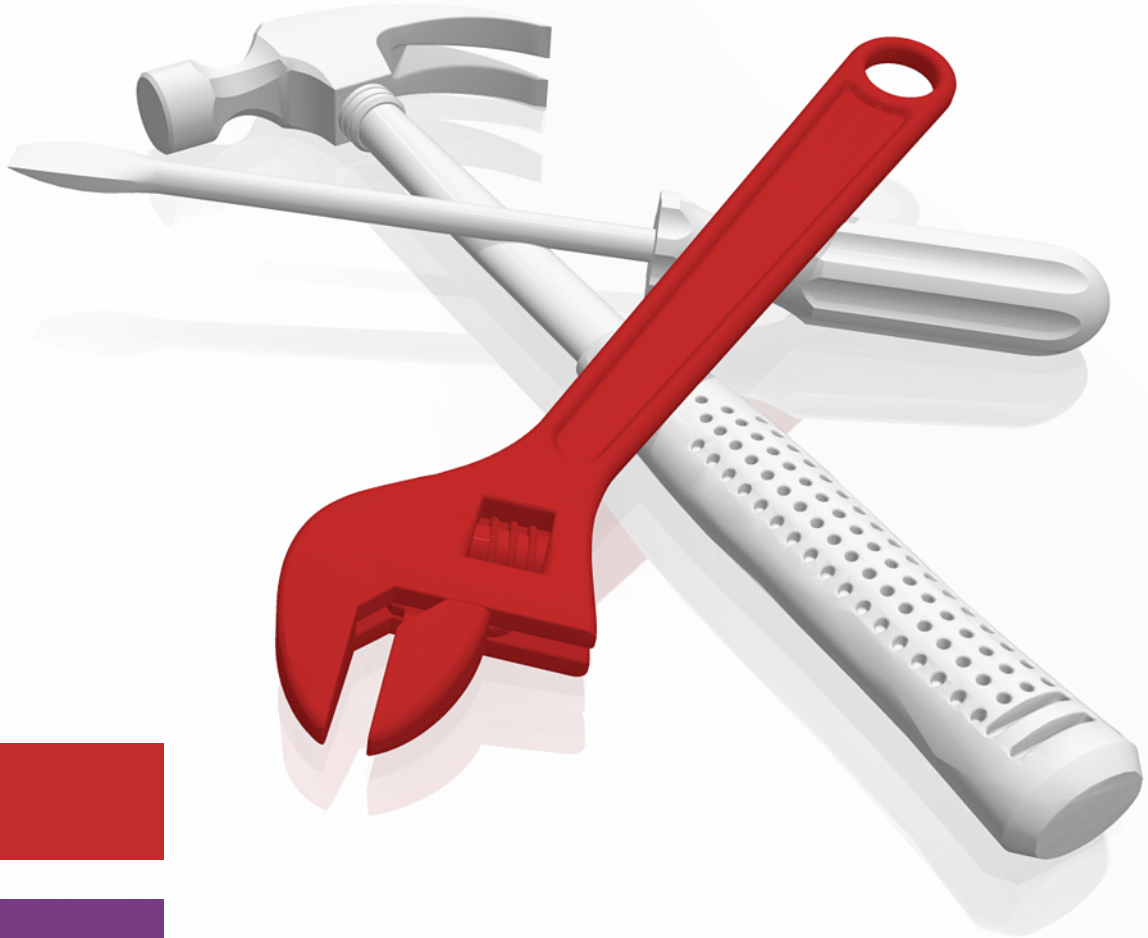
Draft Document for Review January 5, 2023 12:17 pm SG24-8532-00

Redbooks
ibm.com/redbooks

Hybrid Cloud with IBM Z

Lydia Parziale
Yinka Adesanya
Elton de Souza
Peter Haumer
Sandor Irmes
Amey Patil
Lauren K Li
Liyong Li
Filipe Miranda
Sidney Varoni Jr.



Cloud

IBM Z

IBM

Redbooks

IBM Redbooks

Hybrid Cloud with IBM Z

January 2023

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

First Edition (January 2023)

This edition applies to IBM z/OS 2.5, IBM z16, Red Hat OpenShift Container Platform 4.10.12, Instana, IBM Z and Cloud Modernization Stack 2022.

This document was created or updated on December 22, 2022.

Contents

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

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 Garage™	Rational®
Db2®	IBM Watson®	Redbooks®
FICON®	IBM Z®	Redbooks (logo) ®
GDPS®	Instana®	UrbanCode®
IBM®	Jazz®	z Systems®
IBM Cloud®	Parallel Sysplex®	z/OS®
IBM Cloud Pak®	POWER®	z/VM®

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.

Zowe, are trademarks of the Linux Foundation.

