenkins deployment agent

The purpose of Jenkins is to automate jobs. Jobs are executed by dedicated agents. To build the image of the node.js example application, an agent is set up. In this section, we describe how to set up a deployment agent.

8.3.1 Create a Jenkins deployment node

The first step in setting up a Jenkins deployment agent is to create a Jenkins deployment node.

Go back to the Jenkins server user interface by accessing its URL, in our case 129.40.23.72:3000. Then click ‘Manage Jenkins’ on the left side of the screen.

As shown in Figure 8-1, select select ‘Manage nodes and clouds’.
Click on ‘New Node’.

Create the agent as a permanent agent and configure it as shown in Figure 8-2.

You will be provided with an overview of your agent. Copy the agent secret highlighted in Figure 8-3. You will need the secret later to run the deploy agent container.
8.4 Run a Jenkins deploy agent Docker container

In this section, we describe three different options you have to run a Jenkins deploy agent Docker container.

8.4.1 Option 1: Run a Jenkins deploy agent Docker container from a Dockerfile

As in Chapter 7.2, “Build a Jenkins server Docker image” on page 164, the Docker image that has all needed components for build and deployment jobs installed has been built already. The container for the Jenkins deploy agent can be run using the local build, as shown in Example 8-4.

Example 8-4 Running the Docker image from local image cache

34747350a14f981402816f91c731f7067860aa33923f1133e7064722c6c2e7e64 j-deploy-agent

As shown in Example 8-5, the container ID is returned by the run command:

Example 8-5 output of docker run ... j-build-agent command

admin@sc74cn05:~$docker run -d --rm --name j-build-agent --cap-add ALL -v /var/run/docker.sock:/var/run/docker.sock:ro jenkins_agent -url http://129.40.23.72:3000
34747350a14f981402816f91c731f7067860aa33923f1133e7064722c6c2e7e64 j-deploy-agent

8e82ad1d98be97fa18c887d5f7884df223d05fe603d5d17e9125b63d2c5bfa05

8.4.2 Option 2: Run a Jenkins deploy agent Docker container from a private registry

If you pushed your Jenkins agent image to the private registry, as described in “Build a Jenkins agent Docker image” on page 179, you are able to run the Jenkins agent image from the registry, as shown in Example 8-6.
8.4.3 Option 3 Run a Jenkins deploy agent Docker Container from dockerhub

If you want to start very easily with just running a Jenkins agent container without building an image you can pull and run the image from dockerhub, as shown in Example 8-7.

Example 8-7 Pull and run from dockerhub

```bash
docker run -d --rm --name j-deploy-agent --cap-add ALL -v /var/run/docker.sock:/var/run/docker.sock:/romaikehavemann/jenkins-agent-s390x:latest -url http://129.40.23.72:3000 34747350a14f981402816f91c731f7067860aa33923f1133e7064722c6c2e7e64 j-deploy-agent
```

The image will be downloaded and copied to your local system. If the start command was successful you would again see the container ID in the console.

8.5 Check Jenkins deploy agent Docker container status

Using the `docker ps` command allows you to check if the container runs and is configured correctly. You should see output similar to what is shown in Example 8-8.

Example 8-8 output of docker ps | grep j-deploy-agent command

```
admin@sc74cn05:~$docker ps | grep j-deploy-agent 8e82ad1d98be maikehavemann/jenkins-agent-s390x:latest "/usr/local/bin/jenk..." 22 seconds ago Up 22 seconds j-deploy-agent
```
Putting it all together and run the pipeline

If you have completed Chapter 6. “Using Gitea as a code repository”, Chapter 7. “Using Jenkins to automate builds” and Chapter 8. “Using Ansible to automate deployment and tests”, you are now ready to put them all together and run the pipeline.

In this chapter the following topics are provided:

Chapter 9.1, “Pipeline Architecture” on page 194
Chapter 9.2, “Set up a pipeline in Jenkins” on page 194
Chapter 9.3, “Create a webhook in Gitea” on page 199
Chapter 9.4, “Run a pipeline in Jenkins” on page 202
Chapter 9.5, “Monitor a pipeline in Jenkins” on page 204
9.1 Pipeline Architecture

After setting up git as a code repository, Jenkins server as pipeline orchestrator, a Jenkins agent to build a Docker image, a private registry to store that image and a Jenkins agent including Ansible to deploy a node.js web application Docker container the pipeline can be set up. The architecture overview in Figure 9-1 shows the DevOps flow that is going to be described in this chapter.

![Pipeline Overview](image)

9.2 Set up a pipeline in Jenkins

To set up a pipeline in Jenkins, navigate to the Jenkins user interface URL, in our example, 129.40.23.72:3000. In the Jenkins home screen select 'New Item', shown in Figure 9-2 on the left side.
Enter an item name and select ‘Pipeline’ to create a pipeline as shown in Figure 9-3.

To make sure the build and deployment pipeline will be run every time a change (push) to the code repository in Gitea is done, enable a build trigger and create an authentication token that is needed later. The configuration is displayed in Figure 9-4.
As shown in Figure 9-5, scroll down to the pipeline section where a pipeline script can be inserted. It is possibly to type a script here or to get it from a source code management system (SCM). The pipeline script in our case is called Jenkinsfile and is provided in Appendix A, “Additional material” on page 233 in the hello-node folder (included in the DevOps added material folder). If you used the provided hello-node files in Chapter 6.4.1, “Upload Code to Gitea” on page 159, the pipeline script is stored in the Gitea repository now.

In the pipeline definition, select script from SCM and for the SCM type, use Git. Also, type in the repository URL from Gitea where the Jenkinsfile is stored. Then, click Apply. Our completed configuration is shown in Figure 9-5.
The Jenkinsfile executes a pipeline in stages, containing one or more steps. A JSON-based structure organizes the mapping from pipeline/stage to an adequate agent and the order of the steps. The Jenkins pipelines are sequential, meaning the order of single executed steps is fixed and if one of those is not successful, for example, returning a non-zero exit code, the whole pipeline stops. After each run, the pipeline can be either in a state of SUCCESS (everything worked correctly and application was built and deployed) or FAILURE (something went wrong and the run was aborted). The pipeline script we used is shown in Example 9-1.

