

Red Hat OpenShift Container Platform for IBM zCX

Lydia Parziale

Andy Armstrong

Vic Cross

Maike Havemann

Redelf Janssen

Pablo Paniagua

Krzysztof Dzialek

Ravi Kumar



 **Cloud**

IBM Z



IBM Redbooks

Red Hat OpenShift Container Platform for IBM zCX

October 2022

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

First Edition (October 2022)

This edition applies to IBM zCX Foundation for Red Hat OpenShift, IBM z16 and z/OS 2.4 and 2.5.

© Copyright International Business Machines Corporation 2022. All rights reserved.

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

Contents

Notices	v
Trademarks	vi
Preface	vii
Authors	vii
Now you can become a published author, too!	viii
Comments welcome	ix
Stay connected to IBM Redbooks	ix
Chapter 1. zCX Foundation for Red Hat OpenShift	1
1.1 Introduction	2
1.1.1 Benefits	2
1.2 Getting Started	3
1.2.1 Prerequisites	3
1.2.2 High availability and Disaster Recovery	7
1.2.3 Administration and operations	8
1.2.4 Migration for modernization	10
1.3 Lab environment overview	11
1.3.1 Hardware	11
1.3.2 Operating system	12
1.3.3 Lab infrastructure	13
1.3.4 IBM Spectrum Scale	13
1.3.5 Licensing	14
Chapter 2. Use case 1: Using z/OS Connect inside Red Hat OpenShift	15
2.1 Introduction	16
2.1.1 Overview	16
2.1.2 Problem statement	17
2.2 Solution	18
2.2.1 Solution architectural overview	19
2.2.2 Installation overview	20
2.3 Summary	30
Chapter 3. Use case 2: Artificial intelligence and open source tools	31
3.1 Background	32
3.2 Problem being addressed	34
3.2.1 Challenges facing data scientists	34
3.3 Solution description	35
3.3.1 Deploying Linux on Z containers to zCX on OpenShift	36
3.4 Benefits	36
Chapter 4. Use case 3: Migrating a solution from zCX Docker to Red Hat OpenShift in zCX	39
4.1 Differences between Docker and Red Hat OpenShift	40
Chapter 5. Your use case: Run anything on Red Hat OpenShift in zCX	43
5.1 Requirements	44
5.2 Worked example	45
5.2.1 What is needed	45
5.2.2 Preparing our application components for deployment to containers on Red Hat	

OpenShift on zCX	46
5.3 Creating a multi-architecture Containerfile and image for an x86 image to deploy on s390x architecture hosting platforms	47
5.3.1 Creating a s390x Containerfile and image for our NodeJS OCI-Compliant container front-end application	47
5.3.2 Sharing your image to a repository	48
5.3.3 More complex examples of Containerfiles	48
5.3.4 Containerizing an application from scratch	49
5.4 Managing the lifecycle of a containerized deployment	50
5.5 Summary	51
Appendix A. Red Hat OpenShift deployment under zCX	53
Red Hat OpenShift Container Platform installation	54
Installation source for Red Hat OpenShift Container Platform	54
Components outside z/OS and zCX	55
DNS resolution	55
Load balancer	55
Network Time Protocol server	56
Mirror registry	56
Our installation environment	57
Installation support node	57
Networking	57
Load balancer considerations	59
Building Red Hat OpenShift Container Platform on zCX	59
Using manual z/OSMF workflows to install Red Hat OpenShift Container Platform	60
Deploying our first node: A Red Hat OpenShift Container Platform bootstrap node	60
Installation tasks outside z/OS	65
Automatically using Ansible	66
Related publications	73
IBM Redbooks	73
Other publications	73
Online resources	73
Help from IBM	74

Notices

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

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

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

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

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

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

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

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

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

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

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

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


COPYRIGHT LICENSE:

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

Trademarks

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

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

CICS®	IBM Z®	VTAM®
Db2®	IBM z14®	WebSphere®
GDPS®	IBM z16™	z Systems®
HyperSwap®	OS/390®	z/OS®
IBM®	Parallel Sysplex®	z/VM®
IBM Cloud®	RACF®	z15™
IBM Consulting™	Redbooks®	z16™
IBM Spectrum®	Redbooks (logo)  ®	

The following terms are trademarks of other companies:

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

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.

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

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

Preface

Application modernization is essential for continuous improvements to your business value. Modernizing your applications includes improvements to your software architecture, application infrastructure, development techniques, and business strategies. All of which allows you to gain increased business value from existing application code.

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

This IBM Redbooks® publication presents the capabilities of IBM zCX along with several use cases that demonstrate Red Hat OpenShift Container Platform for IBM zCX and the application modernization benefits your business can realize.

Authors

This book 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 technology integrations, software development, project leadership, and strategic planning. Her areas of expertise include business development and database management technologies. Lydia is a PMI certified PMP, an IBM Certified IT Specialist, and an Open Group Master Certified IT Specialist with an MBA in Technology Management. She has been employed by IBM for over 30 years in various technology areas.

Andy Armstrong is a Principal Specialist for IBM Z Technologies, an EMEA Run Leader, and Master Inventor for Global Sales and Infrastructure Sales.

Vic Cross is an IT Solutions Engineer in Brisbane, Australia. Vic works as part of the IBM Worldwide zAcceleration Team, helping customers in adopting new technologies, such as Red Hat OpenShift Container Platform on IBM Z and LinuxONE. Previously, Vic worked with IBM Systems Lab Services and IBM Systems, providing senior design and implementation expertise to IBM Z and IBM LinuxONE projects across Asia-Pacific. He holds a degree in Computer Science from Queensland University of Technology, and has 30 years of experience in general IT, with 25 years on the IBM Z and IBM LinuxONE platforms and their antecedents. Vic has written and contributed to several IBM Redbooks publications, including *The Virtualization Cookbook for IBM Z Volume 1: IBM z/VM 7.2*, SG24-8147, and *Securing Your Cloud: IBM z/VM Security for IBM z Systems and LinuxONE*, SG24-8353.

Maik Havemann is a Client Technical Specialist for systems hardware in Berlin, Germany. She has more than 7 years of experience in the IT industry and within IBM. Her areas of expertise include IBM Z and modern technologies in the field of containers and cloud services.

Redelf Janssen is a Client Technical Specialist at IBM Z hardware sales in Bremen, Germany. He holds a degree in Computer Science from the University of Bremen and joined IBM in 1988. He is responsible for supporting IBM Z customers in Germany. His areas of expertise include IBM Z hardware, z/OS, mainframe simplification, storage management, and availability management. He has written IBM Redbooks publications about several IBM OS/390®, z/OS, and z/OSMF releases. He was also one of the co-authors of the first zCX Redbooks publication, *Getting started with z/OS Container Extensions and Docker*, SG24-8457. He also teaches classes on z/OS and z/OSMF.

Pablo Paniagua is an IBM Z Hybrid Cloud Client Technical Specialist in Spain. He has 5 years of experience in Red Hat OpenShift and LinuxONE. He holds a degree in Telecommunications Engineering from Universidad Pontificia de Comillas. Before IBM, Pablo worked as an IT consultant on distributed systems. During his time at IBM, he has helped many customers with their Linux on Z environments and Hybrid Cloud Strategies.

Ravi Kumar is a Thought Leader Level 3 Certified Technical Specialist at the IBM Consulting™ Organization, from Dallas, Texas. He holds Master of Business Administration (MBA) degree from the University of Nebraska-Lincoln, USA. He has 30 years of experience in IT consulting with various Fortune 500 companies. He is currently working with multiple customers in North America, leading their Hybrid Cloud Transformation journey and focusing on Mainframe Modernization. He has written extensively in the areas of Digital Transformation, Db2®, Data Analytics, and AI.

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

Robert Haimowitz
IBM Redbooks, Poughkeepsie Center

Patrik Hysky
IBM Systems Technical Sales Services, Austin

Tom Ambrosio, Bill Lamastro,
IBM CPO

Charlee Boyle, Anthony Papageorgiou, Ahilan Rajaveda, Anna Shugol, Andrew Smithson
IBM

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

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, IBM Redbooks
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on LinkedIn:

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

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

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

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

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



zCX Foundation for Red Hat OpenShift

In this chapter, we discuss how to get started with the zCX Foundation for Red Hat OpenShift.

We also include a basic overview of the prerequisites, administration, and operation aspects and an overview of our lab environment with a Red Hat OpenShift cluster on zCX that we used in the writing of this IBM Redbooks publication.

This chapter includes the following topics:

- ▶ “Introduction” on page 2
- ▶ “Getting Started” on page 3
- ▶ “Lab environment overview” on page 11

1.1 Introduction

Enterprises increasingly embrace the use of hybrid workloads. Accordingly, the z/OS software system must expand and enable cloud-native workloads on z/OS. With the introduction of zCX, we offer the possibility to deploy Linux on Z applications as Docker containers in a z/OS system.

By announcing zCX for Red Hat OpenShift, zCX was expanded and Red Hat OpenShift's enterprise-level container orchestration and management capabilities to z/OS were extended.

We also can extend and modernize our native z/OS system through an agile and flexible deployment of Linux on Z applications in a self-container Red Hat OpenShift cluster while benefiting from z/OS Qualities of Service. This ability allows applications to become inherently cloud enabled with Red Hat OpenShift while maintaining operational control within z/OS. z/OS Qualities of Service also is extended to their applications and workloads.

The use of Red Hat OpenShift on z/OS by way of zCX feels the same as it does on any other Red Hat OpenShift platform. Provisioning Red Hat OpenShift also should feel the same as provisioning other z/OS middleware components on z/OS.

The ability to easily extend z/OS applications and workloads by allowing existing or new z/OS applications to use services that are not available under z/OS today is a key benefit to IBM Z customers. For non-Linux customers, this the ability to use services that are not available under z/OS can be a starting point for introducing new technology to z/OS with access to a large system of open source and Linux on Z workloads that can be colocated on the z/OS platform. Proximity of Linux software (because of colocation) to z/OS transaction data enables systems operational control and use of z/OS platform benefits.

1.1.1 Benefits

Applications and services transparently benefit from the z/OS QoS. Workloads in zCX for Red Hat OpenShift benefit from high availability and Disaster Recovery (DR) planning by way of features, such as IBM HyperSwap®, storage replication, and IBM Geographically Dispersed Parallel Sysplex® (GDPS®). Applications within zCX also can use z/OS workload management capabilities for capacity planning and tuning.

zCX for Red Hat OpenShift also helps overcome cross-platform operational challenges by managing and servicing the entire software appliance. It poses less burden on operations and dependency management because no requirements exist for systems programmers and Red Hat OpenShift Application developers to acquire new skills.

The Red Hat OpenShift Container Platform is an important technology within IBM's hybrid cloud approach. It is one of the most used container environments, and is available and used on all platforms.

Red Hat OpenShift is not only for developing, it is more. It is about orchestration and servicing those environments, from a full stack automation to a full stack continuous integration and deployment pipeline.

Red Hat OpenShift provides a few key advantages, including the following examples:

- ▶ An automated, full-stack installation is provided.
- ▶ Red Hat OpenShift provides a seamless Kubernetes deployment.
- ▶ Updates for the platform, services, and applications are possible with one click.

Red Hat OpenShift is the foundation. It provides the base with more components available, such as lifecycle management and monitoring.

1.2 Getting Started

The scope of this IBM Redbooks publication does not include a full tutorial about how to install and manage the zCX Foundation for Red Hat OpenShift. However, this section provides a basic understanding of what is needed to get started.

Note: For more information about how to get started and the prerequisites that must be met, see this [IBM Support web page](#).

The [zCX Foundation for Red Hat OpenShift](#) web page contains a tutorial about how to install and maintain zCX Foundation for Red Hat OpenShift. It also includes useful technical resources with an in-depth description of all pertinent considerations.

The installation is done by way of a user-provisioned infrastructure and mixes information from an installation of zCX standalone and a Red Hat OpenShift installation. The installation process includes the following overall steps:

1. Create the user provisioned infrastructure where the user configures IP networking and connectivity for the Red Hat OpenShift Nodes, and configure Load Balancers and DNS.
2. Create the configuration and ignition files as with any other Red Hat OpenShift installation. zCX Foundation for Red Hat OpenShift requires ignition files that are generated in this step.
3. Create a CoreOS base system. This step is done through z/OSMF Workflows, which result in the generation of five (or any other number defined) zCX nodes for Red Hat OpenShift.
4. Wait for installation to complete. After the zCX nodes start and can access their ignition files, they start the configuration process, which completes automatically in a few minutes.
5. Configure Red Hat OpenShift. After the installation is complete, the user can access the Red Hat OpenShift console and perform configurations, such as registry, users, and proxies.

After this process completes, a fully operating cluster is available for the user.

1.2.1 Prerequisites

In this section, we discuss the requirements, graphics, hardware, and software that are required for running the zCX Foundation for Red Hat OpenShift in your IBM Z environment.

Hardware

An IBM z14®, z15™ or z16™ is needed to implement and run zCX for Red Hat OpenShift successfully.

Software

z/OS V2R4 or V2R5 is needed as the underlying operating system to run zCX for Red Hat OpenShift with a recommended maintenance level. Each release should have a current maintenance level.

In addition to the base z/OS components, such as JES2, DFSMS, Communication Server (TCP/IP), SAF-based security (IBM RACF®), you must ensure that z/OSMF is configured and running. For zCX for Red Hat OpenShift, you need the z/OSMF workflow engine to perform provisioning, reconfiguration, and de-provisioning tasks for the zCX instances.

When you have the z/OSMF-based provisioning for the planned cluster instances completed, you must install Red Hat Enterprise Core operating system as the container operating system into your cluster as a part of zCX for OpenShift.

Note: For more information about downloading and installing Red Hat OpenShift Container Platform, see this [Red Hat OpenShift web page](#).

Processors and z/OS LPARs

zCX for OpenShift is an eligible workload for zIIP processors on your IBM Z machines. If you want to avoid running your zCX clusters on general purpose processors, ensure that enough logical zIIP processors are defined to your LPARs where the zCX for OpenShift instances are running.

A minimal zCX for OpenShift configuration must feature the following processor resources:

- ▶ Four virtual CPUs each for:
 - The bootstrap instance (temporarily needed for the installation process)
 - The three required control plane instances
 - The (minimum) two compute node instances
- ▶ Four virtual CPUs: for one optional bastion node that is running on a Docker-based zCX instance.
- ▶ For a nonhigh availability solution, one z/OS system with six zIIP processors is needed that are running in SMT-2 mode.

A recommended zCX for OpenShift configuration includes the following components:

- ▶ For a high availability solution, use three z/OS LPARs on multiple IBM Z CECs.
- ▶ Define one control node and two compute nodes per z/OS system.
- ▶ Define six logical zIIP processors per z/OS LPAR

Figure 1-1 shows the recommended configuration for Red Hat OpenShift high availability.

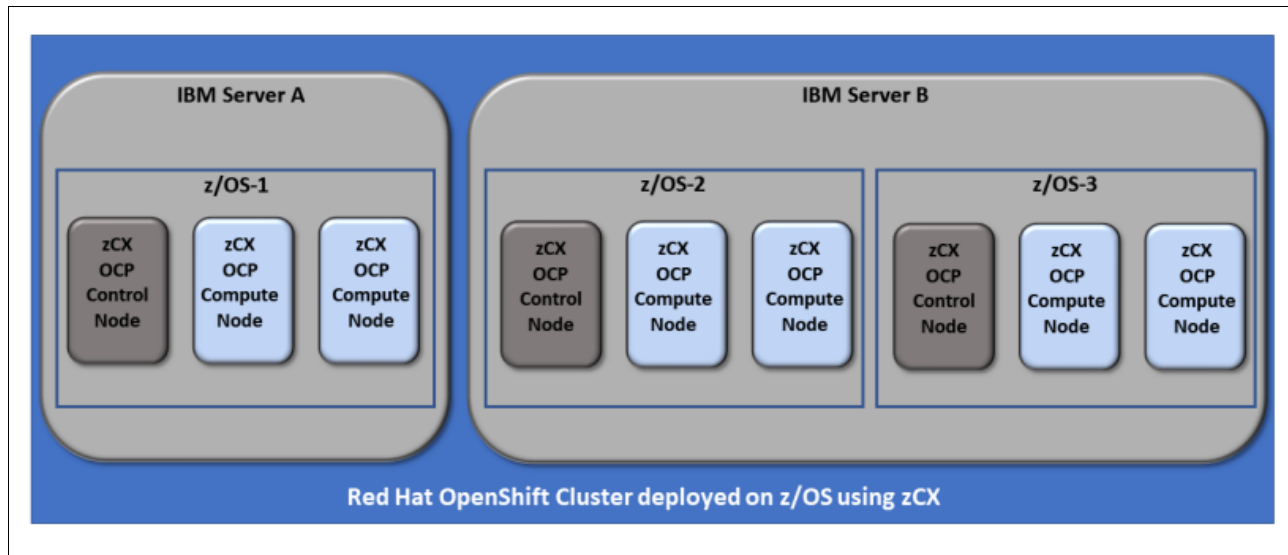


Figure 1-1 Red Hat recommended high availability configuration

Memory

In addition to the processor requirements and recommendations, other processor memory considerations must be fulfilled. Overall, 80 GB of fixed z/OS memory (above the bar) must be allocated for the different zCX for OpenShift instances in a minimum configuration.

At a minimum, the following memory allocations are needed:

- ▶ 16 GB temporarily for the bootstrap instance
- ▶ 16 GB for each of the three control plane instances
- ▶ 8 GB for each of the (minimum) two compute node instances

Optionally, 16 GB fixed memory is needed if you plan to install one bastion node on a Docker-based zCX instance.

Network

You must be able to connect to your zCX for OpenShift cluster environment. Therefore, the following network requirements must be fulfilled:

- ▶ Six z/OS TCP/IP DVIPA addresses are defined for your zCX for OpenShift instances.
- ▶ Network connectivity is defined between your zCX for OpenShift instances and your OpenShift infrastructure services. Depending on your environment, this network connectivity can be the use of:
 - OSA network adapters
 - Existing or newly defined:
 - HiperSocket connections between z/OS systems within one IBM Z CEC
 - SMC-D connections (Shared Memory Communications, Direct Memory Access) between z/OS systems within one IBM Z CEC
- ▶ Optionally, one z/OS TCP/IP DVIPA address is defined for a Docker-based zCX instance that works as a bastion node

DASD space

z/OS based disk space is needed for setting up your zCX for OpenShift environment. The data sets that are needed are allocated and populated during the process of provisioning the instances.

The following minimum requirements must be met:

- ▶ The use of Extended Format VSAM Linear Data Sets (LDS) that are on z/OS extended address volume (EAV). For more information, see *zCX Foundation for Red Hat OpenShift, Installing a Red Hat OpenShift Cluster on z/OS with zCX*, GC31-5709.
- ▶ For each instance within your zCX for OpenShift cluster, 100 GB of DASD space is available; for example, for six instances, 600 GB is needed. If you plan to add compute nodes, 100 GB is needed for each node.
- ▶ If you plan to implement a bastion node, add 30 GB of DASD space.

Persistent storage

If you need more local storage for your zCX for OpenShift instances, you can add persistent storage optionally by defining it during provisioning workflow, or later by using the `ocp_add_local_storage_disks` workflow. For more information, see *zCX Foundation for Red Hat OpenShift, Installing a Red Hat OpenShift Cluster on z/OS with zCX*, GC31-5709.

Another way to add persistent storage is to use a Network File System (NFS). You can use the z/OS NFS server or a non-z/OS server.

Note: For more information about setting up and using a z/OS NFS server, see *z/OS Network File System Guide and Reference*, SC23-6883.

For more information about the use of persistent storage by using NFS in a zCX for OpenShift cluster environment, see this [Red Hat OpenShift web page](#).

Installation

For more information about installing the zCX cluster, see Chapter 4 of *zCX Foundation for Red Hat OpenShift, Installing a Red Hat OpenShift Cluster on z/OS with zCX*, GC31-5709. The chapter also provides information about Red Hat OpenShift web pages that are available for more specific installation information.

Note: For more information about installing OpenShift on IBM Z, see this [Red Hat OpenShift web page](#).

1.2.2 High availability and Disaster Recovery

To achieve the highest possible availability of the OpenShift clusters, Red Hat recommends the system architecture that is shown in Figure 1-1.

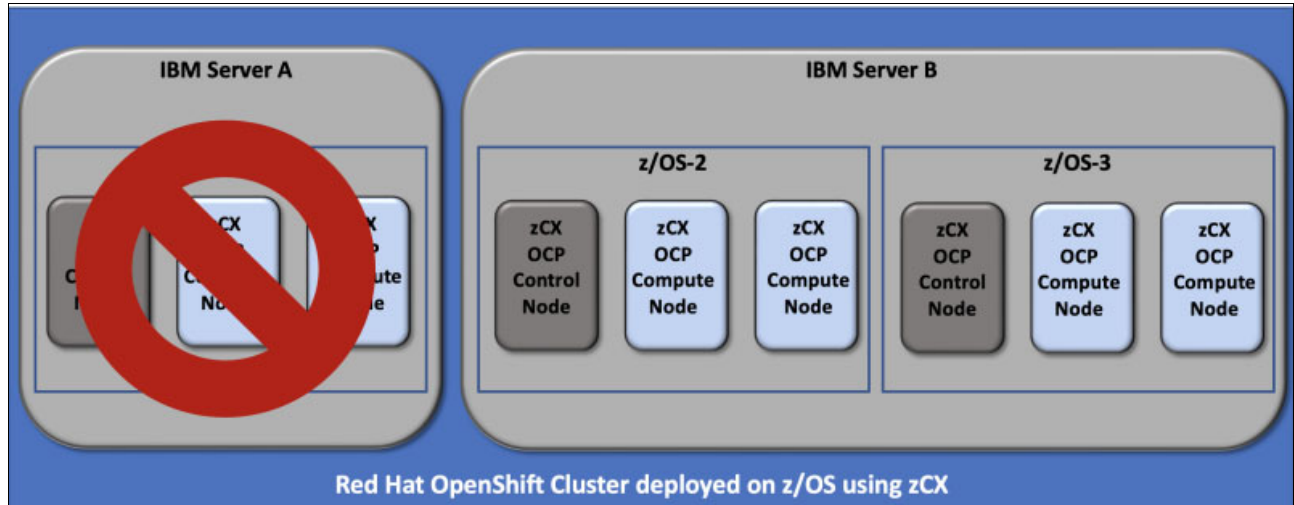


Figure 1-2 System Outage while running Red Hat OpenShift Clusters in zCX

The control nodes must be distributed across different z/OS systems and at best, even across different CECS and LPARs. Following this distribution, the compute nodes also must be distributed across different systems in the sysplex. Each node runs in a separate address space; that is, in its own zCX instance.