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

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

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

VMware, and the VMware logo are registered trademarks or trademarks of VMware, Inc. or its subsidiaries in the United States and/or other jurisdictions.

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

Preface

Today’s enterprises continue to digitally transform their business to modernize their core IBM® z/OS® applications on a hybrid cloud. This means they require simple yet highly secure access to their IBM Z® applications and data through application programming interfaces (APIs) in a timely fashion.

To meet this need for modernization which includes agility, maximized value of their investments and faster innovation, they are choosing a hybrid cloud strategy that offers a single integrated operating model.

IBM Z integrated into a hybrid cloud platform based on Red Hat OpenShift provides resiliency, security, scale and data gravity provides a greater infrastructure effiieny and lower compliance costs. You can accelerate application modernization today by using architectural patterns that are building blocks and process lifecycle enablers to learn how to implement and deploy them in an IBM Z environment and determine which circumstances are best for your enterprise’s applicaiton modernization.

This IBM Redbooks® publication discusses and demonstrates approaches to modernization when adopting a hybrid cloud on the IBM Z platform. We discuss and demonstrate application centric, data integration/access centric, and event driven modernization patterns. Additionally, we provide a chapter on modernizing an enterprise’s DevOps with patterns and a chapter on managing your applications.

Finally, we conclude with a demonstration of deployments of production applicatons.

This IBM Redbooks publication provides information for IT Architects, IT Specialists and system administrators.

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

Yinka Adesanya is a Chief Architect with a focus on cloud and performance at IBM in New York, USA. He has over 12 years of professional experience in the information technology industry. He holds a Bachelor's degree in Electrical/Electronic Engineering and a Master's degree in Computer Information Systems. His areas of expertise include private/public clouds, application/infrastructure performance and delivering solutions at scale. He also has keen interests in security, networking and blockchain.

Elton de Souza is an STSM for Hybrid Cloud Adoption Acceleration based out of IBM Canada. He has worked on the IBM zSystems platform his entire 12 year career at IBM and currently leads Hybrid Cloud adoption as part of the Hyper Protect Services organization. The first half of his career was spent on the internals of Java where he worked on exploiting 200+

hardware instructions on Z for mission critical mobile/cloud workloads. He was one of the first technical experts for Docker/Kubernetes on IBM Z in early 2015 and since then has worked with IBMs largest clients on successful adoption of cloud native technology like kubernetes, IBM Cloud Private and most recently Red Hat OpenShift & IBM Cloud Paks and IBM Hyper Protect Services. He has over 50 publications including patents, books, analyst reports, peer reviewed reference architectures etc, won several awards including external industry awards and contributes to several open source projects.

Peter Haumer is a development lead and senior technical staff member for the IBM Wazi product platform, which comprises of tools and solution packages for z/OS Cloud-native DevOps. He is located at the IBM Silicon Valley Labs in California. He has 40 years of software development experience. For IBM he has worked in the capacity of an agile software development team lead, full-stack software developer, and senior researcher. He has a proven track record of creating and releasing all new software products end-to-end. In recent years, he led the development of several IBM offerings such as IBM Wazi, IBM Z Open Editor, IBM User Build, IBM Application Delivery Intelligence, Jazz® Global Configuration Management, Jazz Reporting Service, reporting components in the IBM Rational® Quality Manager, IBM Self-Check, IBM Rational Method Composer, and the Eclipse Process Framework. He holds a degree of Dr rer. nat. from RWTH Aachen, Germany.

Sandor Irmes is a senior IT architect in Hungary who provides Linux on IBM Z / LinuxONE consulting services within the EMEA Z Lab Services. He has more than 30 years of experience in IBM POWER® and Mainframe server technology and several years of experience in Linux on IBM Z and open source. His areas of expertise include hybrid cloud solutions, infrastructure, and platform solutions, and competencies including network, and Linux. Sandor is an IBM Certified IT Architect and has worked for IBM for over 16 years in various technology areas.

Lauren K Li is a DevOps Transformation Specialist on IBM Systems Z DevOps Acceleration Team, which helps clients begin their mainframe modernization journey by integrating IBM technologies with Git-based CI/CD pipeline solutions. In her four years with IBM, she has also contributed code and documentation to the open-source Zowe Explorer Visual Studio Code extension, as well as the IBM Wazi product platform. Lauren holds a master's degree in Information Science from the University of North Carolina at Chapel Hill, and has special interests in front-end software development and user experience (UX) design and research.

Liyong Li is a Certified Consulting IT Specialist and Open Source enthusiast. Liyong is the technical lead within the IBM Technology Lifecycle Services organization in ASEAN, helping customers in adopting new technologies, such as hybrid cloud solution on IBM Z and LinuxONE, Red Hat OpenShift Container Platform and Ansible Automation Platform. He has been with IBM for 17 years, and holds a degree in Computer Science. He has written and contributed to several IBM Redbooks publications on Cloud, z/VM® and Linux.