Example 9-1  Pipeline script to build and deploy a Docker container image

```groovy
pipeline {
    // Disables automatic agent selection
    agent none

    stages {

        // Defines build stage containing all steps to
        // build and test a docker image for the nodejs application
        stage('Build') {

            // Select an agent that has the label 'build' attached
            // to run the steps of this stage
            agent{
                node {
                    label 'build'
                }
            }

            // Required steps for the build stage
            // (if one fails the whole stage fails)
            steps {

                // Clones the current data on master branch
                // configured in the pipeline configuration
```
// Install dependencies for the application
sh 'npm install'

// Run jest-based unit tests to check if all
// interfaces are responding correctly
sh 'npm test'

// Build an docker image based on the files of the current directory
// and pretag it with the address of you private registry
sh 'docker build -t localhost:5000/hello-node:latest .'

// Push the tagged image to the private registry
sh 'docker push localhost:5000/hello-node'

// Second stage for deployment to dev/test env and integration tests
stage('Deploy') {

    // Select an agent with the 'test' label attached
    // to run the steps of this stage
    agent{
        node {
            label 'test'
        }
    }

    // Execute several ansible playbooks to deploy the application
    steps{
        // Playbook to create and start an nodejs application container in dev
        // environment
        ansiblePlaybook(inventory: 'hosts.ini', playbook: 'playbook_dev.yaml')

        // Playbook to run integration tests that make api calls against the
        // application container
        ansiblePlaybook(inventory: 'hosts.ini', playbook: 'integration_test.yaml')

        // Playbook to create and start an nodejs application container in test
        // environment
        ansiblePlaybook(inventory: 'hosts.ini', playbook: 'playbook_test.yaml')
    }

    // If boths stages and their corresponding steps are executed successfully,
    // the pipeline finishes and displays "SUCCESS" in logs, else "FAILURE"
}

The script starts with a pipeline tag which denotes the starting point of the pipeline definition. Per default, Jenkins can run a pipeline on any available execution processor. For example, when the Jenkins server is initialized the first time, it has two build execution processors. In this scenario, the DevOps workflow is based on containers with scalability in mind. Hence, the
execution of the pipeline should happen on the agent container which can easily be scaled up.

The agent selector of the pipeline is set to none, overriding the default setting. The two stages define the groups of steps for build and deployment. Stages allow specific selection of an agent from the pool of available agents by using e.g. the build label attached. This way, agent containers with different pre-installed packages can be used for different steps and stages.

The build stage pulls the latest version of the application from the master branch of the Git repository. All files from the repository are stored temporarily in the agent container. Then the required dependencies for the application are downloaded and installed. For this step the `sh` keyword can be used to execute shell commands. The same keyword is used to run local tests with `npm`. A simple unit has been created with the jest framework that allows starting a virtual web server locally to test APIs. In this scenario the hello-node application has only one interface which returns a simple text.

After successful unit tests, a new Docker image is created based on the currently downloaded and installed files. The tag of the image is composed of the address of our private registry (to be found under localhost:5000) and the name of the image (hello-node). Also, the version tag, `latest`, which is set automatically if none is specified, is added. The tagged image is pushed to the private registry that henceforth can be used in other agents and containers.

For deployment, the pipeline switches to a different agent that has the `test` label attached. For the steps of this stage, we use the pre-installed Ansible library and Ansible plug-in in Jenkins. It allows a structured definition of deployment and distribution tasks. In Ansible a set of tasks to run is called a Playbook and stored in YAML files. As described in Chapter 8.2, “Set up Ansible using Jenkins” on page 186 three playbooks are used for the deployment of the application in a development environment, the integration test and the deployment into a test environment, respectively. As there aren’t multiple environments in this scenario, we simulate them with multiple containers and different ports.

### 9.3 Create a webhook in Gitea

Go back to the Gitea URL (129.40.23.72:3008 in our case). Open the hello-node repository and click on ‘Settings’ in the upper right corner, as shown in Figure 9-6.
Select ‘Add Webhook’ and choose Gitea, shown in Figure 9-7.
To create the webhook, the authentication token created in Figure 9-4 is needed. Insert the target URL and the content type of “POST” as shown in Figure 9-8.

![Webhook Configuration](image)

**Figure 9-8  Configure Gitea Webhook**

Then change the HTTP Method in the dropdown menu, as shown in Figure 9-9, to GET and click ‘Add Webhook’.
9.4 Run a pipeline in Jenkins

The pipeline is configured to start with every push/update to the hello-node repository in Gitea. As there is no code update to push at the moment, the ‘Test Delivery’ button in the webhook menu can be used. This is shown in Figure 9-10.
Go back to the Jenkins user interface, in our case 129.40.23.72:3000, and click on the pipeline project that was created in Chapter 9.1, “Pipeline Architecture” on page 194.

Since a test delivery was started, the Jenkins pipeline will now run. The pipeline is divided in build and deploy. When the pipeline runs successfully, it will look similar to what is shown in Figure 9-11.
When visiting the URL that Ansible deployed the application to, the node.js application will appear, as shown in Figure 9-12.

![Hello zCX User!](image)

**Figure 9-12  hello-node Application**

### 9.5 Monitor a pipeline in Jenkins

When a new pipeline is created in Jenkins, it appears on the overview page. Each pipeline is displayed as a row in a pipeline table. The first two columns visualize the health status of the pipeline, e.g. the sun tells us that the last 5 pipeline runs were successfully completed. The first column refers to the state of the most recent run of the pipeline and would be red if the pipeline fails. This can be viewed in Figure 9-13.

<table>
<thead>
<tr>
<th>S</th>
<th>W</th>
<th>Name</th>
<th>Last Success</th>
<th>Last Failure</th>
<th>Last Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DevOps_flow</td>
<td>23 min - #1</td>
<td>N/A</td>
<td>2 min 21 sec</td>
</tr>
</tbody>
</table>

**Figure 9-13  Jenkins Pipeline Overview**

Once the pipeline in Jenkins is configured and ready, a build can be triggered by pushing a new commit to the master branch of the Git repository. Alternatively, you can run the pipeline immediately by clicking the ‘Build Now’ button in the left side menu. This is shown in Figure 9-14.
In the center of the window, ‘Stage View’ is an area that displays the current status and progress of the stages defined in the pipeline. On the left bottom corner of Figure 9-14 you can see the build history. Build here means the run of the pipeline, not only a container build. You can click on one of the items in the build history list to see more details about a specific run. In Figure 9-15 you can see the overview of a build history item. The date in the title refers to the start date of the pipeline run.
In the build history is shown who has started the pipeline. Also you can see the ID of revision from the Git repository. On the left are various functions like status or console output. If the pipeline fails multiple times, it may help to have a look at the console outputs to identify the root cause of the issue. Figure 9-16 shows a typical snippet of a console output.