If system outage occurs, the DR in z/OS that is shown in Figure 1-2 is used and the application or the nodes can be moved or restarted in the sysplex.

Note: If it is not possible to provide the described system resources, there are also several other options regarding the application itself, to ensure high availability of Red Hat OpenShift. For more information, see your IBM customer service representative.

1.2.3 Administration and operations

Among the many benefits of the use OpenShift on zCX are administration and operation of functions and tools. As with any other installation, the infrastructure management is done by the infrastructure administrators (which in this case, for z/OS on IBM Z, the z/OS administrators) and for OpenShift administration, the OpenShift administrators.

In this way, administrators do not need to acquire new skills to manage and operate the tasks that are started on z/OS because it is handled as any other started task. Also, OpenShift administrators do not need the skills of the underlying infrastructure, which eliminates the need to acquire z/OS skills.

Note: This section covers only some examples of what a z/OS or OpenShift administrator can use. For more information, see this [IBM Documentation web page](#).

Administering and operating from a z/OS perspective

As discussed, the z/OS uses the familiar tools for operating, and treats each OpenShift for zCX instance as a started task. In this section, we present some examples of what a z/OS administrator might encounter when operating with OpenShift for zCX.

Figure 1-3 shows how SDSF displays the started tasks on the system and allows you to review consumptions, start or stop tasks, check logs, and so on.

```
Display Filter View Print Options Search Help
-----
SDSF DA SC74 SC74 PAG 0 CPU/L 51/ 51 LINE 1-5 (5)
COMMAND INPUT ==> SCROLL ==> PAGE
NP JOBNAME StepName ProcStep JobID Owner C Pos DP Real Paging SIO
OCP1CTL0 OCP1CTL0 GLZBAIN STC07140 EDMCAR NS FE 18T 0.00 0.00
OCP1CTL1 OCP1CTL1 GLZBAIN STC07141 EDMCAR NS FE 18T 0.00 0.00
OCP1CTL2 OCP1CTL2 GLZBAIN STC07142 EDMCAR NS FE 17T 0.00 0.00
OCP1CMP0 OCP1CMP0 GLZBAIN STC07143 EDMCAR NS FE 18T 0.00 0.00
OCP1CMP1 OCP1CMP1 GLZBAIN STC07144 EDMCAR NS FE 18T 0.00 0.00

F1=HELP F2=SPLIT F3=END F4=RETURN F5=RFIND F6=RCHANGE
F7=UP F8=DOWN F9=SWAP F10=LEFT F11=RIGHT F12=RETRIEVE
M& a 04/021
```

Figure 1-3 Monitoring OpenShift for zCX with SDSF

When operating, the resources and other parameters that are associated with the zCX instances can be changed through z/OSMF workflows, as shown on Figure 1-4.

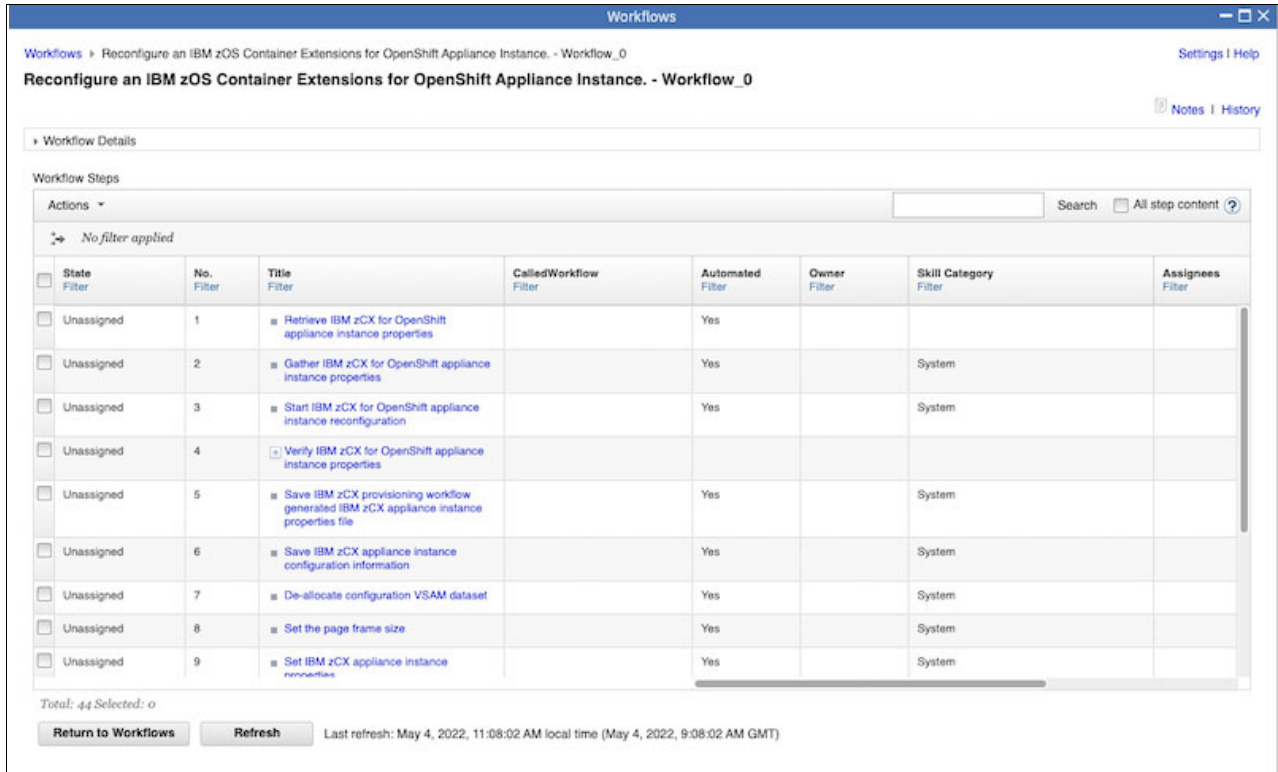


Figure 1-4 Operating OpenShift for zCX with z/OSMF

Administering OpenShift from an OpenShift administrator perspective

An OpenShift administrator still can access the OpenShift console for management and operating purposes.

Figure 1-5 shows how an OpenShift administrator accesses the console to adjust cluster settings and make any changes as needed.

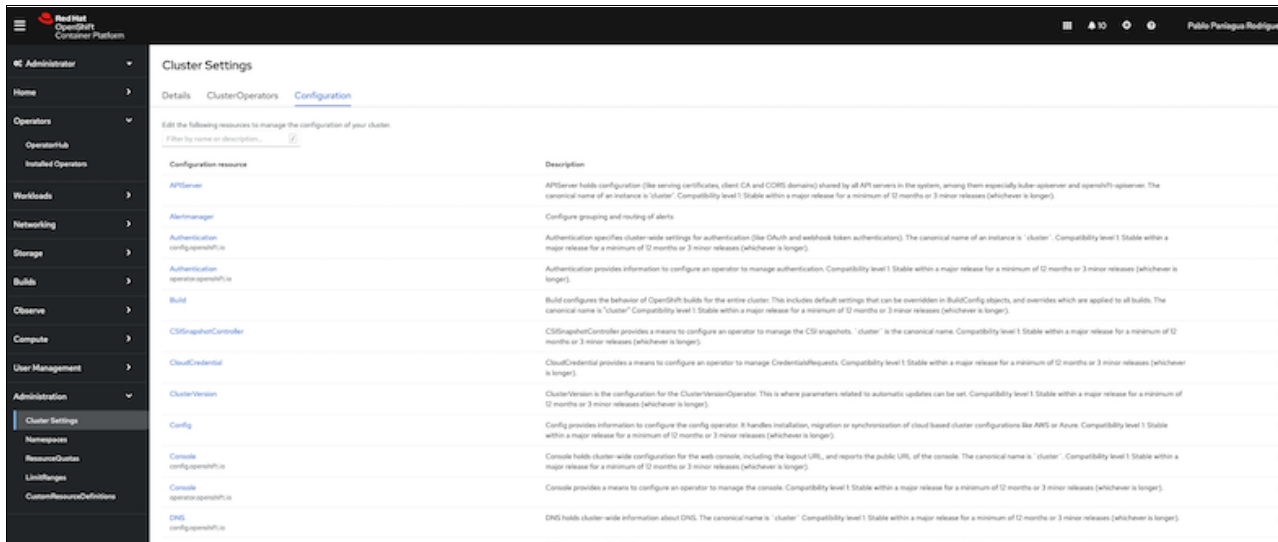


Figure 1-5 Administering OpenShift for zCX with OpenShift Container Platform Console

OpenShift also can be administered by using the command-line interface (CLI), as shown in Figure 1-6. The command that is shown in Figure 1-6 displays the same information that is shown in Figure 1-5 on page 9.

Note: For more information about downloading the OpenShift CLI, see this [Red Hat OpenShift web page](#).

```
mbp-de-pablo:rdbkzcxocp pablopaniagua$ oc get nodes
NAME                                STATUS    ROLES    AGE      VERSION
compute-0.ocpzc1.rdbkocp.pbm.ihost.com Ready    worker   21d     v1.23.5+9ce5071
compute-1.ocpzc1.rdbkocp.pbm.ihost.com Ready    worker   21d     v1.23.5+9ce5071
compute-2.ocpzc1.rdbkocp.pbm.ihost.com Ready    worker   4d20h   v1.23.5+9ce5071
compute-3.ocpzc1.rdbkocp.pbm.ihost.com Ready    worker   4d20h   v1.23.5+9ce5071
control-0.ocpzc1.rdbkocp.pbm.ihost.com Ready    master   21d     v1.23.5+9ce5071
control-1.ocpzc1.rdbkocp.pbm.ihost.com Ready    master   21d     v1.23.5+9ce5071
control-2.ocpzc1.rdbkocp.pbm.ihost.com Ready    master   21d     v1.23.5+9ce5071
```

Figure 1-6 Administering OpenShift for zCX with CLI

1.2.4 Migration for modernization

The zCX Foundation for Red Hat OpenShift opens the door for modernization in the mainframe. It allows users to install and operate new applications that previously were not available in the mainframe.

By giving the users the freedom to deploy Linux applications on z/OS LPARs and operate them with OpenShift, users find that some previous applications that were too complex to deploy on zCX standalone now are viable inside z/OS.

Modernization includes the following examples:

- ▶ **DevSecOps:** Improve time to market and provide a cloud-native experience on z/OS by integrating Linux-based development tools within z/OS environment.
- ▶ **z/OS Software Ecosystem Expansion:** Deploy and operate open source tools, such as \non-SQL databases, and microservices. The new zCX Foundation for Red Hat OpenShift allows the deployment of new workloads as containers and orchestrates them by using a platform, such as OpenShift.
- ▶ **Colocation:** Enrich the workloads on the mainframe with business software that was previously outside (for example, on OpenShift on x86) that can now benefit from colocation in the same LPAR. This colocation improves the interoperability between distributed and IBM Z environments.

1.3 Lab environment overview

This section describes our lab environment that was used for the Red Hat OpenShift cluster on zCX.

1.3.1 Hardware

The Red Hat OpenShift cluster runs in a z/OS LPAR, which belongs to a 2-way parallel sysplex cluster on an IBM z16™ system (3931-A01, software model 724).

The LPAR is configured with four logical CP engines and eight logical zIIP engines. The zIIP engines are configured to run in SMT2 mode (symmetric multithreading). If not already done, you can configure SMT2 mode for zIIP engines by setting the PROCVIEW parameter in the LOADxx member of SYSx.IPLPARM to CORE. By using this setting, you can change SMT between 1 and 2 on a running z/OS system.

Figure 1-7 shows the result of the use of the **D M=CPU** operator command, which displays the processor status of the z/OS system.

```
D M=CPU
IEE174I 06.30.15 DISPLAY M 075
CORE STATUS: HD=Y      MT=2      MT_MODE: CP=1  zIIP=2
ID      ST      ID RANGE      VP      ISCM      CPU THREAD STATUS
0000    +      0000-0001    M      FC00      +N
0001    +      0002-0003    L      0000      +N
0002    +      0004-0005    LP     0000      +N
0003    +      0006-0007    LP     0000      +N
0004    +I      0008-0009    M      0200      ++
0005    +I      000A-000B    L      0200      ++
0006    +I      000C-000D    L      0200      ++
0007    +I      000E-000F    L      0200      ++
0008    +I      0010-0011    L      0200      ++
0009    +I      0012-0013    LP     0200      ++
000A    +I      0014-0015    LP     0200      ++
000B    +I      0016-0017    LP     0200      ++
000C    -      0018-0019
000D    -      001A-001B
000E    -      001C-001D
000F    -      001E-001F
0010    -I      0020-0021
0011    -I      0022-0023
0012    -I      0024-0025
0013    -I      0026-0027
```

Figure 1-7 Result of operator command **D M=CPU**, showing the LPAR processor status

You should also verify that the amount of memory for your z/OS system is sufficient. Use the z/OS operator command **D M=STOR** to verify the amount of memory that belongs to the z/OS system (see Figure 1-8 on page 12).

```

D M_STOR
IEE305I D          COMMAND INVALID
D M=STOR
IEE174I 06.41.21 DISPLAY M 080
REAL STORAGE STATUS
ONLINE-NOT RECONFIGURABLE
      0G-512G
ONLINE-RECONFIGURABLE
      NONE
PENDING OFFLINE
      NONE
      262144M IN OFFLINE STORAGE ELEMENT(S)
      0M UNASSIGNED STORAGE
STORAGE INCREMENT SIZE IS 1G

```

Figure 1-8 Result of operator command D M=STOR, showing the LPAR memory

It is recommended that you run the cluster on different z/OS systems for high availability reasons,

1.3.2 Operating system

The operating system that runs our Red Hat OpenShift cluster is z/OS V2.5. Although this environment runs z/OS 2.5, you can also use z/OS 2.4 as the minimum z/OS release.

z/OSMF, which is the web-based graphical user interface (GUI) for z/OS to administer various aspects of z/OS, also is necessary for parts of the setup of the Red Hat OpenShift cluster.

Mainly, the z/OSMF workflow engine is used for the base part of the setup. When you start setting up your instances for the cluster nodes, you enter the workflow engine in z/OSMF by clicking the icon that is highlighted Figure 1-9.

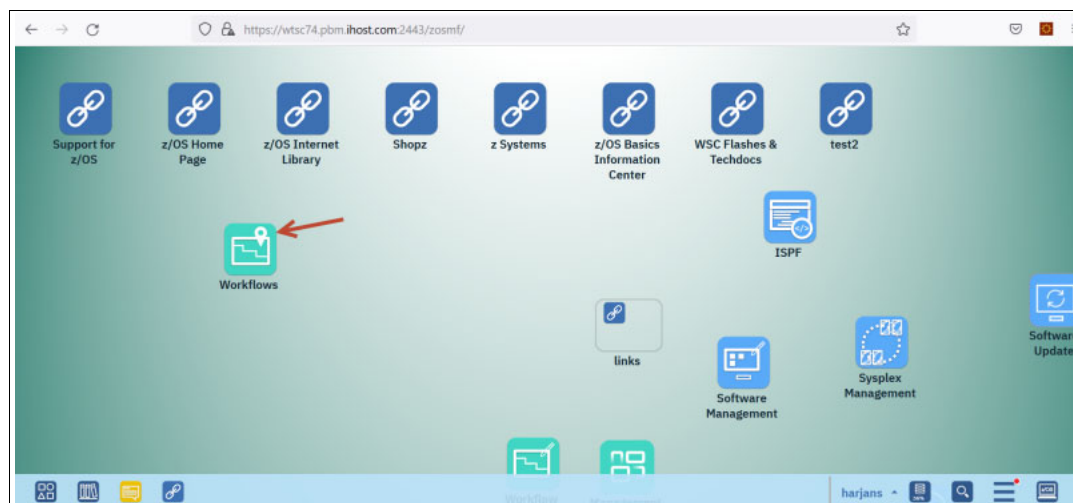


Figure 1-9 z/OSMF icon for workflow engine

For the initial setup, a workflow is created by using `/usr/lpp/zcx_ocp/workflows/ocp_provision.xml` as a base. Then, this file is copied to one of your own directories, and runs it in the z/OSMF workflow engine. A template file also is available for the workflow properties in `/usr/lpp/zcx_ocp/properties`.

1.3.3 Lab infrastructure

The zCX for OpenShift cluster with its minimal configuration with three control and two compute nodes is running on one z/OS system. It is a supported configuration, but from a high availability perspective, it is recommended to split the zCX for OpenShift instances onto multiple z/OS LPARs. The ideal configuration is to have these LPARs on different IBM Z machines.

1.3.4 IBM Spectrum Scale

For persistent storage, an IBM Spectrum® Scale cluster was created and the CNSA and CSI operators were deployed on Red Hat OpenShift.

Note: As of this writing, IBM Spectrum Scale for zCX Foundation for Red Hat OpenShift was not officially supported. However, because the architecture is the same as Red Hat OpenShift for Linux on IBM Z, the team decided to use it as a storage solution

The cluster was created by using the IBM Spectrum Scale installation toolkit on top of three virtual machines (VMs) with Red Hat Enterprise Linux 8.5. The configuration of the IBM Spectrum Scale Storage cluster is shown in Figure 1-10.

```
GPFS cluster information
=====
GPFS cluster name:      gpfsrdbks.pbm.ihost.com
GPFS cluster id:       1862967071590597617
GPFS UID domain:      gpfsrdbks.pbm.ihost.com
Remote shell command: /usr/bin/ssh
Remote file copy command: /usr/bin/scp
Repository type:      CCR

Node  Daemon node name      IP address  Admin node name
Designation
-----
--
  1  rdbksss4.pbm.ihost.com  129.40.23.46  rdbksss4.pbm.ihost.com
quorum-manager-perfmon
  2  rdbksss5.pbm.ihost.com  129.40.23.47  rdbksss5.pbm.ihost.com
quorum-manager-perfmon
  3  rdbksss6.pbm.ihost.com  129.40.23.48  rdbksss6.pbm.ihost.com
quorum-manager-perfmon
```

Figure 1-10 IBM Spectrum Scale Lab environment configuration

On Red Hat OpenShift, another cluster was deployed to act as a Container Native Storage Access (CNSA), as shown in Figure 1-11.

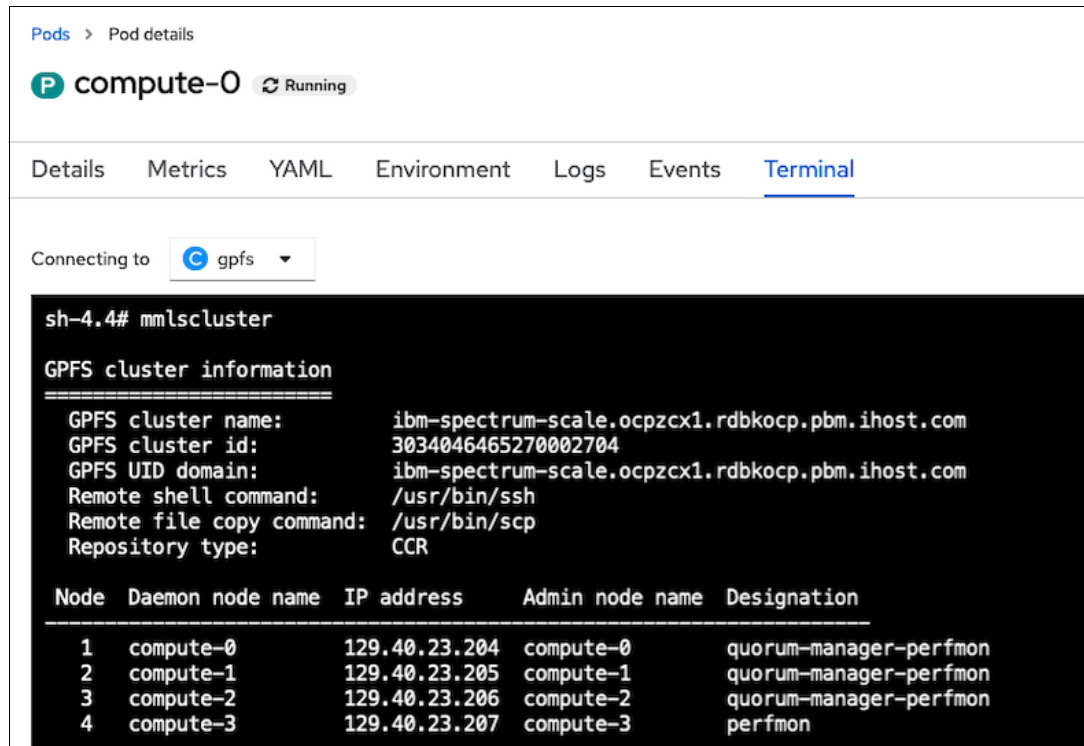


Figure 1-11 Spectrum Scale CNSA

A storage class was created to automatically provision all of the persistent volume claims that are needed in the cluster.


1.3.5 Licensing

When you plan to implement zCX Foundation for Red Hat OpenShift, you must prepare the licensing in advance.