In the output, meta-information and container output are displayed. The dark grey text is output from the pipeline runner and the container itself (container output have a “>” prepended). Pipeline meta-information have a [Pipeline] prepended.
Figure 9-17 shows example output from the Ansible playbooks. As those are run and logged through the Ansible plug-in, the output is much more structured and shows the single tasks and their output.

```
PLAY [build and run the docker container]  *****************************************************
TASK [Gathering Facts]  *****************************************************
  ok: [localhost]  

TASK [Install pip]  *****************************************************
  ok: [localhost]  

TASK [Install docker-py]  *****************************************************
  ok: [localhost]  

TASK [Creating the container]  *****************************************************
  ok: [localhost]  

TASK [Running the container]  *****************************************************
  ok: [localhost]  

PLAY RECAP  *****************************************************
localhost : ok=5  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

```
Figure 9-17  Pipeline Ansible Output
```

**9.6 Summary**

In this part of this IBM Redbooks publication, we ran a simple DevOps flow using Gitea, Jenkins and Ansible in containers on zCX. zCX provides the ability to run Docker images in a z/OS system. This demonstrates the potential for customers to modernize their IT by adopting a DevOps solution that allows them to become more agile and efficient while maintaining and adding new features to z/OS applications and extending the value of z/OS assets to modern applications on IBM Z and to other platforms.

This is a very critical capability to run business applications in containers in a secure and cost-effective way. It enables resiliency for critical workloads.
Part 3

Monitoring and managing

In this part we discuss monitoring and managing your zCX systems.
Monitoring

This chapter describes a monitoring use case that uses IBM Service Management Unite (SMU) Automation. The full Enterprise Edition also supports the IBM Workload Scheduler and IBM Omegamon in addition to IBM System Automation.

Among the benefits of running IBM SMU inside of zCX:
- Provides the ability to use a modernized interface to several IBM management products.
- Has no external dependencies.
- IBM SMU is available if the z/OS system is IPLed in an alternate location.

This section covers the following topics related to IBM Service Management Unite Automation:
- Chapter 10.1, “IBM Service Management Unite” on page 212
- Chapter 10.1.1, “IBM Service Management Unite Automation” on page 212
- Chapter 10.2, “IBM Service Management Unite Overview” on page 213
- Chapter 10.4, “Network and port configuration” on page 216
- Chapter 10.5, “Start the SMU Image” on page 217
- Chapter 10.6, “Uninstalling Service Management Unite” on page 221
- Chapter 10.7, “Command wrapper” on page 223
10.1 IBM Service Management Unite

IBM Service Management Unite (SMU) Enterprise Edition is a customizable service management user interface that provides dashboards to monitor and operate IBM Z environments.

IBM Service Management Unite Enterprise Edition includes the following components:

▶ **IBM Service Management Unite Automation**
  Provides the overall health status of the IBM NetView® domains, automation domains and nodes, and easy access to automation functions to start, stop, or recycle business resources.

▶ **IBM Service Management Unite Performance Management**
  Helps you monitor and manage the performance of z/OS operating systems, network, storage, and other subsystems. You can quickly identify, isolate, and fix z/OS problems from a single point of control.

▶ **IBM Service Management Unite Workload Scheduler**
  Provides comprehensive support for workload automation and scheduling. You can monitor and manage the most critical resources like jobs, job streams, and workstations in your scheduling environment.

These components work together to empower the operations staff to analyze and resolve problems more quickly.

IBM Service Management Unite Automation Enterprise Edition provides the following capabilities:

▶ Provides a consolidated view of system health status, and thus reduces the time and effort in accelerating problem identification.

▶ Delivers simplified, efficient automation, workload scheduling, and network management capabilities, which streamlines operators' workflow.

▶ Provides an integrated operations console, which can be used to issue commands and resolve problems. It increases the degree of automation and avoids manual and time intensive tasks.

▶ Provides highly customized dashboard that helps you best suite your needs.

▶ Supports mobile access, which enables you to check your system anytime and anywhere.

10.1.1 IBM Service Management Unite Automation

IBM Service Management Unite Automation is a free of charge customizable dashboard interface that is available with:

▶ IBM Z System Automation V4.1.0 or higher
▶ IBM Z Service Management Suite
▶ IBM Z Service Automation Suite
▶ IBM Z NetView V6.3 or higher
▶ IBM Z Monitoring Suite

It provides a single point of control for multiple System Automation Plexes (SAPlex) to operate in your environment. Operators can quickly and confidently analyze, isolate, and diagnose problems by providing all relevant data including important logs in a single place.
IBM Service Management Unite Automation also enables operators to interact directly with the system by issuing commands and viewing results without going to a different console. Additionally, it allows the customization of your own dashboards, providing exactly the information needed by the operations in your specific environment. IBM Service Management Unite Automation can be installed on Linux on IBM Z and uses the end-to-end (E2E) automation adapter for secure communication with System Automation (SA) for z/OS.¹

10.2 IBM Service Management Unite Overview

In this section, we provide an overview of a typical IBM SMU installation. We have two z/OS LPARs that are running an end-to-end (E2E) adapter and both are connected to a single SMU instance, as shown in Figure 10-1.

![IBM SMU architecture](image)

The benefits of using SMU is the ability to use a modernized interface to several IBM management products without any external dependencies. SMU remains available for an end-user even when the SMU instance itself or the hosting LPAR is IPLed, as the instance can just be moved to the next zCX instance where the E2E adapters will connect to and offer the service seamlessly.

The following is the description of the single components shown in Figure 10-1.

- Service infrastructure

  Service Management Unite Automation uses a service infrastructure that incorporates key products and services to run the dashboards and provide flexible integration and

customization capabilities. The service infrastructure is provided with Service Management Unite Automation. The infrastructure consists of the following:

- IBM Dashboard Application Services Hub (DASH) / Jazz® for Service Management (JazzSM)

IBM Dashboard Application Services Hub (DASH) provides visualization and dashboard services based on Jazz for Service Management (JazzSM). The DASH integration platform supports data processing and content rendering from multiple sources. The data is integrated and presented in interactive dashboards. DASH has a single console for administering IBM products and related applications.