The following alternatives are available when you want to run zCX for OpenShift:

- ▶ Use a 60 day trial period.
For this option, you must modify your IFAPRDxx member of SYS1.PARMLIB, as described in Appendix A of *zCX Foundation for Red Hat OpenShift, Installing a Red Hat OpenShift Cluster on z/OS with zCX*, GC31-5709.
- ▶ Place an order for zCX Foundation for Red Hat OpenShift in IBM ShopZ and select program identifier 5655-ZCX.
For more information, see this [IBM announcement letter](#).

On the z/OS side, you can find the z/OSMF workflows and a workflow properties template file on your system in UNIX System Services in the `/usr/lpp/zcx_ocp` directory.



Use case 1: Using z/OS Connect inside Red Hat OpenShift

Unlock the value of your IBM Z subsystems with truly RESTful APIs and create consumable APIs in minutes to make IBM Z applications and data central to your hybrid cloud strategy. Call APIs from Z applications to enhance them with the power of cloud native functions.

Running z/OS Connect inside Red Hat OpenShift can help you to attain that goal.

In this chapter, we discuss the deployment of z/OS Connect inside Red Hat OpenShift and the deployment of applications that call these APIs. The zCX Foundation for Red Hat OpenShift provides a unified environment to manage the deployment of multiple applications and helps to improve the performance by collocating the applications and the APIs on the same cluster and machine.

This chapter includes the following topics

- ▶ “Introduction” on page 16
- ▶ “Solution” on page 18
- ▶ “Summary” on page 30

2.1 Introduction

This section provides an overview of z/OS Connect and describes different deployment possibilities.

2.1.1 Overview

z/OS Connect provides a fast, secure, and reliable way to connect to any z/OS asset. z/OS Connect provides a standard way to identify these assets and reach the assets by using REST technology.

z/OS Connect was enhanced to include cloud native development support and API-first mapping for OpenAPI 3 interfaces to z/OS applications and data. These new functions are provided through the following new components that are prepared to be deployed as a container or as an application on Red Hat OpenShift:

- ▶ **z/OS Connect Designer**
Allows the user to create APIs in minutes with a low code approach. It also supports OpenAPI 3 that aligns with the industry standards and allows OpenAPI 3 interfaces to be created for all z/OS assets that are supported by z/OS Connect.
- ▶ **z/OS Connect Server**
Allows APIs to be built as a container image and be deployed in various containers, which decentralizes the ownership. It also allows users to work in parallel. Each API is deployed as a micro-service that can be managed by Red Hat OpenShift.

Also, a noncontainerized z/OS Connect Server is available natively on z/OS for OpenAPI 3.

Figure on page 16 provides an overview of the process to create and deploy a z/OS Connect OpenAPI 3 to an IBM Z container environment or a native (noncontainerized) deployment to IBM z/OS on IBM Z.

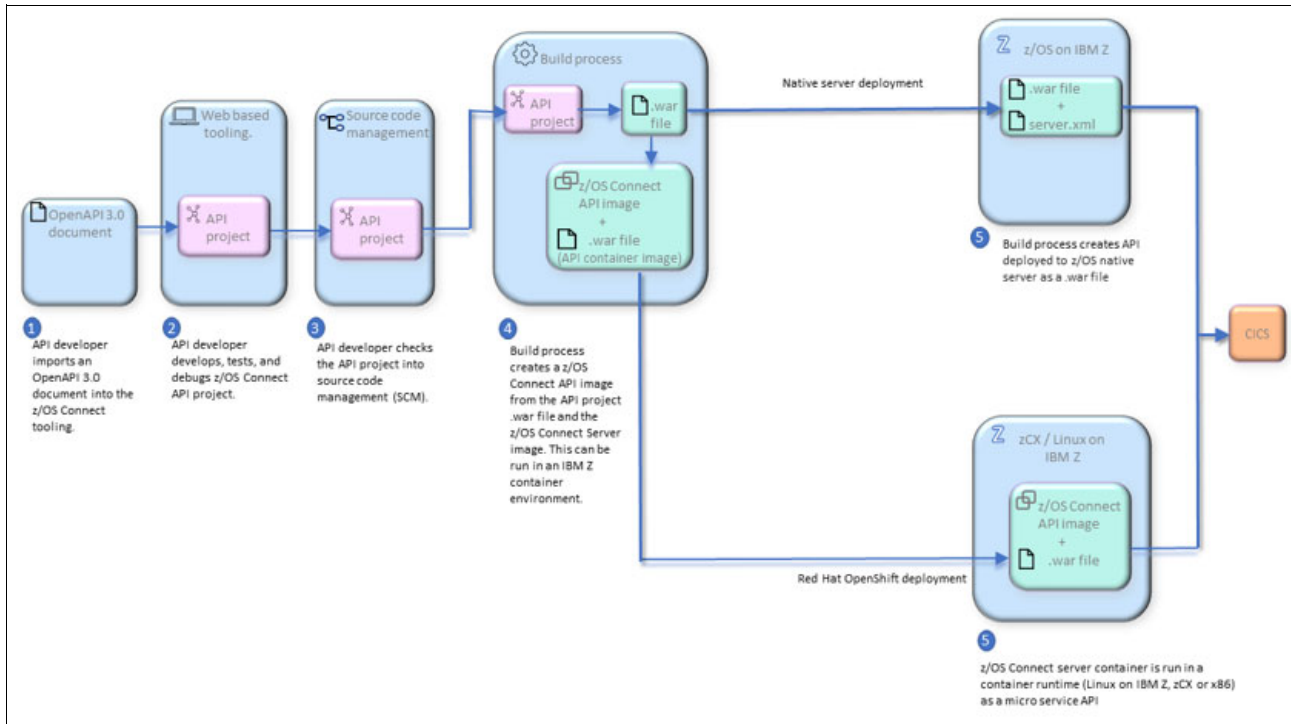


Figure 2-1 z/OS Connect OpenAPI 3 process flow

This IBM Redbooks publication discusses the deployment when Red Hat OpenShift is used.

Note: For more information about z/OS Connect, see this [IBM Documentation web page](#).

2.1.2 Problem statement

z/OS Connect with OpenAPI 2 was deployed on z/OS in the same system that held the applications it was going to API-enable (or Apify). With this architecture, and before zCX, applications that accessed the APIs were deployed outside of IBM Z to connect z/OS applications and data by using z/OS Connect (OpenAPI 2). This configuration led to higher latencies, and security and resiliency issues.

Figure 2-2 shows how z/OS Connect (OpenAPI 2) was deployed on top of z/OS and how a user accessed APIs from an external source.

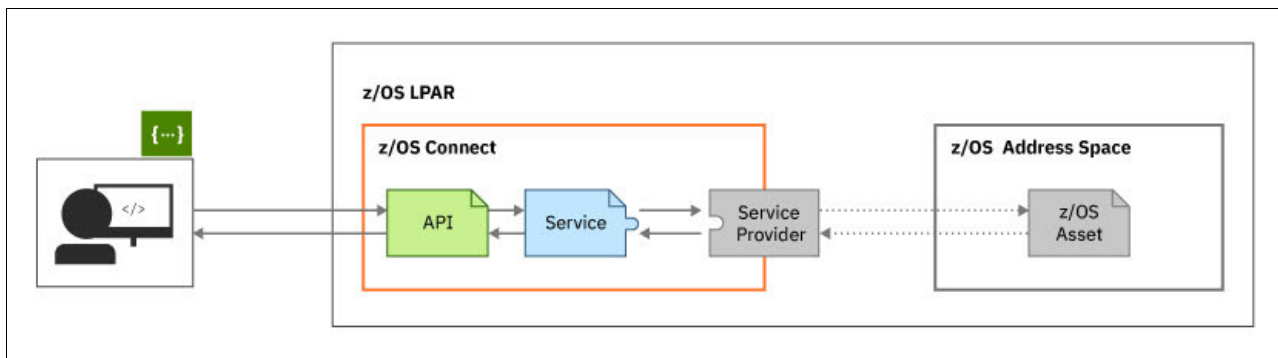


Figure 2-2 z/OS Connect (OpenAPI 2) Architecture

2.2 Solution

The new approach of z/OS Connect with OpenAPI 3 provides many new capabilities and functions, such as providing the option of the use of containers. This use of containers provides flexibility and allows users with different skill sets to easily deploy and manage z/OS Connect APIs.

z/OS Connect can now be deployed on multiple architectures and platforms. It runs on any hardware that supports the chosen operating system.

IBM z/OS Connect that is used to support OpenAPI 3 includes the following new features:

- ▶ The Open Container initiative (OCI) compliant z/OS Connect Designer image, which is a container image that is freely available on the IBM container registry (icr.io). This image allows you to run the Designer container on a Red Hat OpenShift cluster on-premises.
- ▶ The Open Container initiative (OCI) compliant z/OS Connect Server image, which is a new container-based server for running IBM z/OS Connect OpenAPI 3.0 specification APIs on your Open Container Platforms (OCP).
- ▶ The OpenAPI 3 feature in the IBM z/OS Connect native server, which is an IBM z/OS Connect Server that is running natively on z/OS.

For more information about system requirements to run IBM z/OS Connect as a native server or as an image in a container, see this [IBM Documentation web page](#).

The new z/OS Connect Server can be deployed on multiple environments, which can be categorized into the following two main groups:

- ▶ Managed outside of z/OS

Environments that fall into this group use a container runtime on Linux with Red Hat OpenShift or Kubernetes. Users that consider this option must consider the effect on their organization when deploying applications that access z/OS data from outside of z/OS.

Consider the following points:

- Environments that fall into this group include Red Hat OpenShift for Linux on IBM Z, AWS, IBM ROKS, and so on.
- Running in this sort of environment allows the API to be run and managed alongside applications that might need to call the API.
- Application teams that can access these environments can easily deploy new APIs.
- Requires IFLs to run: Users that are not using Linux on IBM Z that want to use Linux on IBM Z or LinuxONE need IFLs (IBM Integrated Facility for Linux), which are processors that are dedicated to Linux workloads on IBM Z and IBM LinuxONE.
- Requires Linux administration skills: Linux skills or people with Linux skills typically are assigned to distributed environments rather than IBM Z.
- HA strategies might be different: Running z/OS and Linux environments can increase the complexity of managing multiple environments when implementing high availability. Also, applications that are running outside of z/OS that need to access data that is inside of z/OS might experience latency issues that can affect high availability.

Whether on IBM Z (z390x) or distributed (amd64), managing outside of the z/OS environment covers various non-z/OS, on-premises, and cloud environments (for example, Linux on IBM Z, IBM Cloud®, AWS, Azure, and GCP).

► Managed inside of z/OS

Although running on x86 or natively in z/OS (as shown in Figure on page 16) are also options, the solutions that are described in this publication involve management inside of z/OS and include zCX and the zCX Foundation for Red Hat OpenShift.

For this solution, the following benefits are realized:

- Low latency between z/OS Connect and the z/OS Application and Data.
- Reuse existing zIIP processors: If a company was installing z/OS Connect on z/OS, they can make use of their zIIP processors to deploy the new z/OS Connect,
- Reuse z/OS skills to manage and operate the underlying infrastructure: Although it is true that Red Hat OpenShift and Docker skills are needed to deploy z/OS Connect, all the underlying infrastructure can be managed with z/OS, which eliminates the need for Linux administration skills.
- Take advantage of colocation: z/OS Connect APIs can be deployed on the same LPAR as the z/OS applications it is connecting to, which improves the communication between both systems and can reduce latency and improve high availability.

When deploying on zCX, z/OS Connect can be easily managed for some simple APIs. However, as the number of APIs increase, so do the number of z/OS Connect Server instances and an orchestration solution is needed.

The zCX Foundation for Red Hat OpenShift offers a great opportunity to deploy not only z/OS Connect Server instances but also the applications that revolve around them. This deployment improves communication between z/OS Connect APIs and z/OS application also between z/OS Connect APIs and the consumers of these applications.

2.2.1 Solution architectural overview

The architecture that we used in this publication use case is shown in Figure 2-3.

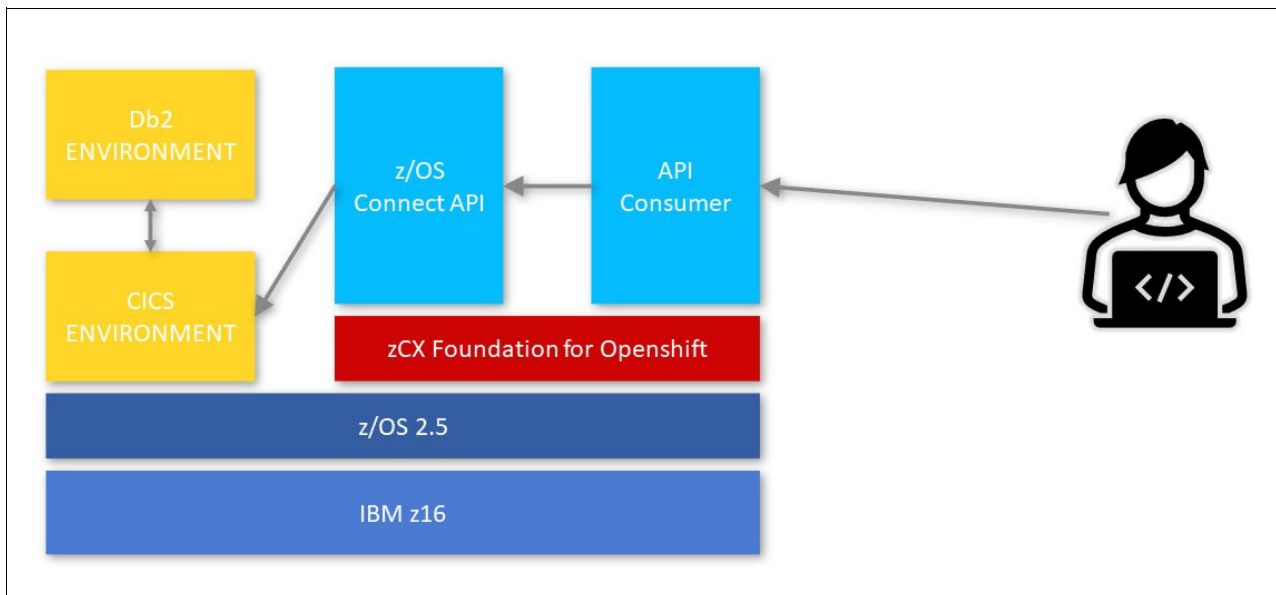


Figure 2-3 General architecture of the solution

The test environment that was used consists of an IBM CICS® environment that is running a simple application that accesses IBM Db2 data and is being API-enabled with z/OS Connect on top of the zCX Foundation for Red Hat OpenShift. Also, a web API Consumer was created to demonstrate how one or many applications also can be deployed in zCX for Red Hat OpenShift to use the low latency and security that workload colocation provides.

2.2.2 Installation overview

In this section, we provide an overview of how we connected the API consumer to the CICS environment.

The CICS environment is running a simple service that provides data about different customers from a Db2 database. The z/OS Connect API was created by using the z/OS Connect Designer to expose this CICS transaction through a simple API that allows other components to use it.

We accomplished this process by following instructions that are available at this [IBM Documentation web page](#).

Note: For more information about the two methods that were used to install z/OS Connect and postinstallation steps, see this [IBM Documentation web page](#).

To fully install the solution, the following main components must be deployed on top of Red Hat OpenShift:

- ▶ z/OS Connect Operator: The operator that is in charge of managing the multiple z/OS Connect server images that are deployed to Red Hat OpenShift.
- ▶ z/OS Connect API Image: An image that embeds the WAR file that is generated in z/OS Connect Designer that contains all the information about your specific API.

The following sections provide an overview of the installation process that was followed in our lab environment.

Deploying z/OS Connect Operator

To deploy the z/OS Connect Operator, the IBM Operator catalog must be included in the Operator Hub. To do so, the user must click the + sign of the Red Hat OpenShift Console and add the code that is shown in Figure 2-4 on page 21.


```

apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: ibm-operator-catalog
  namespace: openshift-marketplace
spec:
  displayName: IBM Operator Catalog
  publisher: IBM
  sourceType: grpc
  image: icr.io/cpopen/ibm-operator-catalog
  updateStrategy:
    registryPoll:
      interval: 45m

```

Figure 2-4 Adding IBM Catalog to the Red Hat OpenShift Marketplace

Figure 2-5 shows how this process is done at the console level.

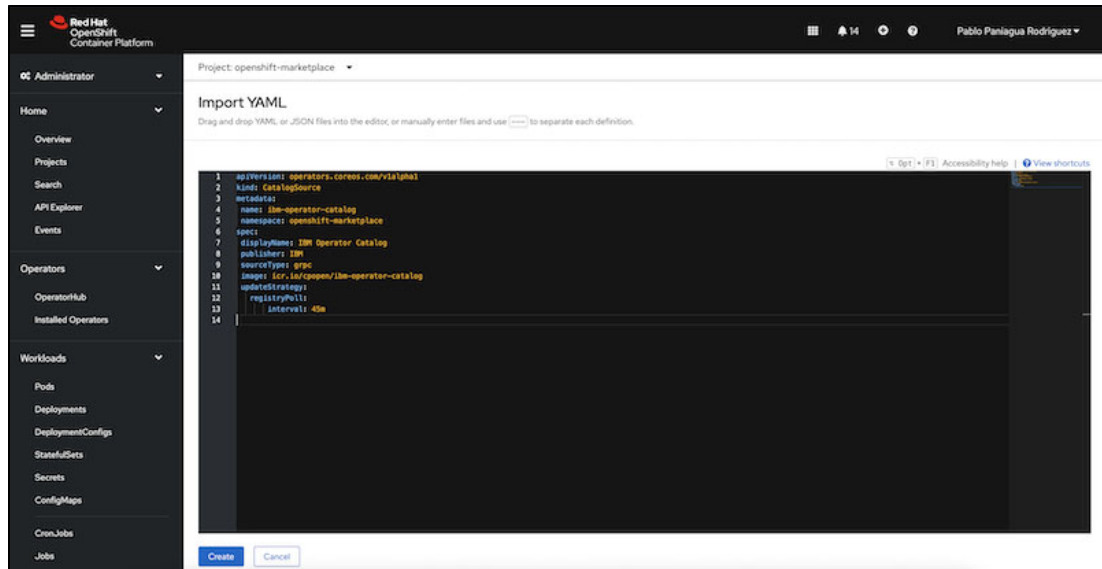


Figure 2-5 Adding IBM Operator Catalog to Red Hat OpenShift

After the catalog is created, you can verify that the catalog is correctly included by using the command in the CLI (see Figure 2-6).

```

[root@rdbksss4 ~]# oc get CatalogSource -n openshift-marketplace

```

NAME	DISPLAY	TYPE	PUBLISHER	AGE
certified-operators	Certified Operators	grpc	Red Hat	135d
community-operators	Community Operators	grpc	Red Hat	135d
ibm-operator-catalog	IBM Operator Catalog	grpc	IBM	112d
redhat-marketplace	Red Hat Marketplace	grpc	Red Hat	135d
redhat-operators	Red Hat Operators	grpc	Red Hat	135d

Figure 2-6 Verify catalog source

Note: Administration authorization is required for these steps. The lab environment is built with full access, but you can check the user requirements at this [IBM Documentation web page](#)

You also might need to include your IBM credentials on the cluster, as described at this [IBM Documentation web page](#).

After the Operator Catalog is included, you should see the IBM z/OS Connect operator in the Operator Hub section of the console.

Tip: You can filter by source to see all IBM operators that are available to you.

To install the operator, click the IBM z/OS Connect Operator and then, click **Install**. The following installation modes are available (depending on the type of deployment that is needed):

- ▶ All namespaces on the cluster, which allows your z/OS Connect resources to be created and managed from any namespace.
- ▶ A specific namespace on the cluster, which allows only you to manage z/OS Connect resources from a specific project.

If you plan to deploy multiple z/OS Connect Servers, it likely is better to install the operator on all namespaces.

Figure 2-7 shows how z/OS Connect is installed through the Operator Hub.

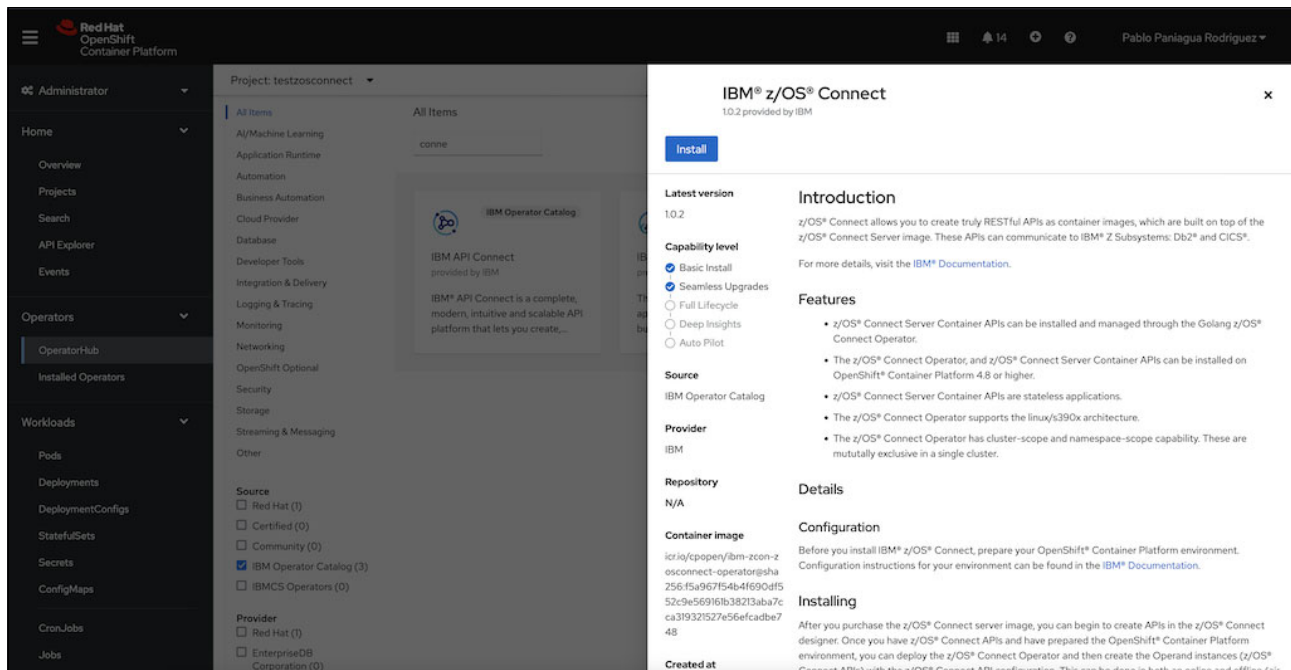


Figure 2-7 z/OS Connect installed

After the operator installation completes, you should see it under your installed Operators on the Red Hat OpenShift console (see Figure 2-8).

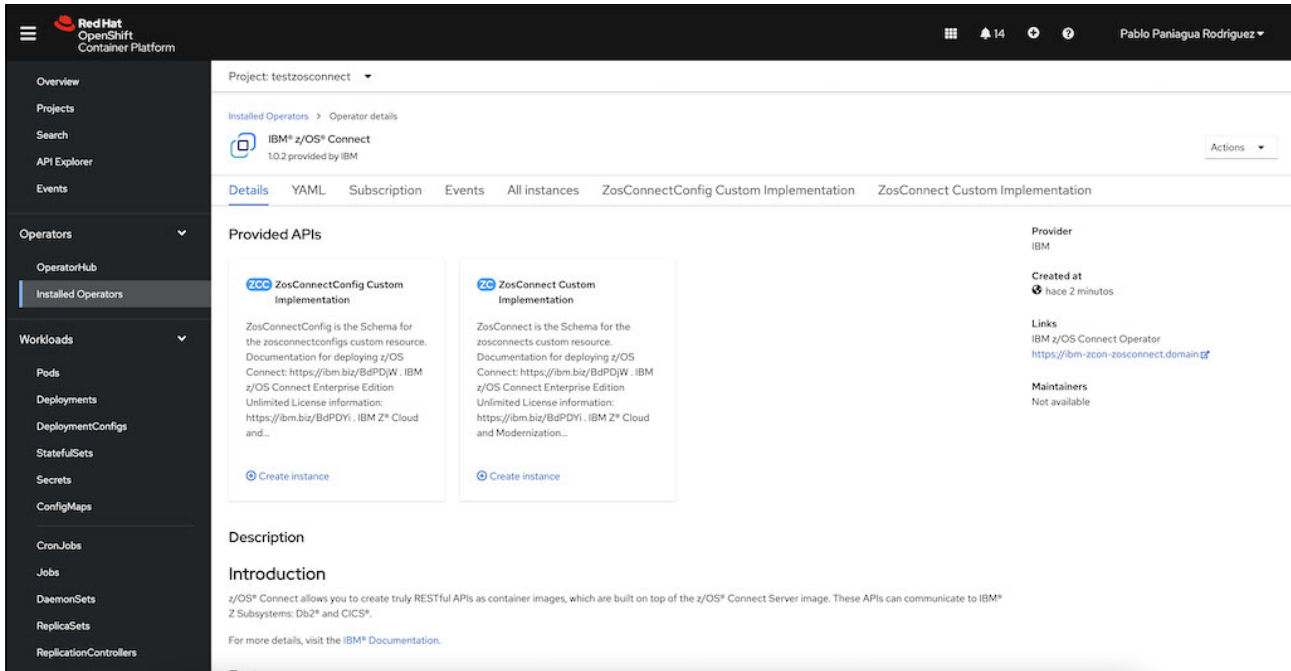


Figure 2-8 z/OS Connect operator deployed

Deploying z/OS Connect Server Resources

To fully deploy a z/OS application on the Red Hat OpenShift cluster, the following prerequisites must be met:

- ▶ Download the z/OS Connect Server image

This image serves as a base image to build all of your z/OS Connect API images. To download it, log in to the IBM registry by using your entitlement registry key. You can use Podman or Docker, depending on the environment that is used.

```
[root@rdbksss4 ~]# podman login icr.io
[root@rdbksss4 ~]# podman pull
icr.io/zosconnectunlimited/ibm-zcon-server:3.0.59
```

Figure 2-9 Podman login and pull

Note: This image must be pulled to the server in which you are going build your z/OS Connect API images.

- ▶ Build the IBM z/OS Connect API Image

To create the z/OS Connect API Image, you must created a z/OS Connect API project on the z/OS Designer and built it by using the [Gradle automation build tool](#).

For more information, see this [IBM Documentation web page](#).

You should have a Dockerfile on your z/OS Connect API project that is referencing the z/OS Connect Server image that you want to use as a base for your API image. This Dockerfile can be verified by going into the Dockerfile and checking the FROM and comparing it to the pull that was completed. You can verify the images on your Podman or Docker environment by using the command that is shown in Figure 2-10.

```
[root@rdbksss4 ~]# podman image ls
REPOSITORY
TAG          IMAGE ID      CREATED      SIZE
icr.io/zosconnectunlimited/ibm-zcon-server
3.0.59      c5571b216996 3 months ago 71.7 MB
```

Figure 2-10 check z/OS Connect version

To build and push the image to our Red Hat OpenShift environment, you must have the registry exposed on your Red Hat OpenShift Cluster. You can check the registry URL by running the command that is shown in Figure 2-11.

```
[root@rdbksss4 ~]# oc get routes -n openshift-image-registry
NAME          HOST/PORT
PATH          SERVICES     PORT    TERMINATION  WILDCARD
default-route
default-route-openshift-image-registry.apps.ocpzc1.rdbkocp.pbm.ihost.com
image-registry <all> reencrypt    None
```

Figure 2-11 Check openshift registry default route

After all of the prerequisites are met, you proceed with building the API image. Ensure that you are in the same location as the Dockerfile before running the command that is shown in Figure 2-12.

```
[root@rdbksss4 ~]# podman build -t
<your-registry-url>/<your-project>/<your-api-name>:<your-version> .

[root@rdbksss4 ~]# podman build -t
default-route-openshift-image-registry.apps.ocpzc1.rdbkocp.pbm.ihost.com/ibm-z
con-zosconnect-system/genapp-api:v1.0 .
```

Figure 2-12 Example of building API image

Note: If you are deploying on a s390x architecture, build the image from an s390x server or use [buildx](#).

After the image is built, you can push it to the Red Hat OpenShift registry by logging in (see Figure 2-13 on page 25) and running the command that is shown in Figure 2-14 on page 25.

```
[root@rdbksss4 ~]# podman login
default-route-openshift-image-registry.apps.ocpzc1.rdbkocp.pbm.ihost.com
--tls-verify=false
Username: pablopani
Password:
Login Succeeded!
```

Figure 2-13 Podman logging to Red Hat OpenShift Registry

```
[root@rdbksss4 ~]# podman push
default-route-openshift-image-registry.apps.ocpzc1.rdbkocp.pbm.ihost.com/ibm-z
con-zosconnect-system/genapp-api:v1.0
```

Figure 2-14 Podman push to Red Hat OpenShift Registry

Note: Before pushing the image to the Red Hat OpenShift registry, you must create the project that you want to use. To do so, click **Projects** → **Create Project**.

Deploying a z/OS Connect resource instance

With the prerequisites met, the API image is available in your Red Hat OpenShift Registry and the z/OS Connect Operator deployed so that they can be used to deploy z/OS Connect resources.

You can generate a z/OS Connect resource by going to the Developer view, clicking **+Add**, and choosing the operator back-end **ZosConnect Custom Implementation**, as shown in Figure 2-15.

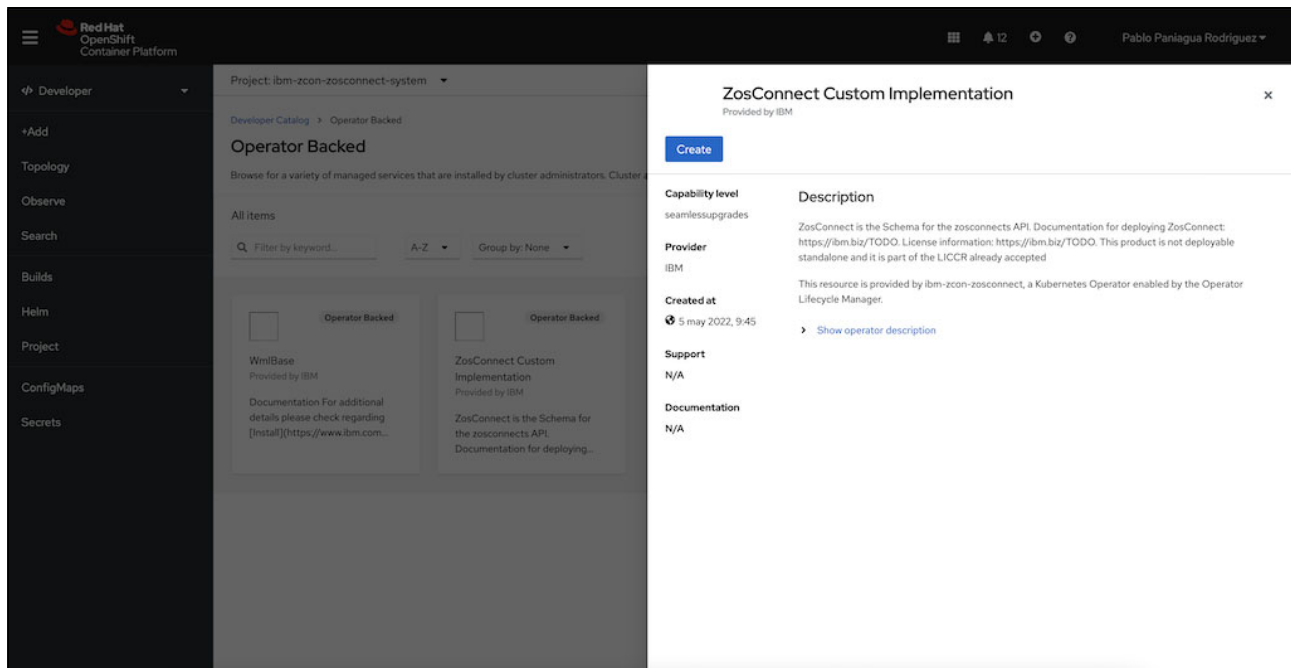


Figure 2-15 z/OS Connect custom implementation

You can select to configure from Form View or YAML view. Often, you have more control of what is being deployed by using the YAML view.

Figure 2-16 shows our lab environment that is deploying in YAML view mode.

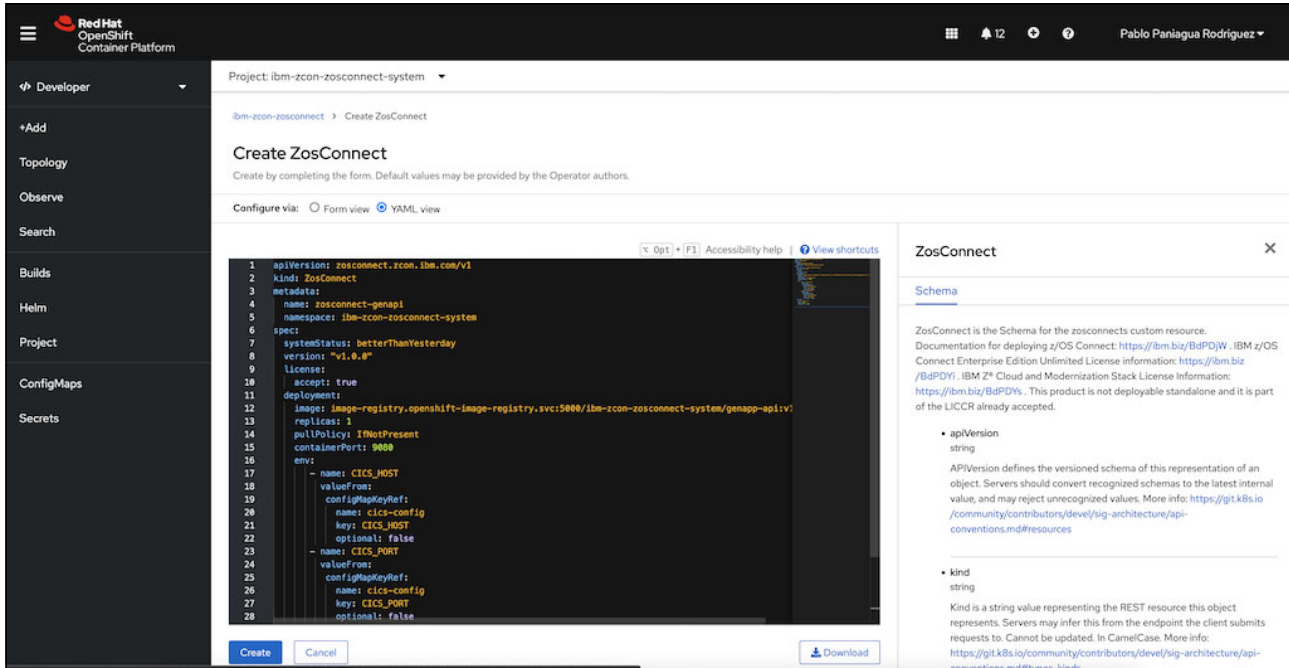


Figure 2-16 YAML View z/OS Connect resource

The lab configuration that we used is shown in Figure 2-16. The sections that are highlighted in **bold** must be adapted to your specific configuration.

```
apiVersion: zosconnect.zcon.ibm.com/v1
kind: ZosConnect
metadata:
  name: zosconnect-genapi
  namespace: ibm-zcon-zosconnect-system
spec:
  systemStatus: betterThanYesterday
  version: "v1.0.0"
  license:
    accept: true
  deployment:
    image: image-registry.openshift-image-registry.svc:5000/ibm-zcon-zosconnect-system/genapp-api:v1.0
    replicas: 1
    pullPolicy: IfNotPresent
    containerPort: 9080
    env:
      - name: CICS_HOST
        valueFrom:
          configMapKeyRef:
            name: cics-config
            key: CICS_HOST
            optional: false
      - name: CICS_PORT
        valueFrom:
          configMapKeyRef:
            name: cics-config
            key: CICS_PORT
            optional: false
  service:
    type: ClusterIP
    port: 9080
    targetPort: 9080
```

Figure 2-17 Example of z/OS Connect YAML Resource

The env section covers the specific environment variables that are used in your application. They should be generated in a ConfigMap file to be correctly used by the application.

After you complete the yaml configuration, click **Create**. After a few minutes, you should see your container deployed in your topology view (see Figure 2-18 on page 28).

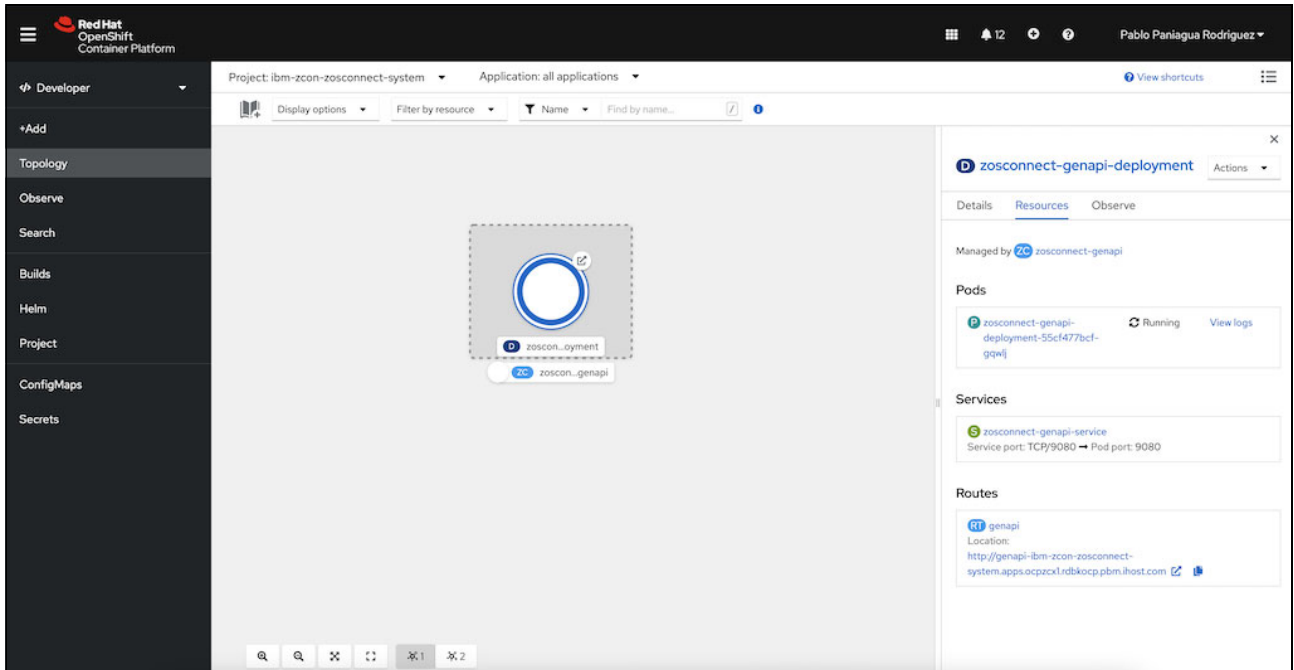


Figure 2-18 Topology view on Red Hat OpenShift Console

If you want to start your APIs from outside your Red Hat OpenShift Cluster, generate a route by going to the Administrator view and clicking **Networking** → **Routes** → **Create Route**. There, you can select a name for your route, a port, the service you want to access, and security rules (see Figure 2-19).

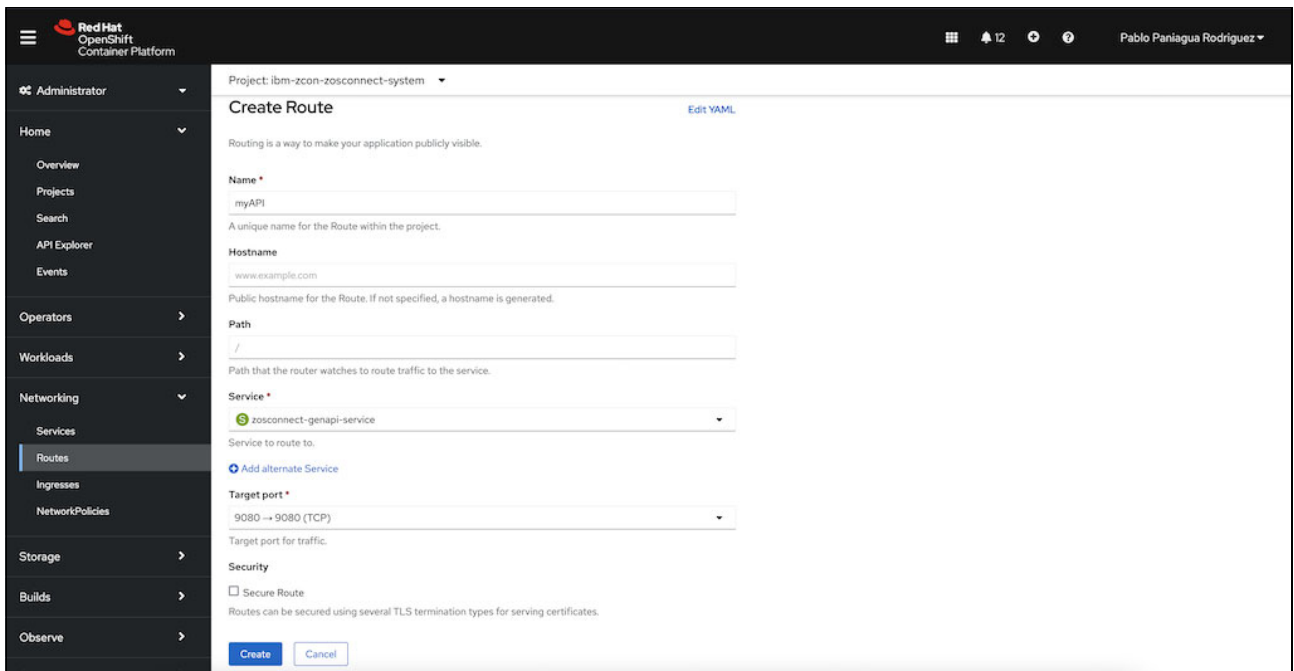


Figure 2-19 Create route for the z/OS Connect API

This environment can be seen by accessing the transaction through a 3270 terminal. Figure 2-20 shows how the data is exposed in JSON format. For z/OS Connect, an operator is deployed on Red Hat OpenShift that oversees managing the different APIs that are generated.

```
{
  "first_name": "Brian",
  "last_name": "Cant",
  "date_of_birth": "1933-07-12",
  "house_name": "",
  "house_number": "58",
  "postcode": "IP221PL",
  "number_of_policies": 0,
  "phone_mobile": "",
  "phone_home": "01855 122134",
  "email_address": "CANT@beebhouse.com",
  "customer_id": 6
}
```

Figure 2-20 z/OS Connect API JSON data

The Web API Consumer is an example of a web application that was created specifically for this publication that serves as an example of how web or third-party applications can use these APIs that were generated with z/OS Connect. Figure 2-21 shows an example of what the user sees after the full architecture is deployed.