- IBM WebSphere Application Server

IBM WebSphere Application Server provides the application server runtime environment for DASH and the Service Management Unite Automation dashboards.

- Service Management Unite Automation

Service Management Unite Automation provides the dashboards to monitor and operate resources that are automated by IBM Z System Automation, issue z/OS and NetView commands, and access system logs. It also provides the Universal Automation Adapter to automate non-z/OS systems from IBM Z System Automation.

- Connectivity to backend systems

- Connect Service Management Unite Automation with z/OS systems

Use the following main components to interact with z/OS systems:

- IBM Z System Automation

IBM Z System Automation is a policy-based, self-healing, high availability solution. It maximizes the efficiency and the availability of critical systems and applications. It also reduces administrative and operational tasks.

- IBM Z System Automation end-to-end (E2E) adapter

The Z System Automation E2E adapter connects an SA z/OS domain to Service Management Unite Automation. It enables Service Management Unite Automation to read data like the status of automated resources and run actions like sending requests. It also provides the capability to issue NetView and z/OS commands and access system logs. In addition, the E2E adapter is used as the connection target by System Automation to provide cross-sysplex end-to-end automation.

- Connect Service Management Unite Automation with non-z/OS systems

Additionally, there are other ways to connect to non-z/OS systems with the setup of the Universal Automation Adapter (UUA). The Universal Automation Adapter (UAA) enables Service Management Unite Automation (SMU) to monitor, operate, and automate resources running on non-z/OS systems. It connects to the remote non-z/OS systems using a Secure Shell (SSH).

### 10.3 Installation steps

In this section, we discuss the steps necessary to install IBM Service Management Unite Automation.

1. Obtain the installation files

   a. Go to the download portal (http://ibm.biz/smu-auto-download) to download IBM Service Management Unite Automation installation files. You will need an IBM ID to log in, to sign up for an ID if you do not have one, access the following website:
The IBM Service Management Unite Enterprise Edition includes three components that you can install:

- IBM Service Management Unite Automation
- IBM Service Management Unite Performance Management
- IBM Service Management Unite Workload Scheduler

b. Provide the access key to get the installation files.
There are two options to obtain the access key:

i. If you have APAR OA56692 installed on IBM System Automation for z/OS, you can find the access key and additional download information on your z/OS system in dataset SINGSAMP(INGESMU).

ii. View the PDF which is supplied with the product materials on the CD titled "LCD8-2753-01 Accessing IBM Service Management Unite Automation CD-ROM".

c. Select the packages and click Download now to get the installation files.

d. If you prefer to install IBM Service Management Unite Automation using a prebuilt Docker image, select IBM Service Management Unite Automation - Docker image for Linux on System z depending on your system. The Docker image contains all the software prerequisites that you need to install IBM Service Management Unite Automation.

2. Installation steps

a. Logon to the the zCX instance and create the smu directory:

   ```
   mkdir smu
   ```

b. Upload the image that has been obtained in section 10.2, “IBM Service Management Unite Overview” on page 213 into the home directory of the admin user. Upload the image using SCP.

c. Once the upload has completed, the .tar file can be extracted into the newly created smu directory as shown in Example 10-1 on page 215.

   **Example 10-1   Extract the uploaded .tar file for SMU**

   ```
   admin@sc74cn11:~$ tar -xvf SMU_Automation_v1.1.7.1_Docker_Image_zLinux.tar --directory /home/admin/smu
   eezdocker.cfg
   eezdocker.sh
   readme_smu_auto.txt
   smu_image_v1171.tar
   util/
   util/licenses/
   util/licenses/LA_en
   util/licenses/LA_in
   util/migration.sh
   util/basic_docker.sh
   util/user_input.sh
   util/messaging.sh
   util/msg/
   util/msg/en/
   ```
Once the extract has completed successfully, you will not need the .tar file anymore. If space is limited, this can be deleted.

### 10.4 Network and port configuration

As a default configuration, the SMU Docker container uses a network bridge and maps the required ports to the Docker host system.

The Docker container's host name is set to the Docker host system's name. The Docker container opens and maps the ports shown in Table 10-1 on page 216 in listen mode for services that are offered by Service Management Unite.

#### Table 10-1 Default port information

<table>
<thead>
<tr>
<th>Port number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16311</td>
<td>Port to access the DASH that hosts the SMU dashboards</td>
</tr>
<tr>
<td>16316</td>
<td>Port to access the WebSphere administrative console</td>
</tr>
<tr>
<td>2002</td>
<td>Port that is used by automation adapters to connect to SMU and send update events for resources</td>
</tr>
<tr>
<td>2005</td>
<td>Port that is opened by the Universal Automation Adapter to receive requests from the E2E agent</td>
</tr>
</tbody>
</table>

If you want to restrict access to a port (for example, the WebSphere administrative console, port 16316), you will need to configure appropriate firewall rules on the host system.

**Note:** If you plan to use the Universal Automation Adapter (UAA) to manage resources that are running on the same host system where the SMU Docker container is running, a special configuration is required.

It is not possible to use the SMU Docker container host's host name as the node name in the UAA Policy because this host name will be resolved to the Docker container's internal IP address by the SMU Docker container - and not to the Docker host's IP address as required. To resolve this issue, you will need to use another host name other than the one that is set to the SMU Docker container, but resolving to the intended IP address.

In `ezdocker.cfg`, you can add `--add-host=<my_host_name>_host:<my_host_ip>` as an option to `DOCKER_NETWORK_CONFIG`. The Docker parameter `--add-host` allows you to introduce the host name IP address mappings for a Docker container. For example, if the host name where the SMU Docker container runs is `smu`, this will (inside the SMU Docker container) introduce the host name `smu_host`, which resolves to the same IP address as `smu`. In the UAA’s policy, you will specify the resource’s node to this new host name, `smu_host`. 


10.5 Start the SMU Image

Before starting the image, the image must be loaded as shown in Example 10-2 on page 217. The license agreement must be accepted, by selecting option 1 as indicated in blue.

Example 10-2  Load the SMU image

```
admin@sc74cn11:~$ /home/admin/smu/eezdocker.sh load
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+  IBM Service Management Unite Docker Command Line Utility +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Executing command load ...