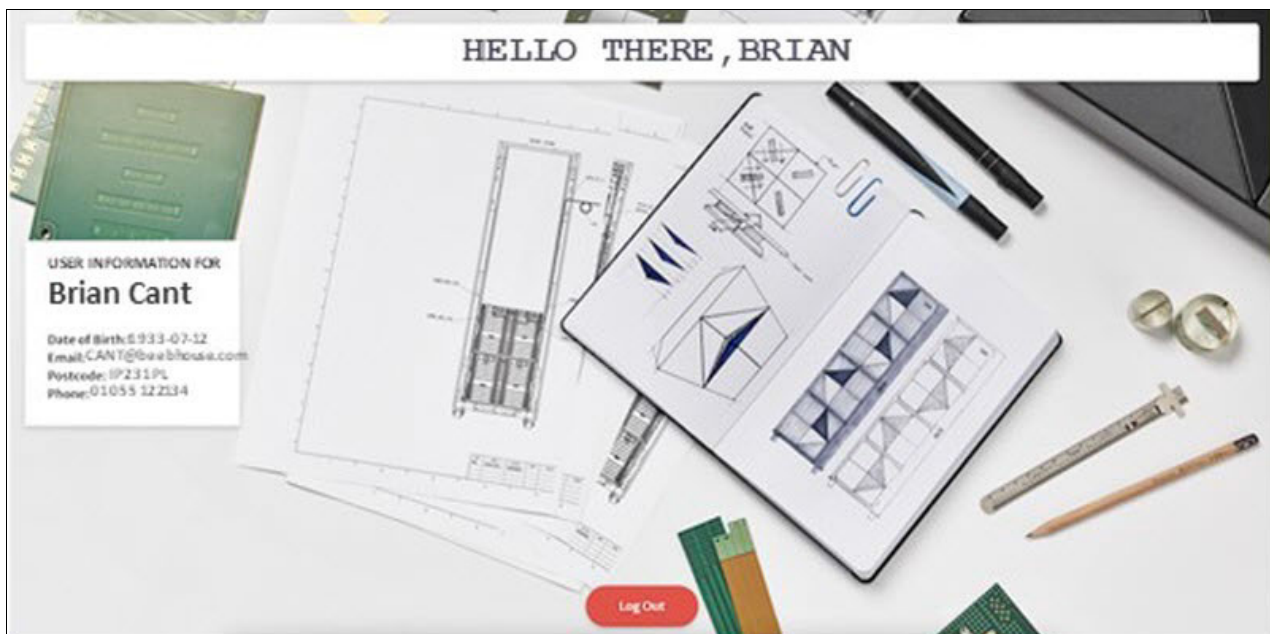


Figure 2-21 Web API Consumer example

The following components were deployed on Red Hat OpenShift for our IBM Redbooks lab environment:

- ▶ **z/OS Connect Operator:** Manages the API instances that are generated and the management and lifecycle of the z/OS Connect APIs in Kubernetes clusters. After it is configured, the operator continually monitors the current state of the cluster, and compares it to the wanted state.
- ▶ **z/OS Connect Instance:** The specific API that is generated for our CICS transaction.
- ▶ **Web API Deployment:** The deployment of the example application. This application is written in NodeJS and is being built through a GIT strategy.
- ▶ We created a z/OS Connect API Route (although we did not need it because the application was in the same project). This route is used to access the API application externally.
- ▶ We created a Web API Route to access the Web application externally.
- ▶ z/OS Connect secrets that include the URL and credentials to connect to the CICS environment.

2.3 Summary

Our use case was simple. We required a cloud-based web application to provide data about customers. Our data was in a Db2 database on z/OS.

Our goal was to access the z/OS data with a CICS transaction and communicate with our web server by using an HTTP protocol.

We accomplished this goal by using the z/OS Connect Designer to create a simple z/OS Connect API project for our CICS transaction. Through the build process, a z/OS Connect API image is created from the API project and WAR file that now can be run in the Red Hat Red Hat OpenShift on zCX container environment.

Because web or third-party applications can use these APIs that were generated with z/OS Connect, a business can run this z/OS Connect API container where it makes the most sense for them, whether it is on the Red Hat OpenShift Container Platform on z/OS, in zCX, or Linux on IBM Z.



Use case 2: Artificial intelligence and open source tools

Red Hat OpenShift Container Platform is self-managed. It accelerates the delivery of artificial intelligence (AI) powered intelligent applications in your z/OS environment and hybrid cloud. Some example use cases include fraud detection, data driven diagnostics and cure, autonomous driving, oil and gas exploration, automated insurance quotes, and claims processing.

This chapter discusses the deployment of AI models for fraud detection inside Red Hat OpenShift. In addition, if you are running on IBM z16 hardware, then integration with the AI Accelerator can seamlessly meet high compute resource requirements to help select the best machine language (ML) model providing the highest prediction accuracy, and ML inferencing as and when the model encounters new data in your online (CICS or IMS/TM) production environment.

This chapter includes the following topics:

- ▶ “Background” on page 32
- ▶ “Problem being addressed” on page 34
- ▶ “Solution description” on page 35
- ▶ “Benefits” on page 36

3.1 Background

In general, the following workloads are suitable to move to zCX Foundation on Red Hat OpenShift Container Platform for z/OS:

- ▶ UX and user interface programs
- ▶ Workloads that are meant to be “elastic” in nature
- ▶ BPM and automation type batch programs
- ▶ Workloads with higher performance requirements in a single thread
- ▶ Workloads that require the advantage of specialty processors
- ▶ Complex workloads with deep data dependencies and high volume
- ▶ Enables private cloud-in-a-box at low data center footprint and extreme consolidation
- ▶ High degree of security concern
- ▶ Parts of high transaction workloads with no COTS alternatives
- ▶ Distributed Java workload for better agility, scalability, and lower costs
- ▶ Colocation with z/OS back end to achieve lower latency
- ▶ AI workloads that can use AI accelerators on z16

z16's AI accelerator, which helps with tensor calculations, is important for data gravity because it provides low-latency, real-time AI processing. Figure 3-1 shows when to use TensorFlow versus ONNX and IBM Deep Learning Compiler

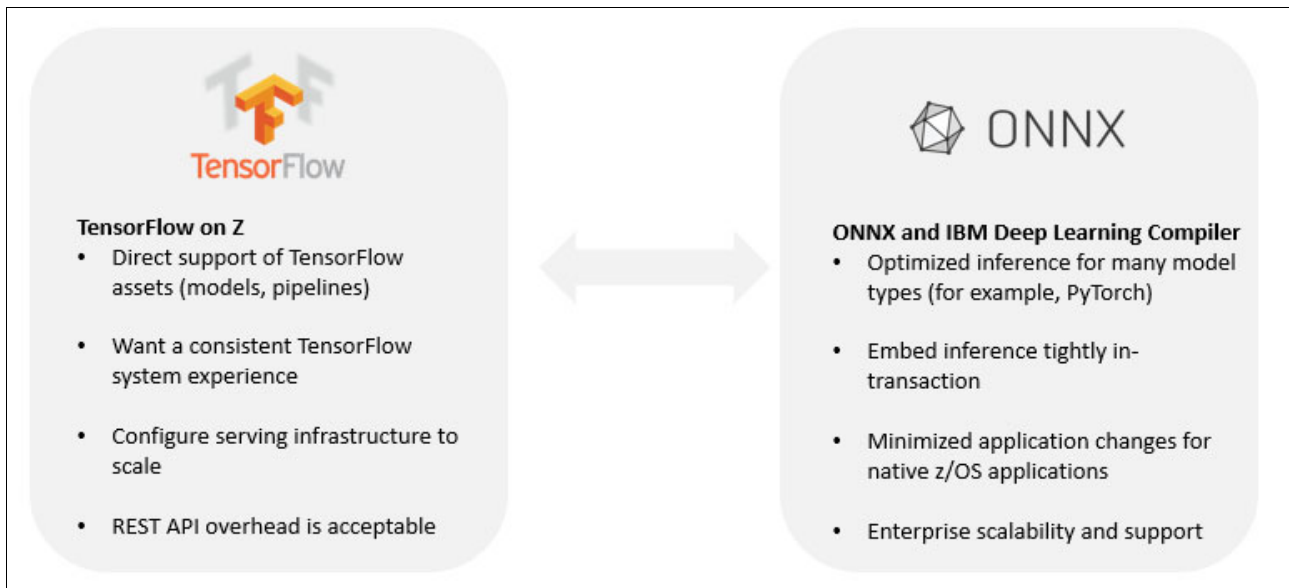


Figure 3-1 Deep learning models with AI accelerator use

zCX Foundation on Red Hat OpenShift Container Platform for z/OS helps with all phases of the AI lifecycle while building an AI powered solution on z16.

AI lifecycle is a multi-phase process to obtain the power of large volumes and various data, abundant compute, and open source machine learning tools to build intelligent applications.

At a high level, the AI lifecycle includes the following phases:

1. Gather and prepare data to ensure that the input data is complete, and of high quality.
2. Develop model, including training, testing, and selection of the model with the highest prediction accuracy.
3. Integrate models into the application development process, and inferencing.
4. Model monitoring and management to measure business performance and address potential production data drift.

Example 3-2 highlights other areas for deployment of open source tools on zCX Foundation for Red Hat OpenShift Container Platform where it can help with various mainframe modernization plays.

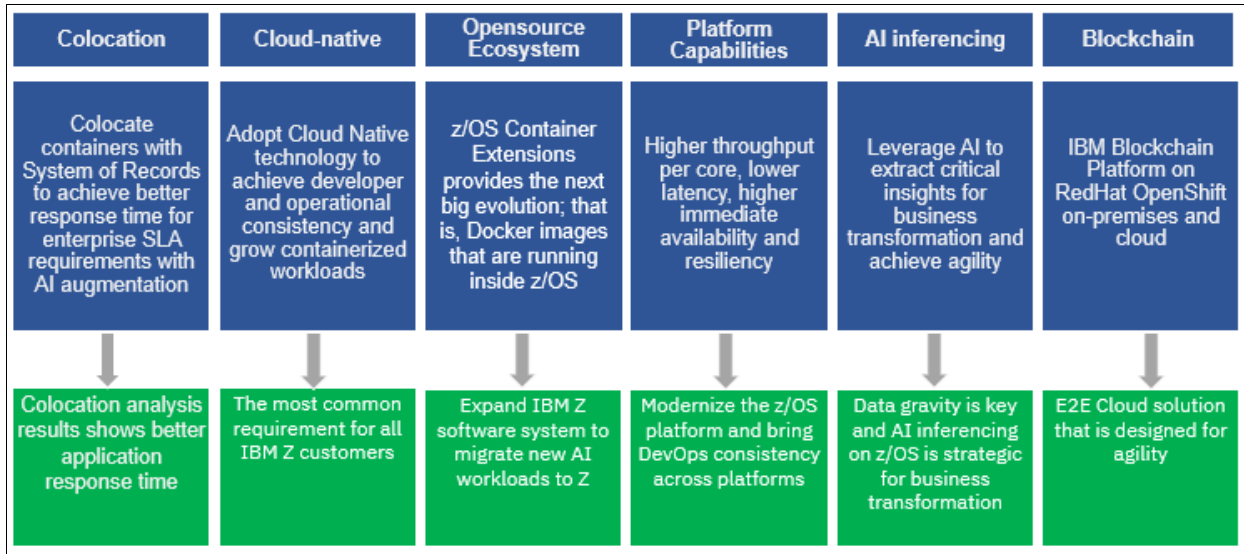


Figure 3-2 Areas for deployment of open source tools

Credit card fraud detection AI workflow

In this scenario, a card processor that is needed to determine which transactions have a high risk of fraud exposure before settlement. The current AI workflow involves TensorFlow inference serving containers that are running in Linux on Z, as shown in Figure 3-3.

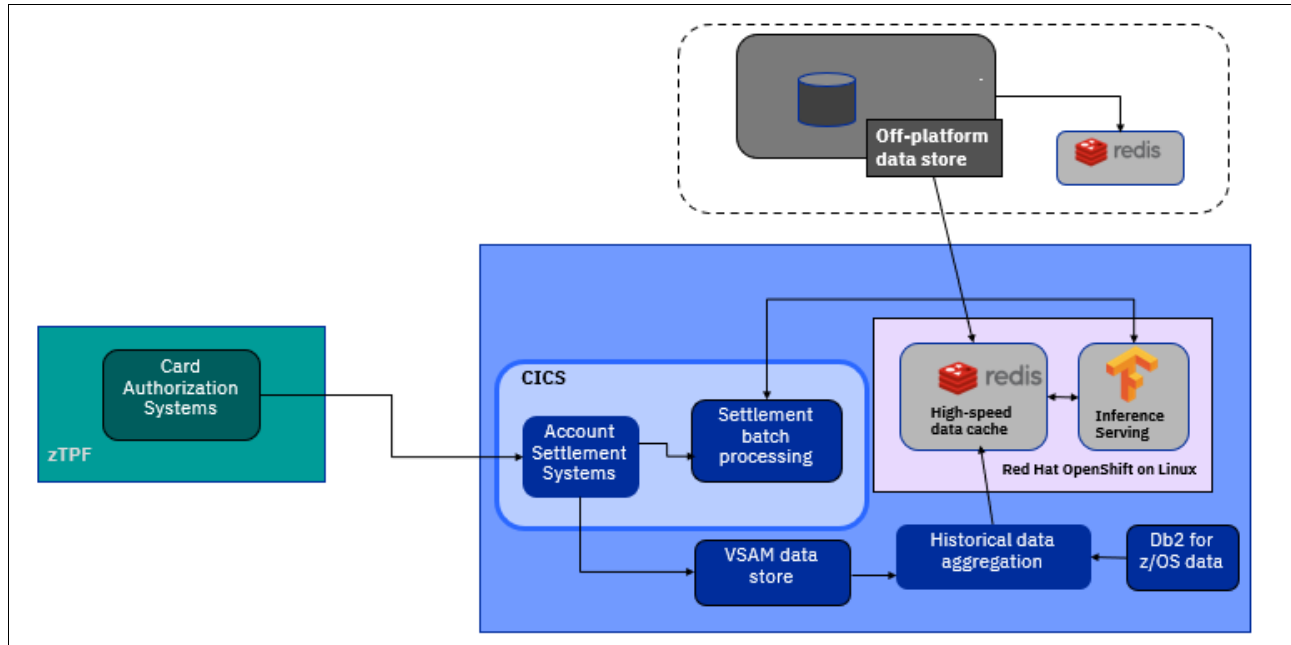


Figure 3-3 AI workflow for credit card fraud detection

3.2 Problem being addressed

In this use case, we address how to deploy Docker Containers on a z/OS system, in direct support of z/OS workloads without requiring a separately provisioned Linux server while maintaining overall operational control within z/OS.

Several other challenges also are addressed in this use case from a data scientists perspective, as discussed next.

3.2.1 Challenges facing data scientists

Data scientists face the following key challenges on the z/OS platform:

- ▶ Lack of DevSecOps for AI Lifecycle and dependency on IT operations to provision and manage infrastructure.
- ▶ Lack of open source system of AI and ML tools.
- ▶ Complexities to train, test, select, and retrain the ML model that provides the highest prediction accuracy.
- ▶ Lack of hardware acceleration.
- ▶ Collaborating with data engineers and software developers to ensure input data quality and model deployment in application development processes.

3.3 Solution description

Here, we focus on creating a simple solution to deploy Linux on IBM Z software components as Docker Containers onto a Red Hat OpenShift Container Platform on z/OS. This deployment is in direct support of z/OS workloads without requiring a separately provisioned Linux server while maintaining overall solution operational control within z/OS and with z/OS Qualities of Service.

Figure 3-4 shows the solution components that were used in the AI Application that are on the z/OS environment.

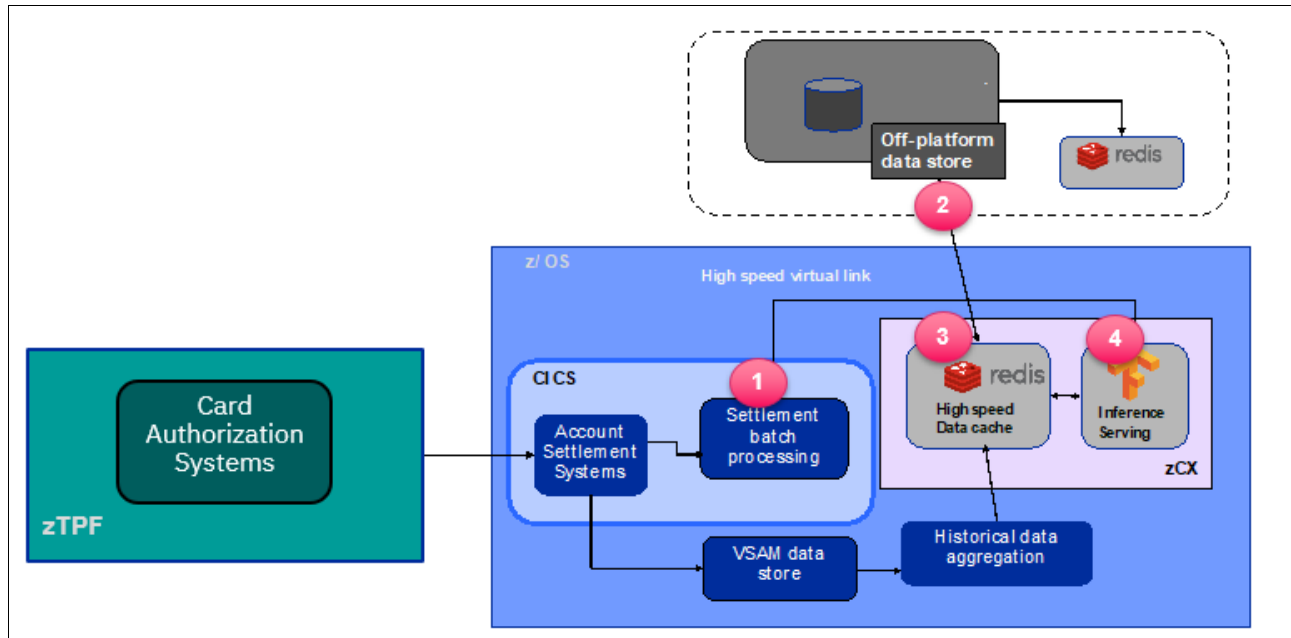


Figure 3-4 Solution components on z/OS

In this scenario, an AI model is hosted and deployed to zCX on Red Hat OpenShift Container Platform. This model requires “new” data that is coming from the business application receiving insight (see #1 Figure 3-4) and historical data that is on an off-platform data lake (see #2 Figure 3-4).

With SLAs that demand ultra low latency, the required off-platform data is replicated to Redis, which is a high-speed in-memory data store that is deployed to zCX on Red Hat OpenShift Container Platform (see #3 Figure 3-4).

When the settlement processing (1) needs insight, it calls a simple go-lang server that is in ZCX on Red Hat OpenShift Container Platform (see #4 Figure 3-4).

This server retrieves the needed extra data from Redis (see #3 Figure 3-4) and then, calls TensorFlow for model inference. The result is then returned to the Settlement application.

3.3.1 Deploying Linux on Z containers to zCX on OpenShift

Complete the following steps to deploy the containers to zCX on OpenShift:

1. Create a project:
 - a. From the Administrator perspective of Red Hat OpenShift, click **Projects** and then, click **Create Project**.
 - b. Enter a name for the project (in our example, `zcx-ocp-ai-project`) and then, click **Create**.

After the project is created, you are redirected to the Overview tab of the Project Details page.

2. Run the following commands to copy an image from a registry to your machine:

```
podman pull liutiebj/ccf_os_loz_tfs:v1
podman pull liutiebj/ccf_os_loz_api_svr:v1
```

3. Deploy the pod (for more information, see Appendix A, “Red Hat OpenShift deployment under zCX” on page 53).

3.4 Benefits

In this section, we discuss the benefits that are specific to zCX Foundation on Red Hat OpenShift Container Platform for z/OS.

A solution architect can create a solution to be deployed on z/OS that is based on components that are available, such as Docker containers in the Linux on IBM Z system. It transparently uses z/OS quality of service (QoS) without requiring z/OS development skills.

z/OS Container Extensions Foundation on Red Hat OpenShift Container Platform provides the next significant evolution by running unmodified Linux on Z Docker images inside z/OS.

In addition to latency reduction, the following benefits can be achieved:

- ▶ Makes available cloud-native and AI workloads at high performance, scalability, and security.
- ▶ Combines the power of Red Hat OpenShift portability across clouds with mainframe capabilities, high scalability, performance, and security.
- ▶ Helps implement private cloud-in-a-box with significant reduction in data center footprint (4:1 space versus 2:1).
- ▶ Helps implement true hybrid cloud solutions that can be moved across a mainframe and then distributed to off- and on-premises cloud platforms.
- ▶ Enables granular de-coupling of older mainframe programs because of low latency, which enables incremental modernization and de-risks mainframe modernization program.

Bringing Red Hat OpenShift benefits to z/OS

Red Hat OpenShift enables unified governance and pushes the computation to wherever the data is as outlined in our use case.

A Red Hat OpenShift application developer and data scientist finds that developing applications in Red Hat OpenShift on zCX is the same as that of developing applications in Red Hat OpenShift on other platforms.

z/OS software can be used by deploying your modern applications in the zCX Foundation for Red Hat OpenShift, which provides integration with mission-critical workloads (including CICS, IMS and Db2). For example, applications and workloads that benefit from accessibility and proximity to key resources on z/OS.

Other benefits include the following examples:

- ▶ Secured Self-service provisioning
- ▶ DevSecOps-CI/CD Solution
- ▶ No vendor lock-in (cloud platform independent)
- ▶ Automation enables faster business innovation



Use case 3: Migrating a solution from zCX Docker to Red Hat OpenShift in zCX

In this chapter, we discuss the differences between Docker and Red Hat OpenShift. We also discuss why you might want to migrate a solution from zCX Docker to Red Hat OpenShift in zCX.

4.1 Differences between Docker and Red Hat OpenShift

In this section, the differences between Docker and Red Hat OpenShift are explained to understand the driving reasons to move from zCX Docker to zCX for OpenShift and when a company might consider doing so.

Docker

Docker is a container solution with which the can user build, deploy, and manage containers. By using Docker, the user has a virtualized environment that can be deployed quickly and with high portability to reduce the time to market.

Some of the main advantages of Docker include the following examples:

- ▶ **Modularity:** Different components of an application run in different containers, so you can easily update components.
- ▶ **Layers and versions:** Images consist of layers. These layers are stacked on top of each other to form the container file system, which changes only the layers that are needed if an update occurs.
- ▶ **Application Isolation:** Containers run in isolated environments. Each container is independent from the rest.
- ▶ **Rapid Deployment:** Docker-based containers can reduce deployments to seconds, which is significantly faster than virtual machines.
- ▶ **Portability:** Different containers can be quickly deployed on different environments and multiple architectures.
- ▶ **Cost Savings:** Dramatically reduces infrastructure resources. Fewer resources are necessary to run the same application.

Although Docker is the most common solution, and it is the name that is most referred to when discussing containers, it is not the only container solution. Many other companies also provide container software that is unrelated to Docker, such as Podman, LXD, or CRI-O.

Red Hat OpenShift

Red Hat OpenShift is a container orchestration platform that is based on Kubernetes. It eases the building, deployment, management, and scaling of applications. It is commonly seen as an alternative to Docker or containers, but it is a platform to handle containers at scale. It also is used to be built on top of Docker (now it runs on CRI-O).

Some of the advantages that Red Hat OpenShift provides include the following examples:

- ▶ **Intelligent Scheduler:** Automatically schedule containers to the best possible host to optimize the use of resources.
- ▶ **Scaling:** Scale containers automatically if more throughput is needed.
- ▶ **DevOps:** Enables productivity by simplifying CI/CD processes.
- ▶ **Container monitorization:** Includes a full stack for monitorization of the resources and applications of the platform.
- ▶ **Multi-tenant:** Access to the different applications and resources is controlled by the platform to integrate multiple tenants into the platform.
- ▶ **Security:** Built-in security with default execution policies that prevent applications to be run as root.

Note: For more information about Red Hat OpenShift, see this [web page](#).

Conclusion

Docker and Red Hat OpenShift differ in terms of the features that they provide. When it comes to the use of the best tool for a zCX deployment, these features must be analyzed with the needs of each application at the forefront of consideration.

Docker often is used in smaller environments that do not require managing many containers because it is easier and faster to implement. For specific use cases on zCX, such as small applications, and open source single containers, Docker is a clear candidate as a deployment environment of a company's application.

However, when an application is bigger or even multiple applications must be managed from a single platform, a need often exists for Red Hat OpenShift because it gives the user the ability to keep all of these containers active, implement high availability, implement DevOps tools, and so on.



Your use case: Run anything on Red Hat OpenShift in zCX

It is likely that your use case does not exactly fit into one of the previous chapters of this IBM Redbooks publication. This chapter is designed to give you the necessary guidance to bring any use case to the platform.

We show you a turn key way to bring your use case to Red Hat OpenShift on zCX. Many of these steps can be used to also bring a solution to Red Hat OpenShift on any platform, such as Red Hat OpenShift on Linux s390X.

This chapter includes the following topics:

- ▶ “Requirements” on page 44
- ▶ “Worked example” on page 45
- ▶ “Creating a multi-architecture Containerfile and image for an x86 image to deploy on s390x architecture hosting platforms” on page 47
- ▶ “Managing the lifecycle of a containerized deployment” on page 50
- ▶ “As with any complex enterprise projects, IBM is ready to partner with you to ensure it is a success. It is highly recommended to reach out to your IBM representative directly if you have unanswered questions and areas in which you need advice.” on page 51

5.1 Requirements

You must define your use case and have a good idea of what your application components are to increase your success and simplify the steps that you need to follow in this section.

We need to understand first the components of the application that you want to run in Red Hat OpenShift. For example, if you are running an application that is well suited for customer lookup and you know it is a Java EE application, you are off to a good start.

However, more information is needed to plot the stages that are required to run the application in Red Hat OpenShift.

From an overall perspective, consider the following points:

- ▶ Have a clearly defined use case:
 - Know the dependencies of the application. Does it rely on a database? How does it connect to that database?
 - Know the availability requirements of the application. How does the application manage high availability today?
- ▶ Know the application's main runtime language, runtime server, and the operating system it uses.
- ▶ Know the application's deployment and packaging mechanism. Is the application containerized? How is deployed? How is it updated? How is it secured?

It is important to be able to answer these questions with as much detail as possible. For example, you might answer the question about runtime language with Java. Ideally, more detail.

Consider the following follow-up questions:

- ▶ What version of Java?
- ▶ What version of the JRE?
- ▶ What runtime server is being used. Liberty? Tomcat? JBoss?

As you can see, detail here is important. Therefore, good answers the previous three questions about runtime language can be:

- ▶ Java 1.8.0 OpenJDK
- ▶ IBM WebSphere Liberty 22.0.0.6
- ▶ SLES 15

Next, an example scenario is used to put our steps into practice.

5.2 Worked example

The example scenario that is used in this section helps you walk through your own use case by using a worked example.

Our example features the following four piece application (what the application does is not important):

- ▶ Front end GUI that is written in NodeJS
- ▶ Back-end server side connectivity that is based on JavaEE application
- ▶ Postgresql is used as the database
- ▶ Back-end search that is powered by Apache Solr

5.2.1 What is needed

As described in 5.1, “Requirements” on page 44, we must be explicit about the requirements of the application. We start by considering the following points:

- ▶ Have a use case defined clearly

The dependencies of the application include the Node JS front-end application that connects to a Java EE server on the back end. This server uses Postgresql as its database and relies on Apache Solr for powering the application’s search capabilities.

The requirements are server-side SSL across all connections. Each application component must be highly available, and features three points of redundancy on each component.

- ▶ The following technical details include:
 - NodeJS version 16.16 Gallium that is running on Ubuntu 22.04 x86 architecture
 - IBM Java JRE v8 with WebSphere® Liberty version 21.0.0.10 that is running on SLES x86
 - Postgres version 14 that is running on RHEL and made available by way of JDBC on x86
 - Apache Solr version 9.0.0 on ppc64
- ▶ The application’s current packaging depends on the component and includes the following components:
 - NodeJS application: A containerized application that is running in Docker across multiple VMs on x86 environment.
 - Java Liberty application: A containerized application that is running under the control of a home made Kubernetes infrastructure.
 - Postgres database application component: Manually managed system infrastructure layer (not yet containerized) that is running on bare metal.
 - Apache Solr application: Not containerized that is running on bare metal.

The information in this section is a useful guide for us as we transition the application to be packaged and run as a fully containerized application in Red Hat OpenShift on z/OS Container Extensions.

5.2.2 Preparing our application components for deployment to containers on Red Hat OpenShift on zCX

Two of our four application components are packaged as containerized applications. Therefore, adjusting those components to run on our new Red Hat OpenShift platform is fairly trivial. The other two components, notably the database and search capabilities, are not containerized.

The two application components that are not yet containerized need some more work to enable them to be fully functioning container applications instead of what they are today.

It is worth noting that all large projects are positioned for the container future bit by bit. A “big bang” approach is not useful. Instead, each of the four application components can move toward the container deployment mechanism over separate timelines. They also can coexist with or without each of their dependent application components being deployed in Red Hat OpenShift.

Note: It is important that you do not attempt to lift and shift a large multiple faceted application in one move. Doing so increases risk, complexity, and cost while slowing down technology adoption. It is wise to move components individually, and coexist the new with the old hand in hand until you have confidence in the solution.

Preparing containerized applications

We must prepare containerized applications so that they are built and optimized for deployment on our s390x architecture.

This process is done by finding the manifest that builds the container and make adjustments so that we are pulling the correct version of software components from our repositories. This manifest is normally called a *Dockerfile* and also can be known as a *Containerfile* (the terms can be used interchangeably).

In 5.3, “Creating a multi-architecture Containerfile and image for an x86 image to deploy on s390x architecture hosting platforms” on page 47, we show you how to adjust a Dockerfile to create a multi-architecture container that is ready for deployment in our new zCX for OpenShift environment.

Depending on the content of the Containerfile, we might not need to make any specific changes. Instead, we need to build of that image by using the same unchanged Containerfile with a specific argument that details the platform to which we are deploying.

It is assumed that the container is an Open Container Initiative (OCI) compliant container (most of what exists is OCI complaint). OCI represents a body of standards and governance for Containers. Red Hat OpenShift is compatible with OCI-compliant containers. If your container deployment is not yet OCI-compliant, getting to this stage is a prerequisite and not covered in this publication.

Preparing noncontainerized applications

We must prepare applications that are not yet containerized. This process entails understanding software versions, which we already know, finding pre-made container components for those software aspects, and building in our application’s compiled source code such that the container has everything it needs to run a single component of our application.

This process culminates in the creation of a manifest that represents the new application's prerequisites and the software for the application. In the following sections, we explain the process to create one of these manifests

5.3 Creating a multi-architecture Containerfile and image for an x86 image to deploy on s390x architecture hosting platforms

If a Containerfile exists, we assume that it is built to deploy to some other architecture, such as x86. Therefore, we have our starting template, the Containerfile that exists, and we must now alter it to be specific to our s390x hosting platform (in this case, OpenShift for zCX).

The process is essentially to get a copy of the Containerfile and then, make small alterations to it so that it is eligible to run on s390x before building it by using a tool, such as Podman, Buildah, or Docker.

5.3.1 Creating a s390x Containerfile and image for our NodeJS OCI-Compliant container front-end application

As an example, consider the following Containerfile for our NodeJS application that is deployed to the x86 environment.

Existing Containerfile

Example 5-1 shows a Containerfile that represents the front-end application that is running NodeJS.

Example 5-1 Sample Containerfile

```
FROM ubuntu:22.04
RUN apt-get update -y
RUN apt-get install npm -y
COPY ./ ./
RUN npm install
CMD ["npm","start"]
```

The Containerfile does not need to be changed. We only need to notify our builder that we want this image to be built for the s390x architecture.

This notification is done by using the **docker buildx** or **podman** commands. They support a flag with which you can specify the architecture for which you are building the image. An example of building this container for our s390x Red Hat OpenShift Container Platform deployment on zCX is shown in Example 5-2 for **docker buildx** and Example 5-3 for **podman**.

Example 5-2 Building a container for the s390x with docker buildx

```
docker buildx build
```

Example 5-3 Building a container for the s390x platform with podman

```
podman build --file nodejs_frontend_x86_containerfile --arch s390x --tag
nodejs_frontend_s390x
```

Adjusting the build to x86 is as simple as changing the `--arch` flag to be `amd64`. Also, if you are building for testing on an Apple M1 CPU, you use an `--arch` flag of `arm64`.

In general, `podman` and `Docker` commands are interchangeable.

5.3.2 Sharing your image to a repository

After you build your image, it is a good idea to share it with an image repository. In Red Hat OpenShift, this repository can be a project-specific repository or a centralized repository. In our example, we use a project-specific repository.

When we create a project (otherwise known as a *namespace*) in Red Hat OpenShift, it also creates for us a repository with which we can push images that belong to this area of control. We can push our image to this repository by using `podman` or `Docker` commands to transfer our image, as shown in the examples that are included in this section.

Note: The commands that are shown in this chapter are the same for building and deploying your image to any Red Hat OpenShift environment on s390 architectures. Therefore, we do not need to do anything specific now because this is part of a zCX deployment.

Example 5-4 shows the command that is used to push our s390x image to our project-specific namespace in Red Hat OpenShift by using the Pod Manager tool - `podman`.

Example 5-4 Pushing the image into a namespace specific repository by using the podman command

```
podman push lala1
```

As before, `podman` and `Docker` generally are interchangeable. However, some subtle differences exist in how `Docker` handles SSL verification of the remote repository compared to `podman`. You might need to add the full Server CA chain of trust to the `podman`, or `Docker` truststore.

Example 5-5 shows the command that is used to push our s390x image to our project-specific namespace in Red Hat OpenShift by using `Docker`.

Example 5-5 Pushing the image into a namespace specific repository by using the Docker command

```
docker push lala1
```

5.3.3 More complex examples of Containerfiles

Sometimes your Containerfile needs some s390x centric changes. You can make an s390x-specific Containerfile by using those changes and then, use a manifest to allow an image repository to have a multi-architecture image.

In this way, if you pull the image from an x86 system, the repository serves the x86 image. If you pull the image from an s390x system, the repository serves you the s390x version of the image, which is useful for the hybrid cloud. It gives your application the ability to rapidly deploy to multiple different architectures of endpoints according to the requirements at any time.

The *manifest* is a JSON file that can be built manually by using the `Docker manifest` command or instead using `Docker buildx` to create one automatically. We opt for the `Docker buildx` approach because it is a more seamless way to create a multi-arch image.

5.3.4 Containerizing an application from scratch

If your application or application components are not built into containers, coexist some elements of an application that is running in a container with their other components left untouched. This configuration reduces risk and allows you to onboard with new technologies in a stepwise manner.

After you gain some experience and want to commit further to the idea of containers, you can consider moving an application into a state ready for a container runtime platform, such as Red Hat OpenShift Container Platform.

Assuming that you took a small step, tried a component of an application that is deployed to OCP, and now decide it is a good idea to bring more of the application into this style of deployment, you can continue with the process. In our example, we had the database component, which was not containerized, and we had the Apache Solr component, which also was not containerized.

We use the Apache Solr component as our use case for turning this application element into a containerized deployment.

To begin, we must find a suitable base image from which to build our container. You can browse a repository of images, such as dockerhub, the IBM Container Registry, or your own private enterprise collections. The Apache Solr application component that we are using is Apache Solr version 9.0.0 on ppc64.

Searching the registries that are available to us, we find a suitable image exists for Apache Solr version 9.0.0 that is built for s390x on dockerhub (<https://hub.docker.com/r/s390x/solr/>).

If you do not have direct access to dockerhub from your enterprise machines, you might have a process for bringing images like this into a private registry for your enterprise.

From this starting point, we have the following options:

- ▶ Pull the image from the repository directly to our workstation
- ▶ Test the image directly by pulling from the registry into the Red Hat OpenShift platform
- ▶ Start crafting a Containerfile that bases itself from this image. We use this option.

The Containerfile must have all aspects of the Apache Solr application catered for, including runtime setup, custom configuration, and data access.

Before and during the process of building a custom Containerfile, it is essential that you consult the documentation for the base container image that you use. You normally find this documentation linked in the location from where you pull the image.

In our example, we review the documentation that is found at [this web page](#).

For this container, much of the configuration is handled by mounting a file system, which the Apache Solr run time finds important configuration files that exist for our application at run time. In this way, the bare basic container can run mostly unchanged; therefore, our Containerfile can be kept basic, and most of the changes are handled by the files that are accessed at runtime and through environment variables.

An example of a Containerfile that is basing itself from the available official Apache Solr 9.0.0 s390x image is shown in Example 5-6.

Example 5-6 Custom Containerfile for containerizing our Apache Solr installation

```
FROM s390x/solr:9.0.0
#Bring in our existing configurations
COPY ./ ./
RUN ["setup.sh"]
```

In this example, we base our image off the main Solr official image, and add files from a local directory that then are run by way of custom a configuration shell script `setup.sh`.

Many other strategies can be used here because no right or wrong choice exists. However, it is always best to keep customized configurations as shell scripts as much as possible. Also, having configuration parameters scripted and detailed in a structured way, such as a custom Containerfile, allows staff to share best practices and keep things DRY (Don't Repeat Yourself).

We can then build our image in the same way as we did for our example, which had a Containerfile, as shown in Example 5-2 on page 47. Per that example, we then push the image to our Red Hat OpenShift environment and benefit from this powerful container hosting platform.

5.4 Managing the lifecycle of a containerized deployment

Over time, you want to scale a deployment through the test environments before production. Per typical requirements, you have strategies for availability, security, updating the application, and so on. Some of these strategies might be changed processes with the opportunity to make improvements in behaviors that you used in the past.

It is important that opportunities are taken as you containerize applications to make an improvement that demonstrates measurable business impact, whether that is making software easier to update, improving security, delivering function more quickly, reducing downtime, and so on. If nothing is improved, the project is an expensive time sink without much purpose. On your journey, try to keep this idea in your mind at all stages.

Note: Managing availability, security, updating, and monitoring and all other lifecycle considerations is discussed in Red Hat OpenShift documentation and is not a topic that is covered in this IBM Redbooks publication.

5.5 Summary

The aim of this chapter was to walk you through an example of moving elements of a complex application into Red Hat OpenShift, which we did in various ways.

One case presented an example of an application that was ready for container deployment, but not built for s390x. Another case highlighted a component of an application that was not yet containerized.

Many more steps must be completed before considering an application to be fully “ported” to a containerized platform. The aim of this chapter was to examine this process and provide a starting point to help you begin to ask the right questions and head in the right direction.

As with any complex enterprise projects, IBM is ready to partner with you to ensure it is a success. It is highly recommended to reach out to your IBM representative directly if you have unanswered questions and areas in which you need advice.



Red Hat OpenShift deployment under zCX

This appendix describes the process of deploying Red Hat OpenShift Container Platform on zCX by using the IBM zCX Foundation for Red Hat OpenShift. We also describe the required supporting environment, and how we satisfied the requirements in our installation lab.

We also discuss how the z/OSMF workflows that are provided with zCX Foundation for OpenShift can be automated by using Ansible, which integrates Red Hat OpenShift Container Platform on zCX into deployment of Red Hat OpenShift across your computing environment.

This appendix includes the following topics:

- ▶ “Red Hat OpenShift Container Platform installation” on page 54
- ▶ “Components outside z/OS and zCX” on page 55
- ▶ “Our installation environment” on page 57
- ▶ “Using manual z/OSMF workflows to install Red Hat OpenShift Container Platform” on page 60

Red Hat OpenShift Container Platform installation

The installation of Red Hat OpenShift Container Platform requires the use of an external computer, which in Red Hat documentation is referred to as the *Installation Support Node* (ISN). The ISN is used to run programs that support the Red Hat OpenShift Container Platform installation, including the following examples:

- ▶ `openshift-install`, a program that generates files that are used by the installation. It also provides status information during the installation.
- ▶ `oc`, the general Red Hat OpenShift operation and management command.

The ISN does not have to be the same platform as the cluster that is being installed. However, the programs that are used on the ISN must be the correct version for the architecture of the cluster that is being installed.

Note: Always download your client support programs from the location that corresponds to the architecture on which you are installing your Red Hat OpenShift Container Platform cluster. Direct download locations for Red Hat OpenShift Container Platform support code are available from the following web page:

```
https://mirror.openshift.com/pub/openshift-v4/<arch>/clients/
```

Where *<arch>* is the architecture of the Red Hat OpenShift Container Platform cluster to be installed.

Installation source for Red Hat OpenShift Container Platform

The default installation of Red Hat OpenShift Container Platform is a “live” installation that fetches container image content directly from registries on the internet. However, most enterprise sites do not permit direct access to the internet from systems within the data center. This issue can be resolved in one of the following ways:

- ▶ Cluster-wide proxy

If internet access can be provided through a proxy server, the cluster installation can be performed by using that proxy. The configuration for how to use the proxy is included in the file that defines the cluster configuration (`install-config.yaml`); for more information, see this [Red Hat Documentation web page](#)) and the installation proceeds a normal direct-to-internet installation.

Note: It is important to ensure that the list of domains that do not use the proxy is configured correctly. If cluster-internal requests are forwarded to the proxy instead of directly to cluster nodes, the cluster might not operate correctly.

- ▶ Mirror registry

This method is an alternative to semi-connected installation by way of a proxy when no internet connectivity is possible.

Components outside z/OS and zCX

The Red Hat OpenShift Container Platform cluster that is running in zCX needs the following infrastructure support, which must be provided outside of the zCX environment:

- ▶ DNS resolution

For the cluster to build an operate, DNS must be functional. Most importantly, the cluster nodes must use DNS to resolve the IP address that is assigned to them back to a hostname (this process is called *reverse name resolution*).

- ▶ Load balancer

A TCP load balancer is used within the cluster (to distribute connections to API servers) and outside it (to manage external connections to services that are provided by the cluster).

- ▶ NTP server

CoreOS uses NTP to maintain consistent time across the Red Hat OpenShift Container Platform cluster.

- ▶ Mirror registry

If a disconnected installation is required, a container registry that supplies the images for the cluster installation must be available somewhere within the network that is reachable from the cluster nodes.

DNS resolution

The documented Red Hat OpenShift Container Platform installation process for IBM Z and LinuxONE does not specify host names in the configuration files for the cluster nodes. Instead, the cluster nodes derive their host names by using reverse DNS. Therefore, the documentation specifies that reverse DNS resolution is required for the installation; the cluster nodes cannot obtain their host names without it.

Load balancer

Part of what allows Red Hat OpenShift Container Platform to provide high levels of reliability and resiliency for applications is that the cluster maintains inbound routing of traffic through services that are known as *ingress routers*. These ingress routers manage the connectivity to application services that are running inside the cluster, and at least two are in the cluster always.

Connections to the ingress routers are distributed by using a load balancer to distribute load through the cluster nodes as evenly as possible and to maintain high availability of the ingress service.

In addition to the applications that are serviced by the cluster, the load balancer is used to distribute connections to important cluster management services that are inside the cluster. The Kubernetes API server and the Red Hat OpenShift “machine config” server service use a load balancer to present highly available services and to balance requests between instances.

Placement of the load balancing function is important for maintaining good cluster performance.

Network Time Protocol server

Consistent time recording is critical in compute environments today, and the Network Time Protocol (NTP) server is almost universally used in modern environments to provide consistent time across a computing environment. NTP is an IP-based protocol that efficiently allows time synchronization across many systems, even over the internet.

As with most other computer systems, Linux systems use NTP as a time source. CoreOS, on which Red Hat OpenShift Container Platform is based, is an operating system that is based on the Linux kernel and uses NTP.