Docker image smu_auto:1171 ---------------------------------- Not loaded.
You must accept the license agreement before the IBM Service Management Unite Docker image can be loaded.
To read the license agreement select 3 and toggle between the license files with :n (next) and :p (previous).
Exit the reader when you have finished with q.
If you want to accept the license, continue with 1.
If you want to decline the license, cancel with 2.
[Waiting for response] Accept license [ 1-accept or 2-decline or 3-view; decline is default ]? 1
License accepted. Continue loading the IBM Service Management Unite Docker image.
Loading image from local file /home/admin/smu/smu_image_v1171.tar.
550cfcb7b1d35: Loading layer [========================================]
66.07MB/66.07MB
0cf58cc46810: Loading layer [========================================]
991.7kB/991.7kB
bbed7c7060c0: Loading layer [========================================]
15.87kB/15.87kB
116be52951b9: Loading layer [========================================]
3.072kB/3.072kB
f7ccbe30c7ea: Loading layer [========================================]
490.2MB/490.2MB
d88865165dca: Loading layer [========================================]
249.9MB/249.9MB
9bca0be1df95: Loading layer [========================================]
2.496GB/2.496GB
65bcac0e23a3: Loading layer [========================================]
39.65MB/39.65MB
59b1e7003f8e: Loading layer [========================================]
10.24kB/10.24kB
d7b372ef49e6: Loading layer [========================================]
5.12kB/5.12kB
73626721a8a1: Loading layer [========================================]
2.56kB/2.56kB
4c7a4f0ed798: Loading layer [========================================]
201.4MB/201.4MB
979088069557: Loading layer [========================================]
444.3MB/444.3MB
d9470ba6ffe4: Loading layer [========================================]
124.7MB/124.7MB
Loaded image: smu_auto:1171
```
To start the image, issue the command shown in Example 10-3. As shown, the `eezdocker.sh` script checks first, if the container is already running and then, if the image is available to be started. If the start command is not executed the first time, the output differs a bit as the container is already available. This is shown in Example 10-4.

**Example 10-3  Start the SMU image for the first time**

```
admin@sc74cn11:~$ /home/admin/smu/eezdocker.sh start
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+         IBM Service Management Unite Docker Command Line Utility         +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Executing command start ...
Status of Docker container smu_auto_1171 ------------------- Not started.
Docker container smu_auto_1171 ------------------------------ Not available.
Docker image smu_auto:1171 ---------------------------------- Loaded.
Creating Docker Container smu_auto_1171 --------------------- Successful.
Starting Docker container smu_auto_1171 --------------------- Successful.
Status of Docker container smu_auto_1171 ------------------- Started.
```

**Example 10-4  Start the SMU image not the first time**

```
admin@sc74cn11:~$ /home/admin/smu/eezdocker.sh start
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+         IBM Service Management Unite Docker Command Line Utility         +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Executing command start ...
Status of Docker container smu_auto_1171 ------------------- Not started.
Docker container smu_auto_1171 ------------------------------ Available.
Starting Docker container smu_auto_1171 --------------------- Successful.
Status of Docker container smu_auto_1171 ------------------- Started.
```

Example 10-5 demonstrates what happens when the start parameter is used when SMU is already running.

**Example 10-5  Start the SMU image when already started**

```
admin@sc74cn11:~$ /home/admin/smu/eezdocker.sh start
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+         IBM Service Management Unite Docker Command Line Utility         +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Executing command start ...
Status of Docker container smu_auto_1171 ------------------- Started.
```

Once SMU is started, it might take a few seconds until the built in Websphere Application Server is started. Once it is started, you can access the SMU Dashboard with the following link:

```
https://<zCX_DNS>:16311/ibm/console
```
Accessing the website with a browser will show a login page similar to that shown in Figure 10-2 on page 219.

![IBM SMU Dashboard logon page](image)

The default user ID and password to logon is eezadmin/eezadmin.

To access the WebSphere administrative console you can either access it via this link:

https://<zCX_DNS>:16316/ibm/console

Or when logged on the Dashboard, you can follow the two steps as shown in Figure 10-3 on page 220.
From the main menu of the IBM Service Management Unite Automation, you will find information from all connected System Automation Plexes (SAPlex). The configuration details and all prerequisites can be found at the following link: https://www.ibm.com/support/knowledgecenter/SS9U8J_1.1.7/com.ibm.smu.doc/CMIS/SA/toconf_configuring.html.

When the SA configuration is completed and you select “Domain and Automation Health”, you will get an overview page on the connected Domains and Systems as shown in Figure 10-4 on page 221.
10.6 Uninstalling Service Management Unite

To uninstall Service Management Unite, remove the SMU Docker image and all SMU Docker containers from the host system.

1. Any SMU Docker container must be stopped before the uninstallation. Issue the command shown in Example 10-6 to stop the SMU Docker container.

   **Example 10-6  Stop the SMU image**

   ```
   admin@sc74cn11:$ /home/admin/smu/eezdocker.sh stop
   +---------------------------------+
   + IBM Service Management Unite Docker Command Line Utility +
   +---------------------------------+
   
   Executing command stop ...
   
   Status of Docker container smu_auto_1171 ------------------------ Started.
   Stopping Docker container smu_auto_1171 ------------------------ Successful.
   Status of Docker container smu_auto_1171 ------------------------ Not started.
   ```
2. Issue the command shown in Example 10-7 to remove the SMU Docker container and Docker image. Confirm the deletion as shown in red in the example to remove all customer data.

**Example 10-7  Uninstall SMU**

```
admin@sc74cn11:$ /home/admin/smu/eezdocker.sh uninstall
+---------------------------------+---------------------------------+
| IBM Service Management Unite Docker Command Line Utility | +---------------------------------+
| Executing command uninstall ... | +---------------------------------+
| Uninstalling SMU from the Docker environment will delete the existing Docker image smu_auto:1171 and container smu_auto_1171. | +---------------------------------+
| [Warning] ALL CUSTOM CONFIGURATION WILL BE LOST! | +---------------------------------+
| [Waiting for response] Perform deletion [ 1-yes or 2-no; no is default ]? 1 | +---------------------------------+
| Confirmed. Continue with deletion. | +---------------------------------+
| Docker container smu_auto_1171 ------------------------------ Available. | +---------------------------------+
| Status of Docker container smu_auto_1171 ---------------------- Not started. | +---------------------------------+
| Deleting Docker container smu_auto_1171 ----------------------- Successful. | +---------------------------------+
| Docker image smu_auto:1171 ----------------------------------- Loaded. | +---------------------------------+
| Untagged: smu_auto:1171 | +---------------------------------+
| Deleted: sha256:740fa7b486975d77c9713423841d5fbc8a71dfafdf80bfc15cb7a550123f38bb32 | +---------------------------------+
| Deleted: sha256:c0da71e660f34f0e9ea15bdac7b33f8a868114477b515c7fba621b11eb2533b | +---------------------------------+
| Untagged: smu_auto_1171_running | +---------------------------------+
| Deleted: sha256:c03943c5f927a24ba3a094ae360ba9981033b0301d93834a1aa1b7b1882de87 | +---------------------------------+
| Deleted: sha256:066b48e80efdc5ca8b83aceb1d63d93bbf4144db00e2389557052356261cf8 | +---------------------------------+
| Deleted: sha256:738b80f0e0af373cc266303b3917d3e156f68a6ecd6f88ef2c781b89aa3105 | +---------------------------------+
| Deleted: sha256:b221b75c47c4c18c6f19f19e5c7475fd7a3681cfece7918ebe1be5f3738cc7 | +---------------------------------+
| Deleted: sha256:4a06aba3b0ab51d5fca51b44d585474649b9493d92a00aee32776b366bd95d | +---------------------------------+
| Deleted: sha256:782eef30a4e8f3bd5861b1e2f72ce73a127db4cf4edc316666763029b4c802ca | +---------------------------------+
| Deleted: sha256:0151f8a879f90324d60314d65b99799139c56e7cbfbd72ab132306ea35b8a | +---------------------------------+
| Deleted: sha256:6716c7502a1e1595635a262a74bdcba976db6c61617785672a5d7021a306425 | +---------------------------------+
| Deleted: sha256:89ae5add42ed249c086f5c091198d40b6d595ae7c36075dfbe3e462bf019d | +---------------------------------+
| Deleted: sha256:78b28590308e6692ba741262a34bb2160593d4b47b34f8470a791049602e958 | +---------------------------------+
| Deleted: sha256:59f3e4529c2e2a561c09b256a984a41418120a7d672f8369a7b25e0284eeb | +---------------------------------+
| Deleted: sha256:531c74d4ec5d4466c63dede17cf7913c78198961bae2df3efca818958ded | +---------------------------------+
| Deleted: sha256:20b31cdded976b3672b3e09f86f2f6654916c44a9afdbdb6976a302c03ec34f54 | +---------------------------------+
| Deleted: sha256:550cfc7b1d35a3a70b83418a9a8692eaca1e41d7939c6ccc6d442e4a43620c223 | +---------------------------------+
```
Deleting Docker image smu_auto:1171 ------------------------- Successful.
Uninstalling SMU from the Docker host -------------------------- Successful.

3. At this point, the SMU instance is completely removed. For the final cleanup, remove the whole smu directory created in Chapter 2, “Installation steps” on page 215.

10.7 Command wrapper

There is a command wrapper available to manage an SMU instance.

When you download the SMU Docker archive, extract the package, and initially run command `ezdocker.sh load`, the SMU Docker image is loaded into your local Docker environment.

The image is like a blueprint for you, containing SMU and all its prerequisites but missing your custom configuration. To use SMU in your environment, you need to create a Docker container from the SMU Docker image. A container is a concrete, runnable instance of an image. Theoretically, you can create more than one container instance from the same image in parallel and configure each container individually.

When you first run the `ezdocker.sh start` command, the SMU Docker Command Line Utility automatically creates a new SMU Docker container from the SMU Docker image for you. The created Docker container can be started and stopped as often as you like and also survives from a restart of the Docker environment or a restart of the host system.

Unless you explicitly delete the SMU Docker container (`ezdocker.sh reset` or `ezdocker.sh uninstall`), the SMU Docker Command Line Utility operates on the same SMU Docker container instance. For example, if you stop the SMU Docker container (`ezdocker.sh stop`) and start it again (`ezdocker.sh start`), it will be the same SMU Docker container instance.

Every SMU and WebSphere Application Server configuration change that you perform on a running SMU Docker container is stored within this container instance, but not in the SMU Docker image. For example, if you create a custom dashboard in DASH, the change is stored within the SMU Docker container and will be there until the container is deleted. The dashboard will also still be there if the SMU Docker container is restarted or even if the host system is restarted.

Therefore, the easiest way to reset SMU to factory defaults is to delete the SMU Docker container and create a new one from the SMU Docker image (`ezdocker.sh reset`).

In SMU V1.1.4, Docker volumes are used to store the SMU and WebSphere Application Server configuration outside of the SMU Docker container in a specific directory on the host system. From SMU V1.1.5, the SMU Docker container does not use any Docker volumes anymore. All configuration is stored within the SMU Docker container and won't get lost unless you delete the container.

In addition, a migration command is provided that allows you to migrate your custom configuration from an SMU Docker container of an old version to a new version.

10.7.1 Commands provided by the SMU Docker command line utility

In this section, we outline some useful commands that can be used with the command line utility.
**Important Note:** As the path used to upload the files (in our example, /home/admin/smu) is not part of the $PATH configuration of the zCX Instance, the `eezdocker.sh` script must be called with an absolute or relative path such as `./eezdocker.sh <parm>` or by adding the `/home/admin/smu` path into the $PATH list.

```
eezydocker.sh load  Loads the IBM provided SMU Docker image into the local Docker environment. One only needs to run this command once. If the image is successfully loaded, one can delete the .tar file.

neezydocker.sh start Starts the SMU Docker container if it is stopped. If no SMU Docker container exists, it creates a new SMU Docker container from the loaded SMU Docker image.

neezydocker.sh stop Stops the running SMU Docker container.

neezydocker.sh restart Restarts the running SMU Docker container. It stops the SMU Docker container and starts it again. For example, one can run the command if needed to restart the WebSphere Application Server after a configuration change.

neezydocker.sh shell Opens a Bash shell to the running SMU Docker container. It allows to access the internal of the container, for example, if the configuration files need to be edited manually. To exit the shell, issue command `exit` in the shell. It only exits the shell connection into the SMU Docker container, and the container and SMU continue to run.

neezydocker.sh cfgsmu Starts the SMU Automation `cfgsmu` tool and sets up the necessary X-Forwarding so that the tool's GUI can be displayed with the host's Window Manager.