When Red Hat OpenShift Container Platform is running under zCX, it does not inherit any time-of-day data from the z/OS system on which it is hosted. The virtual TOD clock of the zCX instance is set to match the TOD clock, which is known by z/OS when the zCX instance is started. However, after the zCX instance is started, the Linux kernel uses its own timekeeping mechanisms. Red Hat OpenShift monitors the NTP synchronization of CoreOS nodes and sends an alert if nodes are not synchronized with NTP.

Mirror registry

If a disconnected installation is required, a supported container registry is needed to host the images that are used to build the cluster.

The process that is used to build the required mirror registry is documented in Red Hat's OpenShift Container Platform documentation. The process includes the following high-level steps:

1. A supported container registry is obtained and run.
2. The registry content is downloaded.
3. The content is loaded into the registry.
4. The Red Hat OpenShift Container Platform installation configuration file is modified so that the mirror registry is used instead of the "live" registries.

Often, an intermediary device (such as an administrator workstation) is used to download the content. The system that is used to download the content must have internet access, and the **openshift-install** command that corresponds to the version of the Red Hat OpenShift Container Platform that is being installed. If the download workstation can access the internet and the mirror registry, the downloading and loading can be done in a single step.

If the download workstation does not have access to the registry (for example, the registry is in an air-gapped location), the downloaded content is transferred to the air-gapped location by using some other means. It is then installed into the registry by using the **openshift-install** command.

Our installation environment

We used a z/OS 2.5 system as part of a two-member Parallel Sysplex to build our zCX-based Red Hat OpenShift Container Platform cluster. The zCX Foundation for Red Hat OpenShift product was installed according to the documentation.

Also per the documentation, we defined the required VIPA addresses to support our zCX instances. We added an SMS storage group and management group configurations to allow the automatic allocation of the data sets (zFS and VSAM linear) that are used by the zCX instances.

Installation support node

To provide the supporting infrastructure, we used a Red Hat Enterprise Linux 8 system that is hosted by an IBM z/VM system in a neighboring LPAR. This RHEL 8 system hosted DNS, load balancer (HA-Proxy), and an NTP server, and an HTTP server to deliver the CoreOS installation files.

Because our environment can connect to the internet, we did not need to use a disconnected installation process.

Networking

Apart from the Virtual IP address (VIPA) and SAMEHOST definitions that are always required for zCX deployments, we performed some extra configuration steps to see how they operated in our environment.

HiperSockets Converged Interface

We configured our z/OS systems to use the HiperSockets Converged Interface (HSCI), which is a z/OS networking technology that allows a HiperSockets CHPID to transparently carry network traffic between systems in preference to the traffic that is sent by way of Ethernet.

Note: HSCI was introduced with z/OS V2.3 to make the use of HiperSockets more convenient and less complicated in most configurations. For more information about HSCI in z/OS V2.5, see this [IBM Documentation web page](#).

Before HSCI, enabling z/OS systems to use HiperSockets involved adding the HiperSockets to z/OS TCP/IP as a separate network. Although z/OS systems that use a dynamic routing protocol can easily use HiperSockets network in this way, other systems (such as non-dynamically routed z/OS systems and other systems, such as z/VM or Linux) cannot use the HiperSockets so easily.

HSCI allows the “separate HiperSockets network” to be eliminated and have a HiperSockets function as an overlay of the network that z/OS attaches to by using OSA-Express.

Note: HSCI functions in a similar manner to other technologies, such as SMC and Dynamic XCF (when the IBM VTAM® **IQDCHPID** start option is coded). However, some differences exist between them as well.

When **IQDCHPID** is used with Dynamic XCF, the HiperSockets CHPID that is nominated by the **IQDCHPID** start option is used to carry Dynamic XCF traffic within a CPC. However, the Dynamic XCF connection is still a separate IP network, such as traditional HiperSockets (the network addressing is set by using the **IPCONFIG DYNAMICXCF** statement in the TCP/IP Profile).

With SMC-R and SMC-D, the Shared Memory Communications path is selected by VTAM based on data that is exchanged by systems during connection establishment. As with HSCI, no separate IP network is associated with SMC. However, SMC works only with TCP traffic (therefore, it cannot be used with non-TCP traffic, such as Enterprise Extender); HSCI supports any IP protocol.

Because our installation support node was in a z/VM system on the same CPC as our z/OS systems, we made the corresponding change to z/VM to allow HiperSockets to be used for the traffic between the ISN and the cluster. The change was made to enable the z/VM HiperSockets VSwitch Bridge.

z/VM HiperSockets VSwitch Bridge

Similar to the earlier issues that were described for z/OS usage of HiperSockets, Linux use of HiperSockets under z/VM is not straightforward. A HiperSockets is an isolated network; therefore, if a guest is attached to HiperSockets to have high-performance connectivity to z/OS, it cannot connect directly to anything outside.

If it is attached to a LAN (by way of an OSA-Express card or a VSwitch) and a HiperSockets, it becomes a multi-homed system, and now must make routing decisions when it is communicating with other systems.

To avoid such complications, the z/VM HiperSockets VSwitch Bridge provides a way to attach a z/VM VSwitch to a HiperSockets CHPID. The connection allows guests of z/VM that are attached to that HiperSockets CHPID to connect to the external LAN network through the VSwitch.

In addition to the simplified connectivity for HiperSockets-attached guests of z/VM, the use of the HiperSockets VSwitch Bridge allows the guest to take advantage of QDIO Enhanced Buffer State Management (QEBSM). QEBSM is an optimization of the QDIO data transfer mechanism that greatly reduces the CPU involvement in network traffic handling by doing more of the work by using the system firmware.

For example, comparing a guest that is attached to VSwitch and one attached to a bridged HiperSockets, traffic incoming to the VSwitch guest is almost entirely processed in general-purpose CPUs (or IFLs, for z/VM systems on Linux-only environments). For the bridged HiperSockets guest, incoming traffic is copied by the system firmware from the OSA-Express card directly to the HiperSockets buffer in the guest, almost completely bypassing CPUs.

Important: Although HiperSockets can offer considerable reductions in latency and increased throughput in many situations, scenarios exist in which HiperSockets performance is affected by high CPU utilization. If the CPU utilization of your compute systems is expected to be consistently high, HiperSockets might not be the best choice for your configuration.

Load balancer considerations

A load balancer is required for Red Hat OpenShift Container Platform. Because internal traffic is balanced and the inbound connections to the cluster for application serving, it is an important issue to consider.

“External” load balancer

Although Red Hat does not mention it specifically, many guides about installing Red Hat OpenShift Container Platform imply that the load balancing function is provided by an enterprise-grade (or even internet-grade) commercial full-function load balancer. Indeed, for the qualities of service that are expected in deploying applications on zCX, such a load balancer, might be a requirement.

In the context of an Red Hat OpenShift Container Platform deployment on zCX however, be aware that the network location of such an external load balancer might affect the performance of the cluster. z/OS systems are generally connected to different (and arguably more secure) parts of the corporate network, and challenges might exist in the use of load balancing equipment, which is more commonly associated with internet-facing applications.

Because even internal connections (within the cluster) are load-balanced, an external load-balancer might introduce considerable latency to this cluster-internal traffic. If connections out to an enterprise load balancer add even as little as 5ms of latency to a connection, that addition can significantly affect the operation and reliability of the cluster.

Cluster-local load balancer

Because of the issues with added latency, a hybrid approach to load balancing must be considered. If the amount of traffic that is incoming to applications justifies the use of a large-scale load balancer, retaining a more localized load balancer to serve the cluster-internal traffic might be needed. This load balancer is configured so that it can perform all of the same functions as the large-scale load balancer, but its prime purpose is to provide the cluster-local balancing functions in a more responsive way.

For Red Hat OpenShift Container Platform on zCX, where is this load balancer deployed? One approach is to use the same technique that was used in our configuration; if a z/VM (or KVM) system is available in an LPAR nearby your z/OS, it makes an ideal location. Alternatively, you can run a load balancer, such as HA-Proxy or NGINX, in a container in zCX.

Building Red Hat OpenShift Container Platform on zCX

We installed Red Hat OpenShift clusters under zCX in two different ways. First, we used the z/OSMF Workflow interface to perform the cluster installation steps manually. Then, we used automation that is provided by the z/VM Express System Installation environment.

Using manual z/OSMF workflows to install Red Hat OpenShift Container Platform

To familiarize ourselves with the z/OSMF workflows that are provided by zCX Foundation for OpenShift, we went through the manual process of building a workflow and deploying the zCX instances containing our nodes.

However, before the zCX instances were deployed, we used the configuration techniques that are documented by Red Hat for setting up the prerequisite supporting infrastructure (see “Components outside z/OS and zCX” on page 55).

Deploying our first node: A Red Hat OpenShift Container Platform bootstrap node

After logging on to z/OSMF, we opened the Workflows function. The Actions menu contains the Create Workflow... option (see Figure A-1), which is the selection that is used to start creating the workflow creation.

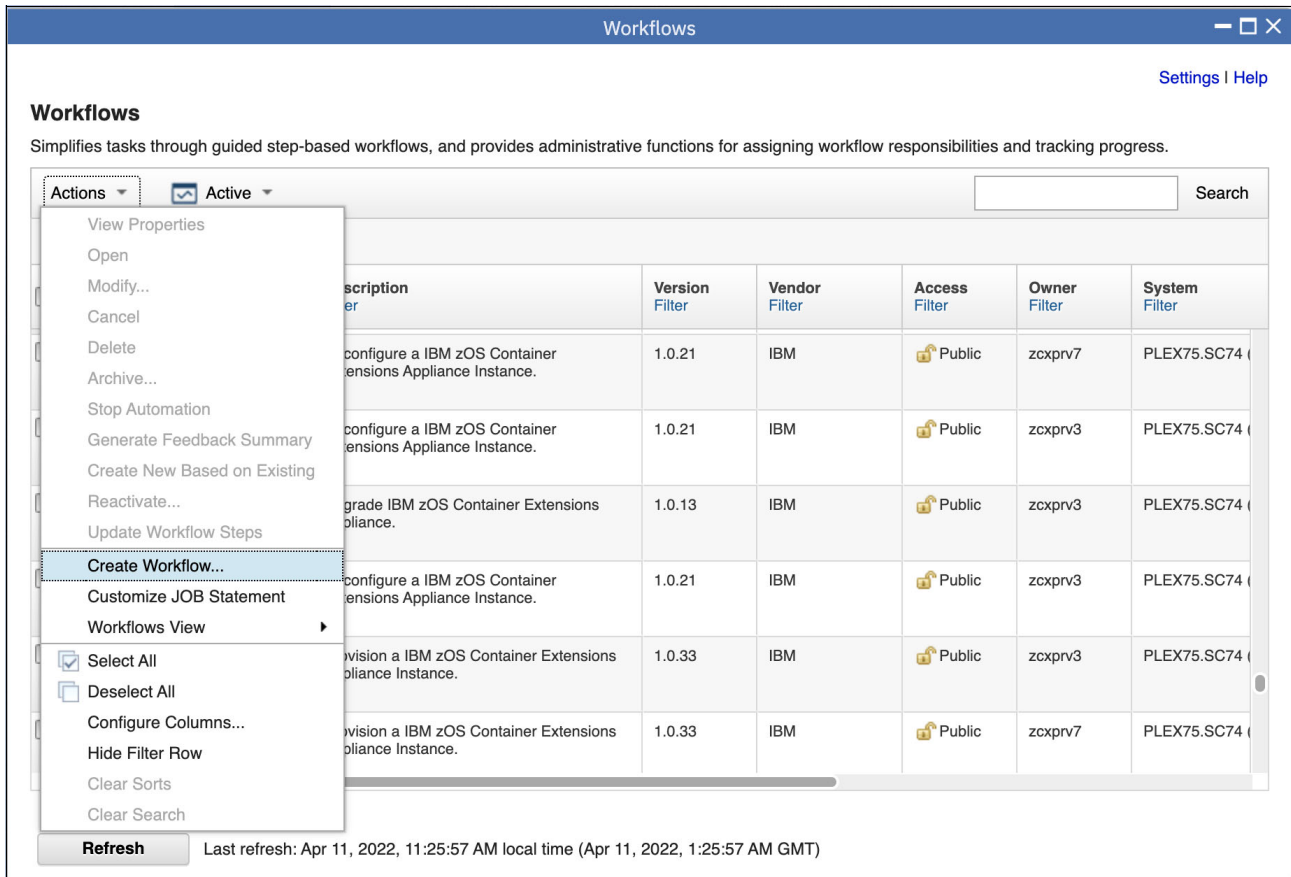


Figure A-1 Selecting the Create Workflow option

The process of creating a workflow starts with z/OSMF prompting for details about the workflow to be created. Initially, the system must know where the workflow definition file is stored (see Figure A-2).

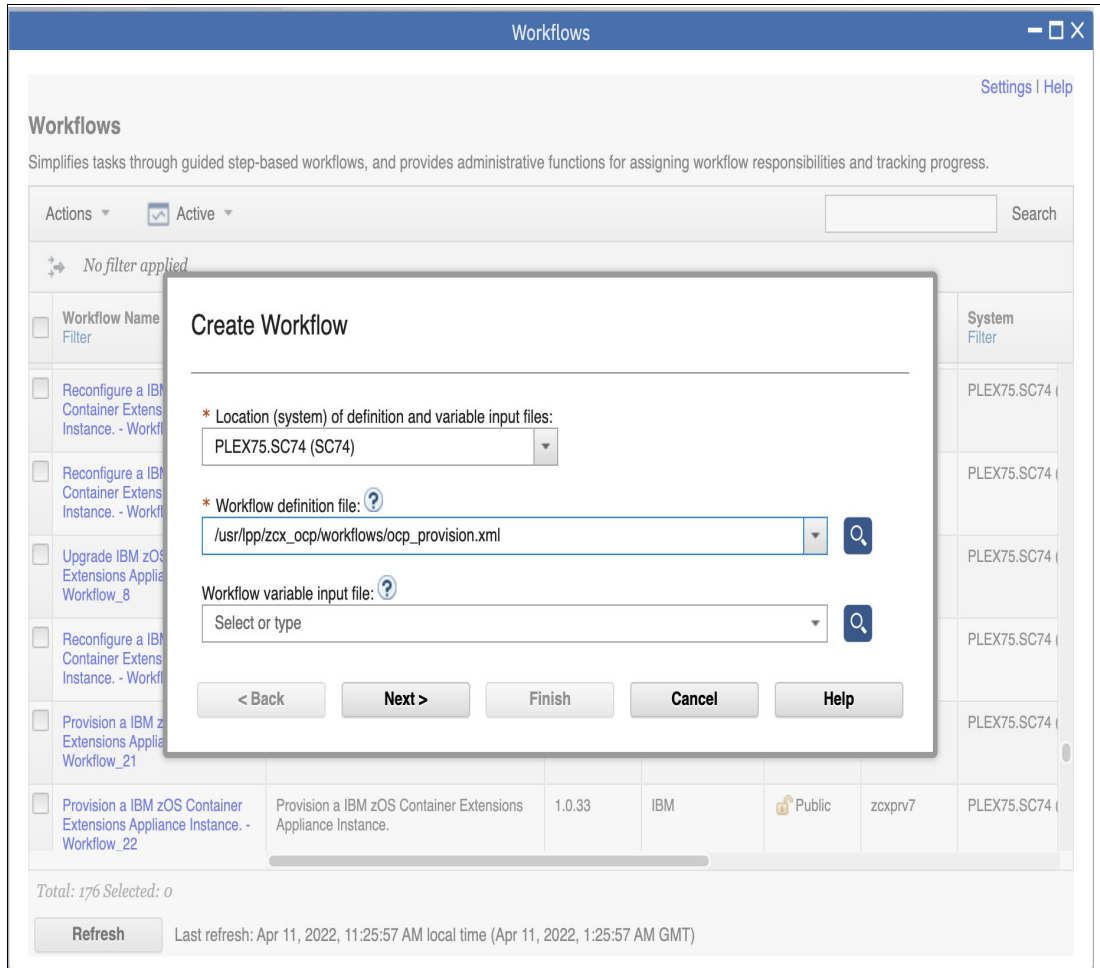


Figure A-2 Create Workflow window (initial)

The workflow definition file is provided as part of the z/OS Foundation for Red Hat OpenShift product. Because we were creating a Red Hat OpenShift Container Platform cluster node, the correct file to use was `ocp_provision.xml`. Because this workflow was the first we created, no variable input file is yet available.

Note: The z/OS Foundation for OpenShift product provides example variable input files for the workflows in the product. Therefore, it is possible to create even your first Red Hat OpenShift Container Platform provisioning workflow by using a variable file.

After we specified the workflow definition file, z/OSMF started to build the workflow. Figure A-3 shows the next stage of creating the workflow.

Create Workflow

Location (system) of definition and variable input files:
PLEX75.SC74 (SC74)

Workflow definition file:
/usr/lpp/zcx_ocp/workflows/ocp_provision.xml

Description:
Provision an IBM zOS Container Extensions for OpenShift Appliance Instance.

Vendor: Version: Is Callable: [?](#)
IBM 2.4.6 Cannot be called by another workflow

* Workflow name:

* Owner user ID: Archive SAF ID: [?](#) * System (where workflow steps will be performed):

Comments:

* Access([Learn More](#)):

Save jobs output

Open workflow on finish Assign all steps to owner user ID Delete workflow on completion

Figure A-3 Completing the Create Workflow window

We recommend giving some thought to the Workflow name. The default value is a long, descriptive text string that summarizes the workflow well. However, after running many provisioning workflows, we found that all of them used names that were based on the default *and* a number. The issue is that we had no idea which name of number was for which node. At the least, we suggest that you include the name of the Red Hat OpenShift Container Platform cluster node that is being provisioned in the workflow name that is provisioned.

The **Open workflow on finish** option controls whether z/OSMF opens the settings of the workflow after it is created. Because we wanted to configure and run the workflow immediately, we left this option selected.

We also selected the **Assign all steps to owner user ID** option so that the workflow is immediately ready to run after it is configured. The **Delete workflow on completion** option is useful as a way to automatically delete the workflow from the Workflows task when all of its steps are marked Complete or Skipped.

After the workflow was set up, we clicked **Finish**. Because we selected the **Open workflow on finish** option, the system opened the new workflow (see Figure A-4).

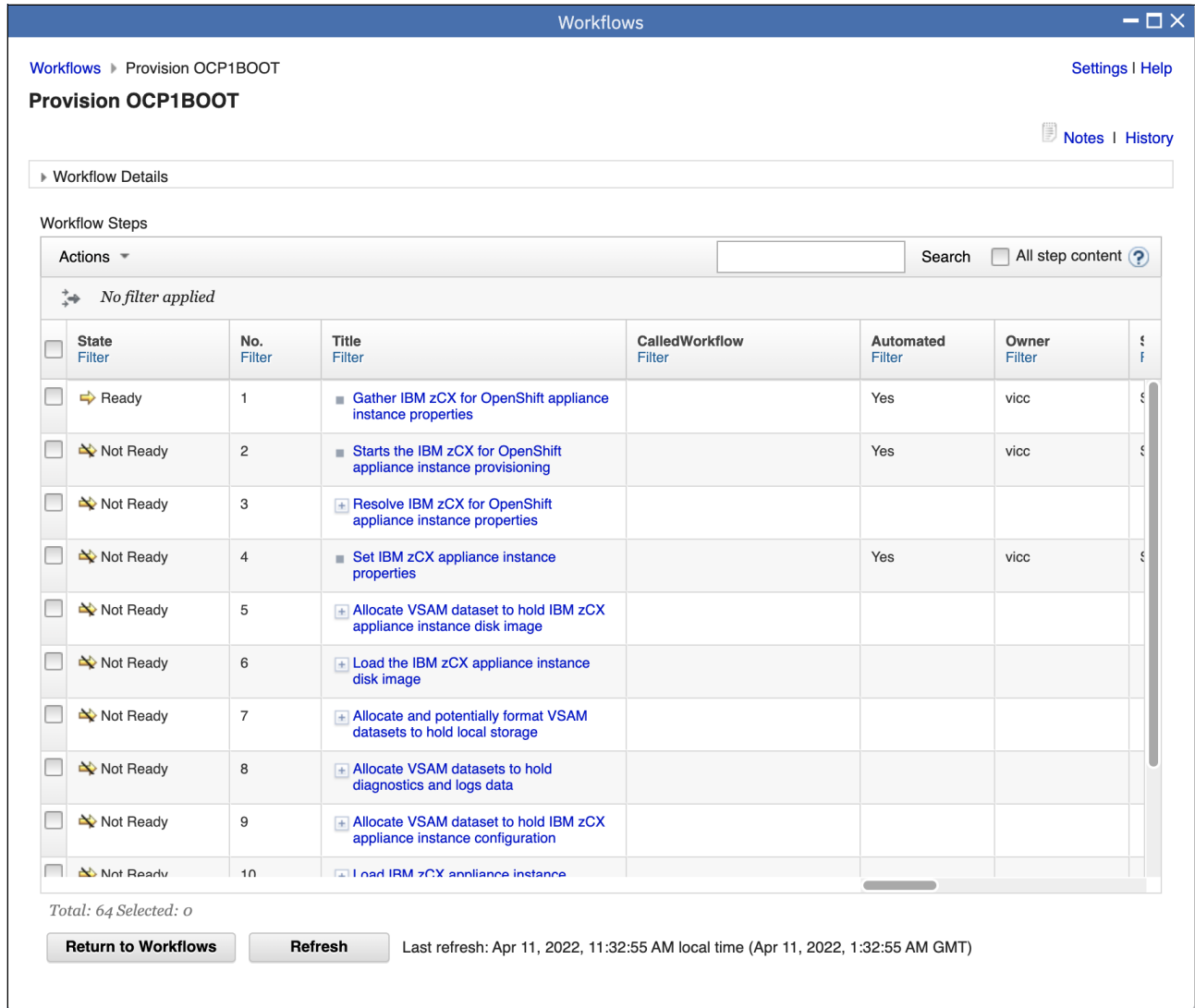


Figure A-4 Created provisioning workflow

The steps that are performed by the workflow are seen in the workflow window. Only the first step shows in Ready state (the remaining steps are shown as Not Ready). This state is shown because (as the step's title suggests) the purpose of the step is to gather settings and other data that is needed to run the workflow.