   If `cfgsmu` cannot be run out of the Docker container, it might be necessary to allow access to the X11 session on the host system. Run command `xhost+local:all` before running `neezydocker.sh cfgsmu` to ensure that the Docker process can access the user's X session.

neezydocker.sh collect_logs Collects and bundles all relevant log files from the running SMU Docker container and copies them to the host system's /tmp folder.

   For example, run this command if facing a problem and the IBM Support team requests the log information.

neezydocker.sh reconfigure Some Docker configurations can only be specified during the creation of the SMU Docker container, for example the network configuration. If you need to change a configuration option, issue this command to make the configuration changes take effect.

   Internally, the current SMU Docker container is committed to a snapshot of the SMU Docker image, from which a new container is created. The new container has all the same configurations and customizations of the old container, but runs with the new configuration. When you issue the command `neezydocker.sh reconfigure`, a new snapshot is created. Keep the snapshot image because an image cannot be removed if there are containers derived from it. The image doesn't take up too much disk space because only the changes, compared to the official SMU Docker image, are stored in it.

neezydocker.sh migrate Migrates all of the custom configuration from the old SMU Docker container into the new container of a new SMU release,
**eezdocker.sh reset**  Deletes the current SMU Docker container. If one runs command **eezdocker.sh start** afterwards, a new SMU Docker container is created.

**Warning:** All custom configuration will get lost! Only run this command if you need to reset to factory defaults.

**eezdocker.sh uninstall**  Deletes the current SMU Docker container and the SMU Docker image.

**Warning:** All custom configuration will get lost! Run the command if you need to remove SMU from your Docker environment.
Chapter 11. Container restart policy

After your Docker containers are set up and running in production, you need to configure some Docker policies in order to restart containers after particular events such as failures or if the zCX server itself is restarted for a planned change. There are four restart policies that will help you to automatically restart your containers when they exit.

In this section, we provide guidelines to change the container restart policy to ensure all of your containers are up and operational when the zCX server is restarted.
11.1 Introduction to Docker restart policies

Containers are inherently ephemeral. They are routinely destroyed and rebuilt from a previously pushed application image. Another specific characteristic is the ability to be quickly stopped or killed that makes container easy to maintain. But containers can be reloaded shortly after their termination in no-time, within milliseconds.

It is recommended when running application containers in production, an update for the restart policy to ensure application is up and running after a zCX restart of an unexpected application failure.

Docker considers any containers to exit with a non-zero exit code to have crashed. By default a crashed container will remain stopped.

According to the official Docker documentation, the restart policies can be changed by supplying the `--restart` command line option:

```
docker run --restart=always --name <container_name> -d <image_name>
```

The following four options can be specified with the `--restart` flag, based on the required policy. These options effect how the container starts at boot as well.

The flags are described in Table 11-1:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>Do not automatically restart the container. (the default)</td>
</tr>
<tr>
<td>on-failure</td>
<td>Restart the container if it exits due to an error, which manifests as a non-zero exit code</td>
</tr>
<tr>
<td>always</td>
<td>Always restart the container if it stops. If it is manually stopped, it is restarted only when Docker daemon restarts or the container itself is manually restarted. (See the second bullet listed in restart policy details)</td>
</tr>
<tr>
<td>unless-stopped</td>
<td>Similar to always, except that when the container is stopped (manually or otherwise), it is not restarted even after Docker daemon restarts.</td>
</tr>
</tbody>
</table>

For the examples in this book, we supplied `--restart=always` so that a container will always be restarted after the zCX instance is restarted. Please keep in mind that when a container is manually stopped using the command `docker stop <container_name>`, the restart policy will not be applied to the container.

You should consider these options in order to better control container restarts on zCX.

In the following steps, we configure a Docker container on our zCX instance to demonstrate an example for the Docker restart policy and how it is important for production environments. We will create a special container that will print a message and then exit with code 1 to simulate a crash.

4. Issue the following commands to create a folder for our restart policy demonstration:
   ```
   mkdir docker-restart-example
   cd docker-restart-example
   ```

5. Create a file named `docker-crash.sh` with the following content:
#/bin/bash
 echo $(date)  "Booting up..."
 sleep 10
 exit 1

This script will print a message, wait for 10 seconds and exit with code 1 indicating an error.

6. Create the file Dockerfile.policy with the following content:

```
FROM ubuntu:latest
ADD docker-crash.sh /
CMD /bin/bash /docker-crash.sh
```

7. Now build the new image by using the following command:

```
docker build -t docker-restart-example-img -f Dockerfile.policy .
```

You will receive the following output:

```-
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM ubuntu:latest
---> f8835575bd80
Step 2/3 : ADD docker-crash.sh /
---> 447fe06defa
Step 3/3 : CMD /bin/bash /docker-crash.sh
---> Running in 9f48429b6d5c
Removing intermediate container 9f48429b6d5c
---> cba8fc357912
Successfully built cba8fc357912
Successfully tagged docker-restart-example-img:latest
```

8. Launch a container instance named app-restart-default by using the following command:

```
docker run -d --name app-restart-default docker-restart-example-img
```

9. Issue the command `docker ps` to check the status of the app-restart-default container and confirm it exited and it is not running. See Example 11-1.

```
Example 11-1   Output of docker ps command
```

<table>
<thead>
<tr>
<th>CONTAINER ID</th>
<th>IMAGE</th>
<th>COMMAND</th>
<th>CREATED</th>
<th>STATUS</th>
<th>PORTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>a7503f1d5c08</td>
<td>jenkins-s390x</td>
<td>/sbin/tini -- /usr/...</td>
<td>24 hours ago</td>
<td>Up 24 hours</td>
<td>0.0.0.0:8080-&gt;8080/tcp, 0.0.0.0:50000-&gt;50000/tcp - jenkins-app</td>
</tr>
<tr>
<td>5bac28336b8b</td>
<td>sshd-img</td>
<td>/usr/sbin/sshd -D</td>
<td>26 hours ago</td>
<td>Up 26 hours</td>
<td>0.0.0.0:10000-&gt;10000/tcp, 0.0.0.0:32769-&gt;22/tcp - sshd-app</td>
</tr>
<tr>
<td>5e09f5fc924d</td>
<td>ubuntu</td>
<td>/bin/bash</td>
<td>36 hours ago</td>
<td>Up 36 hours</td>
<td></td>
</tr>
<tr>
<td>1e2b54213f2b</td>
<td>portainer/portainer:linux-s390x</td>
<td>/portainer</td>
<td>9 months ago</td>
<td>Up 4 minutes</td>
<td>0.0.0.0:9000-&gt;9000/tcp</td>
</tr>
<tr>
<td>31df3838c951</td>
<td>ibm_zcx_zos_cli_image</td>
<td>sudo /usr/sbin/sshd_...</td>
<td>9 months ago</td>
<td>Up 36 hours</td>
<td>8022/tcp, 0.0.0.0:8022-&gt;22/tcp</td>
</tr>
</tbody>
</table>

```
admin@31df3838c951:~$ docker ps
```
10. Issue the `docker logs app-restart-default` command to check the logs and confirm application was started but exit after 10s. You should receive the following message:

```
Sat Jun 27 11:22:32 UTC 2020 Booting up...
By default, if an application that is running within a container crashes, the container stops and it will remain stopped until someone or something restarts it.
```

11. Depending on the application failure, restarting a failed process might correct the problem. Docker can automatically retry to launch Docker a specific number of times before it stops trying. In the following example, we will create a container with the option `--restart=on-failure:3` where the number 3 indicates how many times Docker should try to restart before stopping the container:

```
docker run -d --name app-restart-3 --restart=on-failure:3
docker-restart-example-img
```

If you run the `docker ps` command, you will see app-restart-3 container in the running container list. However, after some seconds, the container will exit with a return code 1 and stop.

Issue the command, `docker logs app-restart-3` and check for boot messages as shown in Example 11-2.

**Example 11-2  Output of docker logs command**

```
Sat Jun 27 11:33:49 UTC 2020 Booting up...
Sat Jun 27 11:34:05 UTC 2020 Booting up...
Sat Jun 27 11:34:25 UTC 2020 Booting up...
Sat Jun 27 11:34:40 UTC 2020 Booting up...
```

12. Finally, the following command will use the 'always restart' option, in this case, Docker will keep trying until someone or something explicitly stops it. To demonstrate this option, issue the following command:

```
docker run -d --name app-restart-always --restart=always
docker-restart-example-img
```

13. Wait a few moments and issue the command, `docker logs app-restart-always` many times to see the restart attempting via the logs as shown in Example 11-3.

**Example 11-3  Docker logs output for always restart flag**

```
Sat Jun 27 11:44:15 UTC 2020 Booting up...
Sat Jun 27 11:44:33 UTC 2020 Booting up...
Sat Jun 27 11:44:49 UTC 2020 Booting up...
Sat Jun 27 11:45:08 UTC 2020 Booting up...
Sat Jun 27 11:45:22 UTC 2020 Booting up...
Sat Jun 27 11:45:38 UTC 2020 Booting up...
Sat Jun 27 11:45:53 UTC 2020 Booting up...
Sat Jun 27 11:46:10 UTC 2020 Booting up...
Sat Jun 27 11:46:27 UTC 2020 Booting up...
Sat Jun 27 11:46:45 UTC 2020 Booting up...
Sat Jun 27 11:47:00 UTC 2020 Booting up...
Sat Jun 27 11:47:26 UTC 2020 Booting up...
Sat Jun 27 11:47:43 UTC 2020 Booting up...
```
14. Do not forget to delete the example container using `docker rm --force app-restart-always` command

## 11.2 Considerations

As detailed in this chapter, four restart options are available to change the behavior of how Docker will handle a container failure or unforeseen conditions. Using an adequate policy can assure your container is up and running in your environment even after a planned zCX outage.

You need to review each option and select the one that best applies to your application environment.

For detailed information, see: https://docs.docker.com/config/containers/start-containers-automatically/
Additional material

This refers to additional material that can be downloaded from the Internet as described in the following sections.

Locating the web material

The web material associated with this is available in softcopy on the Internet from the IBM Redbooks web server:
ftp://www.redbooks.ibm.com/redbooks/SG248471/

Alternatively, you can go to the IBM Redbooks website:
ibm.com/redbooks

Search for SG248471, select the title, and then click Additional materials to open the directory that corresponds with the form number, 8471.

Using the web material

The additional web material that accompanies this includes the following files:

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DevOps_add_mat.zip</td>
<td>Contains folders listed as Agent, Gitea, and hello-node-app_files. Each folder contains sample images to use as you follow along in this book.</td>
</tr>
<tr>
<td>CicsToKafkaProjects.zip</td>
<td>Contains folders listed as CatalogEvents, CicsToKafkaBunde and CicsToKafkaDemo. Each folder contains</td>
</tr>
</tbody>
</table>

Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and extract the contents of the web material .zip file into this folder.
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- **Getting started with z/OS Container Extensions and Docker**, SG24-8457
- **????full title???????, SG24-xxxx**
- **????full title???????, REDP-xxxx**
- **????full title???????, TIPS-xxxx**

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Other publications

These publications are also relevant as further information sources:

- [https://ibm-cloud-architecture.github.io/refarch-eda/kafka/readme/](https://ibm-cloud-architecture.github.io/refarch-eda/kafka/readme/)
- [https://ibm-cloud-architecture.github.io/refarch-eda/kafka/readme/](https://ibm-cloud-architecture.github.io/refarch-eda/kafka/readme/)
- [https://zookeeper.apache.org/doc/current/zookeeperStarted.html#sc_RunningReplicatedZooKeeper](https://zookeeper.apache.org/doc/current/zookeeperStarted.html#sc_RunningReplicatedZooKeeper)

Online resources

These websites are also relevant as further information sources:

- Description1
  - [http://?????????.???.???.??/?](http://?????????.???.???.??/?)
- Description2
  - [http://?????????.???.???.??/?](http://?????????.???.???.??/?)

© Copyright IBM Corp. year.
Help from IBM

IBM Support and downloads
ibm.com/support

IBM Global Services
ibm.com/services
To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine Width for the book and hide the others:

```
Special>Conditional Text>Show/Hide>SpineSize(-->Hide:)>Set
```

Move the changed Conditional text settings to all files in your book by opening the book file in 5.0.7 or later and File>Import>Conditional Text Settings (ONLY!) to the book files.
To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the spine width for the book and hide the others:

Special>Conditional Text>Show/Hide>SpineSize(-->Hide:) Set

Move the changed Conditional text settings to all files in your book by opening the book file with the spine.fm still open and File>Import>Formats Conditional Text Settings (ONLY!) to the book files.