To start the data gathering, we selected the step and then, clicked **Action** → **Perform**.

Figure A-5 shows the properties of Step 1 of Provision OCP1B00T.

The screenshot shows a web-based configuration interface for a workflow step. The title bar reads 'Workflows'. The breadcrumb path is 'Workflows > Provision OCP1B00T > 1. Gather IBM zCX for OpenShift appliance instance properties'. The main heading is 'Properties for Workflow Step 1. Gather IBM zCX for OpenShift appliance instance properties'. Below this are tabs for 'General', 'Details', 'Dependencies', 'Notes', 'Perform' (selected), 'Status', 'Input Variables', and 'Feedback'. On the left is a sidebar with a tree view under 'Input Variables', including 'zCX General Configuration' (selected), 'zCX CPU and Memory Configuration', 'zCX Network Configuration', 'zCX Root Storage Configuration', 'zCX Instance Directory Storage Configuration', 'zCX Local Storage Disk Configuration', 'zCX Start and Stop Configuration', and 'Review Instructions'. The main area is titled 'Input Variables - zCX General Configuration' and contains the following fields:

- * Install Directory: - IBM zCX for OpenShift installation directory path:
- * Install Directory: - IBM zCX installation directory path:
- * zCX Instance Name: - A unique IBM zCX appliance instance name (i.e. jobname):
- * zCX Instance Registry Directory: - Directory path to store IBM zCX appliance instances related information:
- * CTRACE Member Name: - CTRACE configuration member to use:
- * Temporary DASD Unit Parm: - UNIT= parm for temporary DASD:
- * Workflows Format Disks: - VSAM data sets will be formatted by workflows:
- Directory for saving input properties file: - Directory path to save a copy of workflow input variable properties:

At the bottom of the form are buttons for '< Back', 'Next >', 'Save', 'Finish', and 'Cancel'. A 'Close' button is located at the bottom left of the window.

Figure A-5 Completing details to perform the workflow

At this step, all of the settings that are required for the workflow to provision a Red Hat OpenShift appliance instance are gathered. The following settings were required:

- ▶ zFS directories for zCX and the zCX Foundation for OpenShift
- ▶ The name of the zCX instance that is running the Red Hat OpenShift appliance
- ▶ The directory location for saving the properties file
- ▶ Memory configuration (use of large pages for the zCX instance address space)
- ▶ Allocation parameters (or SMS details) for the zFS and VSAM data sets for the zCX instance
- ▶ IP configuration, such as IP address and hostname
- ▶ Whether the instance is started when the provision workflow is complete

After the details are gathered, the step can be marked Completed. If auto-run is enabled for the remaining steps, they are run at this point (otherwise, they must be performed individually).

Note: The workflow shows you the command that you use to start the instance. Even if you set the auto-start property, you must know what the start command is so that you can start the instance in future. You might also want to write a script to start all of the instances, or include the start command into your system automation processing.

After the instance is provisioned, it must be started to complete its installation as a Red Hat OpenShift node.

Starting the instance

After the provisioning workflow is complete, the instance must be started to begin the installation of CoreOS and Red Hat OpenShift. If you selected to auto-start the instance at the completion of the provisioning workflow, it is now running. If you selected to start the instance manually, run the command to start the instance.

In SDSF, you can see the started task for your zCX instance.

Repeat for remaining nodes

The bootstrap node is the first node that must be started for a Red Hat OpenShift Container Platform cluster build. We repeated the process for the rest of the nodes to be deployed for our cluster: three control plane nodes and two compute nodes.

Installation tasks outside z/OS

After the nodes are started, the cluster builds itself. The bootstrap node downloads minimal-function versions of important cluster functions, such as the API server and machine-configuration server. These minimal services are available during the bootstrap process only.

You can use the `openshift-install` command to monitor the progress of the cluster through the remaining steps.

Console messages during installation

While the cluster installation progresses, many messages that look like error messages appear on the consoles of the control plane and compute plane nodes. Repeated messages, such as “fetch failed: EOF” and “Internal server error” appear on the control or compute plane nodes (or both).

These messages are mostly normal, and relate to the timing of events that are occurring during installation. The EOF messages relate to the control plane nodes trying to communicate with the API and machine-configuration servers before the minimal-function versions of those servers are started on the bootstrap node. The internal server error messages appear on the compute plane nodes because the minimal-function version of the API and machine-configuration servers can support only control plane nodes.

It can be useful to review the statistics page or log files of the system you use as the load balancer to put these messages in context. Before the load balancer reports that the API and machine-configuration servers are active, for example, any of the nodes that attempt to start receive an EOF message as the balancer drops the connection that cannot be served. Likewise, before the API and machine-configuration servers on the control plane nodes take over those functions, requests from the compute plane nodes receive the internal server error.

Waiting for the cluster bootstrap to complete

The control plane nodes automatically configure themselves according to the details that are provided to the bootstrap node in its Ignition file. This process is called *bootstrapping*, and after it is complete, the bootstrap node is no longer required.

The `openshift-install --wait-for bootstrap-complete` command informs that the bootstrap of the cluster is done. You can run this command as soon as the nodes are started, and it returns when the bootstrap is finished (or if a timeout occurs).

After the bootstrap is complete, the zCX instance that is running the bootstrap node can be shut down. If the disk resources it uses must be reused, it can be de-provisioned or kept and used for another Red Hat OpenShift Container Platform cluster installation.

Approving certificate signing requests

After the bootstrap is completed, the compute nodes in the cluster are provisioned. Because the control plane nodes' configuration is entirely automatic, another manual step is required for the compute plane nodes to allow them to join the cluster.

To view signing requests, issue the `oc get csr` command. To approve a signing request, use the `oc adm csr approve csr-XXXXX` command (where the last five characters are a random identification of the request).

Waiting for the cluster installation to complete

After the bootstrap node is shut down, the compute nodes successfully connect to the services that are provided on the control plane and then, begin their configuration.

After the compute node certificate requests are approved and the certificates issued, you can use the `openshift-install --wait-for install-complete` command to wait for the installation to complete.

Note: Run the `install-complete` command in a separate window so that you can still use your terminal to issue `oc` commands. The `oc` command allows you to view the details of nodes, operators, pods, and other cluster resources.

Final cluster setup

After the cluster installation is complete, the final configuration tasks can be performed, including the following examples:

- ▶ Replace the default (self-signed) ingress certificate
- ▶ Set up log in integration for the cluster, such as the use of LDAP
- ▶ Create persistent storage (at the time of this writing, NFS is supported)
- ▶ If the cluster is for proof-of-concept or other testing work, you might enable the “empty” image registry rather than one backed by persistent storage

Automatically using Ansible

Having practiced with starting the deployment workflows manually, we then automate the process by using Ansible. We started with a known-good playbook that automatically deploys a Red Hat OpenShift Container Platform cluster on a z/VM system by using the z/VM System Management API (SMAPI), and determined what needed to be changed.

Note: Ansible is a system configuration management and infrastructure-as-code tool that allows sophisticated management and control of hundreds of system components across servers and networking. By using Ansible, it is possible to apply consistent configuration across groups of devices in various parts of a network environment.

Ansible is an Open Source community project that is sponsored by Red Hat. For more information about Ansible, see [this website](#).

The original playbook performed all of the configuration that was necessary to deploy a Red Hat OpenShift Container Platform cluster, only in the context of z/VM. The most important change to be made was in the part of the playbook that interacts with the z/VM System Management APIs (SMAPI); this part was reworked to call z/OSMF workflows instead.

Automated cluster deployment tasks

All of the prerequisite external configurations (such as DNS and load balancer) were configured in the z/VM ESI system. However, a difference existed in the network configuration that required that we change IP addresses in the Ansible inventory to suit our zCX setup.

We also had to update the configuration of the BIND DNS server with the zCX IP addresses. The HA-Proxy load balancer configuration did not need to change because host names were used in the `/etc/haproxy/haproxy.cfg` file.

z/VM ESI also provides some of the support files for Red Hat OpenShift (and the underlying CoreOS) already downloaded. However, these files are for an earlier version of Red Hat OpenShift Container Platform than zCX supports. We had to download new versions of these files that corresponded to the minimum supported version (which is Red Hat OpenShift Container Platform 4.10).

Note: If you attempt to start a version of Red Hat OpenShift Container Platform before 4.10, it fails to start, with a kernel panic. Red Hat OpenShift Container Platform 4.10 is the earliest version that runs under zCX.

Cluster configuration template

A Red Hat OpenShift cluster installation starts with the settings in the `install-config.yaml` file. We compared the example `install-config.yaml` that is found in the installation documentation and saw that the settings are unchanged.

After updating the Ansible inventory with the IP addresses that were used in our zCX environment, we were confident the resulting `install-config.yaml` file set up a working cluster.

SMAPI to z/OSMF workflow

The key change that was needed was to switch from the use of calls to the z/VM SMAPI to calls to z/OSMF. For this change, we turned to the Red Hat Ansible Certified Content for IBM z Systems®, which provides Ansible modules for interacting with z/OSMF. The key module that we needed was `zmf_workflow`, which supports working with workflows through Ansible.

We used Ansible Galaxy to install the Ansible collection that contains the modules that support z/OSMF, as shown in Example A-1.

Example A-1 Installing the `ibm_zosmf` collection by using Ansible Galaxy

```
$ ansible-galaxy collection install ibm.ibm_zosmf
Process install dependency map
Starting collection install process
Installing 'ibm.ibm_zosmf:1.1.0' to
'/home/support/.ansible/collections/ansible_collections/ibm/ibm_zosmf'
$
```

Note: Ansible Galaxy is an open repository of community-created modules for Ansible. Developers who create a module for Ansible can submit their work to Ansible Galaxy, which makes it available to the rest of the Ansible community.

The next challenge was to create the workflow. On IBM z/VM®, the virtual machines are created dynamically as part of the Ansible automation. We wanted to do the same thing with z/OSMF; that is, create the workflows as part of the automation, rather than create the workflows on z/OS and submit them from Ansible only. To create the workflows as part of the automation, we looked closer at the workflow creation process from when we performed it manually.

In the manual runs, each workflow featured a corresponding workflow definition file. The workflow creation task (what we interacted within the first step in the GUI) created this workflow definition file, which was read by the subsequent steps in the workflow execution.

We copied one of the workflow definition files to our Linux system and inspected it. As a text file with no extravagant formatting, we found it easy to convert the file we copied into an Ansible template by using the correct Jinja2 syntax.

Note: Jinja2 is a template language that is used extensively in Ansible and other fields related that are to the Python programming language. It has an extensive syntax, but in its simplest form it provides easy methods for substituting variables in text files.

A portion of the converted workflow definition file is shown in Example A-2.

Example A-2 Part of a z/OSMF workflow definition file converted into an Ansible template

```
# Specify a unique instance name
ZCX_INSTANCE : {{ cluster_nodes[coreos_role][item].guest_name }}

# Specify the type of OpenShift node
ZCX_OPENSHIFT_NODE : {{ cluster_nodes[coreos_role][item].node_type }}

# Provide the URL for the Ignition file (watch out for line wrap)
ZCX_RHCOS_IGNITION_URL : http://{{ isn_public_ip_address }}:8080/ignition/{{
cluster_nodes[coreos_role][item].ign_profile }}
```

In Jinja2 syntax, the `{{ }}` brackets enclose areas where variables are substituted. Of all the fields in Example A-2, the `{{ isn_public_ip_address }}` value is replaced by the content of the inventory variable `isn_public_ip_address`. The other fields are slightly more complicated, being references to special values held in the inventory by using YAML dictionaries.

The definition of the dictionaries is shown in Example A-3 and helps to explain the template.

Example A-3 Portion of the Ansible inventory, which shows dictionaries

```
guest_pfx: "0CP1"
cluster_nodes:
  bootstrap:
    bootstrap-0:
      guest_name: "{{ guest_pfx }}BOOT"
      ip: "129.40.23.200"
      disk: "dasda"
      node_type: bootstrap
      ign_profile: bootstrap.ign
    control:
      control-0:
        guest_name: "{{ guest_pfx }}CTL0"
        ip: "129.40.23.201"
        disk: "dasda"
        node_type: "control-node"
        ign_profile: master.ign
      control-1:
        . . .
      control-2:
        . . .
    compute:
      compute-0:
        guest_name: "{{ guest_pfx }}CMP0"
        ip: "129.40.23.204"
        disk: "dasda"
        node_type: "compute-node"
        ign_profile: worker.ign
      compute-1:
        . . .
```

A dictionary is like an array where the index to the array does not have to be a simple integer value (perhaps like a stem variable in REXX). Example A-3 shows the following Jinja2 and Ansible concepts:

- ▶ A simple variable `guest_pfx` is set, which contains the prefix characters to be used for the names of the zCX instances.
- ▶ A dictionary that is named `cluster_nodes` is defined, which has three entries: `bootstrap`, `control`, and `compute`. Each of these entries is dictionaries.
- ▶ The `cluster_nodes["bootstrap"]` dictionary has a single entry, `bootstrap-0`. This dictionary includes several entries that define instance attributes.
- ▶ The `cluster_nodes["control"]` dictionary has three entries (only one is shown in full in Example A-3). The names of the entries in the contained directories are the same as the entries in the `bootstrap-0` dictionary.
- ▶ The `cluster_nodes["compute"]` dictionary includes two entries (only one is shown in full in Example A-3). Again, the entries in the subordinate dictionaries are the same as all the other subordinate dictionaries.
- ▶ The `guest_name` entry contains the `guest_pfx` variable, which shows that variables can be “nested” inside the inventory (the names `guest_pfx` and `guest_name` are carried over from z/VM; `guest_name` is used as the name of the zCX instance, but we did not need to change the name of the variable).

The inventory and the template are brought together by the Ansible playbook, part of which is shown in Example A-4.

Example A-4 Segment of Ansible playbook for zCX instance creation

```
. . .

- name: boot the bootstrap node
  include_tasks: zcx-workflow-create-instance.yml
  vars:
    coreos_role: bootstrap
  with_items: "{{ cluster_nodes[coreos_role] }}"

- name: boot the control nodes
  include_tasks: zcx-workflow-create-instance.yml
  vars:
    coreos_role: "control"
  with_items: "{{ cluster_nodes[coreos_role] }}"

- name: boot the compute nodes
  include_tasks: zcx-workflow-create-instance.yml
  vars:
    coreos_role: "compute"
  with_items: "{{ cluster_nodes[coreos_role] }}"
  when: cluster_nodes.compute is defined

. . .
```

A task definition file shown in Example .

Example A-5 Ansible task definition file containing z/OSMF specific interactions

```
# zcx-workflow-create-instance.yml
# Watch out for line wrap!
#

- name: template the workflow definition file
  template:
    src: workflow_variables.openshift.properties.j2
    dest: "{{ workdir }}/{{ cluster_nodes[coreos_role][item].guest_name
}}-user.properties"
    mode: 0644

. . .

- name: Authenticate with z/OSMF server
  ibm.ibm_zosmf.zmf_authenticate:
    zmf_host: "wtsc74.pbm.ihost.com"
    zmf_port: "2443"
    zmf_user: "{{ zmf_user }}"
    zmf_password: "{{ zmf_password }}"
  register: result_auth

- name: start a zCX workflow to create an instance
  ibm.ibm_zosmf.zmf_workflow:
    state: "started"
```

```

    zmf_credential: "{{ result_auth }}"
    workflow_owner: "{{ zmf_user }}"
    workflow_name: "ansible_workflow_{{
cluster_nodes[coreos_role][item].guest_name }}_create"
    workflow_file: "/usr/lpp/zcx_ocp/workflows/ocp_provision.xml"
    workflow_file_system: "SC74"
    workflow_vars_file: "/global/zcx_zos/ocp_properties/{{
cluster_nodes[coreos_role][item].guest_name }}-user.properties"
    workflow_host: "SC74"

```

The Ansible tasks that are defined in Example on page 70 are those tasks that are relevant to talking to z/OSMF. Including these tasks in a separate task file is an important factor in the reusability of the task definition.

The playbook that contains the lines that are shown in Example A-4 on page 70 also contains all the other tasks that are required for creating a Red Hat OpenShift Container Platform cluster (generating the cluster configuration file, generating ignition files, and so on). When the playbook performs all of the prior tasks and reaches the task that is named “boot the bootstrap node”, the following actions occur:

- ▶ The module `include_tasks` tells Ansible to find the file `zcx-workflow-create-instance.yml` and get ready to process the contained tasks.
- ▶ The `vars` parameter on the task tells Ansible to set the variable `coreos_role` to the value `bootstrap` during the execution of the included tasks.
- ▶ The `with_items` parameter on the task tells Ansible that it should run the tasks once for each value that is defined in the dictionary `cluster_nodes[coreos_role]`, setting the value of a variable that is called `item` to that value when it runs the task.
- ▶ The first time through, with `coreos_role` set to `bootstrap`, the dictionary `cluster_nodes[“bootstrap”]` contains only one item. Therefore, the tasks in the included file are run once, with `item` set to `bootstrap-0`.

The first task in `zcx-workflow-create-instance.yml` uses the built-in Ansible `template` module to generate our customized workflow definition file. The `template` module reads the template file that we created by using the original workflow definition file (with the Jinja2 code included). With no programming necessary, Ansible writes out a copy of the template file to the z/OS system, with all the Jinja2 fields replaced by values from the Ansible inventory.

The `dest` field in the task specifies the name of the file that is to be written to the local file system. The name of the file that is created is `0CP1B00T-user.properties`. The name is generated by using the following process:

1. The variable `coreos_role` is set to `bootstrap` (from the `vars` parameter) and `item` is set to `bootstrap-0`, so the first expansion of those variables yields:

```
cluster_nodes[“bootstrap”][“bootstrap-0”].guest_name
```
2. In the dictionary, that value is `{{ guest_pfx }}B00T`, which yields `0CP1B00T` after `guest_pfx` is substituted.
3. This value is placed into the `dest` variable (along with the `workdir` variable, which sets the path) to create the file name.

Next in the task file is transferring the file to z/OS (we have not shown this process here because the exact method you might use can differ from ours).

After the workflow definition file is transferred to z/OS, the z/OSMF work can be done. The `zmf_authenticate` module performs a log on to z/OSMF. In our case, we performed a prompt of the user for the values of `zmf_user` and `zmf_password` at the beginning of the execution of the playbook, but many other valid approaches are available.

Note: Ansible provides a secure file and value store that is called Ansible Vault, which is keeps inventory and other data securely encrypted. Vault even allows files that contain encrypted data to be stored in public repositories while still maintaining security (if the password to the vault is not also kept in a public repository).

If the login is successful, the value that is returned by `zmf_authenticate` is an authentication token that is used for subsequent z/OSMF operations. Ansible allows the output of modules to be “registered”, which means stored in a variable for later reference. We register the output of the authentication module and then proceed.

Next, the `zmf_workflow` module creates the workflow. There are three critical values in this task:

- ▶ `workflow_file`, which is the file that is provided by zCX Foundation for Red Hat OpenShift that defines the actions that are performed by the workflow
- ▶ `workflow_vars_file`, which was the file we created by using the template task
- ▶ `state: started`: whether z/OSMF runs the workflow or defines it (not to be confused with the auto-starting of the zCX instance after the workflow runs)

The task also defines other attributes, such as the owner of the workflow and the host on which it is to run.

If this task is successful, the bootstrap node is running and starts its build. The main playbook then proceeds to the next task, which is starting the control nodes. This process works in the same way as the previous run for the bootstrap node, except that for the control plane the dictionary `cluster_nodes["control"]` has three entries instead of only one. Therefore, the task definition file `zcx-workflow-create-instance.yml` is included and run three times (once for each node that is defined in the dictionary).

After they are all run, the playbook reaches boot the compute nodes. Again, as with the compute node section, it expands the task definition file once for each node that is defined. However, to support so-called “3-node” clusters (which have no compute nodes), an extra parameter skips the task if the `cluster_nodes["compute"]` dictionary has no entries.

At this point, all the z/OSMF work is done. The playbook continues on with the `wait-for-bootstrap-complete` task.

Related publications

The publications that are listed in this section are considered particularly suitable for a more detailed discussion of the topics that are covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide more information about the topic in this document. Note that some publications that are referenced in this list might be available in softcopy only:

- ▶ *Getting started with z/OS Container Extensions and Docker*, SG24-8457
- ▶ *IBM z/OS Container Extensions (zCX) Use Cases*, SG24-8471

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

The following publications also are relevant as further information sources:

- ▶ *zCX Foundation for Red Hat OpenShift, Installing a Red Hat OpenShift Cluster on z/OS with zCX*, GC31-5709
- ▶ *z/OS Network File System Guide and Reference*, SC23-6883

Online resources

The following websites also are relevant as further information sources:

- ▶ Persistent storage using NFS:
https://docs.openshift.com/container-platform/4.10/storage/persistent_storage/persistent-storage-nfs.html#persistent-storage-nfs
- ▶ Installing a cluster on IBM Z and LinuxONE:
https://docs.openshift.com/container-platform/4.6/installing/installing_ibm_z/installing-ibm-z.html
- ▶ Deploying Bastion Infrastructure Services for IBM zCX for OpenShift:
<https://community.ibm.com/community/user/ibmz-and-linuxone/blogs/allen-zhou/2022/08/10/deploy-ocp-bastion-infrastructure-services-on-zcx>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



SG24-8528-00

ISBN 0738460877

Printed in U.S.A.

Get connected