Filipe Miranda is a Principal Technical Specialist for Hybrid Cloud on IBM Z and LinuxONE member of the zAcceleration team (ZAT). Filipe brings more than 15 years of experience working with open-source technologies. He lives in the United States of America with his family, holds a bachelor's degree in System Information with many academic specializations in telecommunications and computer networks. His expertise includes, Linux (on x86, IBM Power and IBM Z/LinuxONE), Virtualization (KVM/z/VM, VMware, Xen, and other hypervisors), Containers (Docker, Podman, Cri-O), Kubernetes, OpenShift (on x86, IBM Power and IBM Z/LinuxONE), many products from IBM and Red Hat.

Amey Patil is an Application Architect for the IBM CIO team located in Raleigh, North Carolina. He has 15+ years of experience in IT consulting and software development. He holds a Masters degree in Software Engineering from Birla Institute of Technology & Science, Pilani, India. His areas of expertise include Cloud computing, Blockchain, Business Process

Management (BPM), Service Oriented Architecture (SOA), API & Microservices, Enterprise Application Integration, Middleware.He has written extensively on Chapter 4, “Event-driven architecture with z/OS” on page 43.

Sidney Varoni is an IBM Z Technical Sales Specialist in IBM Australia. He has 16 years of experience in the IT Infrastructure field, including IBM Z and Storage solutions, having worked on IBM Global Technology Services, IBM Storage and currently at IBM Z business unit. He holds a bachelor degree in Computer Science from Faculdade Politecnica de Jundiai, Brazil, and holds an MBA from FGV, Brazil, with an extension in IT Innovation and Leadership from MediaX at Stanford University, CA, US. His areas of expertise include High-Availability and Business Continuity solutions, as well as performance analysis. He has co-authored several IBM Redbooks publications on IBM GDPS®, IBM z/OS and IBM Storage solutions.

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

Robert Haimowitz
I**BM Redbooks, Poughkeepsie Center**

Patrik Hysky
IBM Redbooks, Austin Center

Tom Ambrosio, Bill Lamastro,
IBM CPO

Mythili Venkatakrishnan, Nasser Ebrahim, Dennis Behm, Nicolas Dangeville, Mike Fulton
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

1

Introduction

This chapter discusses reasons for adopting a Hybrid Cloud approach to IT infrastructure using the IBM Z platform. It also introduces patterns that will be covered more in-depth throughout this book, in examples of using Hybrid Cloud on IBM Z to resolve real business challenges.

This chapter is organized as follows:

1.1 What does hybrid cloud mean in 2023?

"Cloud is not a place, it's an experience"

The cloud was originally an umbrella term for resource capacity (primarily compute) either on-premises behind a corporate firewall, or on infrastructure managed by a 3rd party and beyond company premises (i.e., public cloud). This form of compute delivery is known as Infrastructure as a Service (or IaaS). Since then, cloud has evolved to higher layers of the architecture stack i.e., platform, service, functions etc. all offered in an as-a-Service model. This offers higher layers of abstraction which enables rapid innovation and accelerated go to market for products and services. It also shifts responsibility for uptime SLAs to a provider.

There are several vendors in the public cloud space, each with varying capabilities to meet client demand. Some focus on providing the widest set of services or building blocks, others that focus on specific workloads e.g., analytics, highly regulated industries, confidential computing, hardware-based workload accelerators etc. IBM focuses on highly regulated industries (e.g., our Financial Services Cloud or FS Cloud), thanks to our industry leading confidential computing capabilities and evolved operations around regulatory controls.

Cloud Native is another term (wrongly) used interchangeably with cloud. Cloud Native is not a technology or tool but a set of characteristics that was originally offered only through the public cloud but extended to hybrid cloud environments and to environments that would not be traditionally considered to be in the purview of cloud. While there is no industry standard definition of cloud native, some of the characteristics typically include:

- elasticity (both vertical and horizontal)
- source control managed infrastructure as code and platform configuration
- DevOps (the tooling and the culture of DevOps)
- software packaged in a way that enables easy reproducibility (e.g., containers)
- microservice architecture (only where it makes architectural sense!!)
- BYOL (bring your own language aka polyglot development)
- no hard dependencies on the base/host operating system (outside of container base images)
- end to end automation where feasible
- standardized access points via industry standard protocols (e.g., HTTPS, json, protobuf, gRPC, Avro etc.)
- separation of state (typically implemented via stateful containers consumed by stateless containers) etc.

Figure 1-1 shows a sample hybrid cloud topology based on the IBM Hybrid Cloud Platform (Red Hat OpenShift)[1].

[1] https://www.ibm.com/cloud/architecture/decision-guides/container-workload-hybrid-cloud/overview

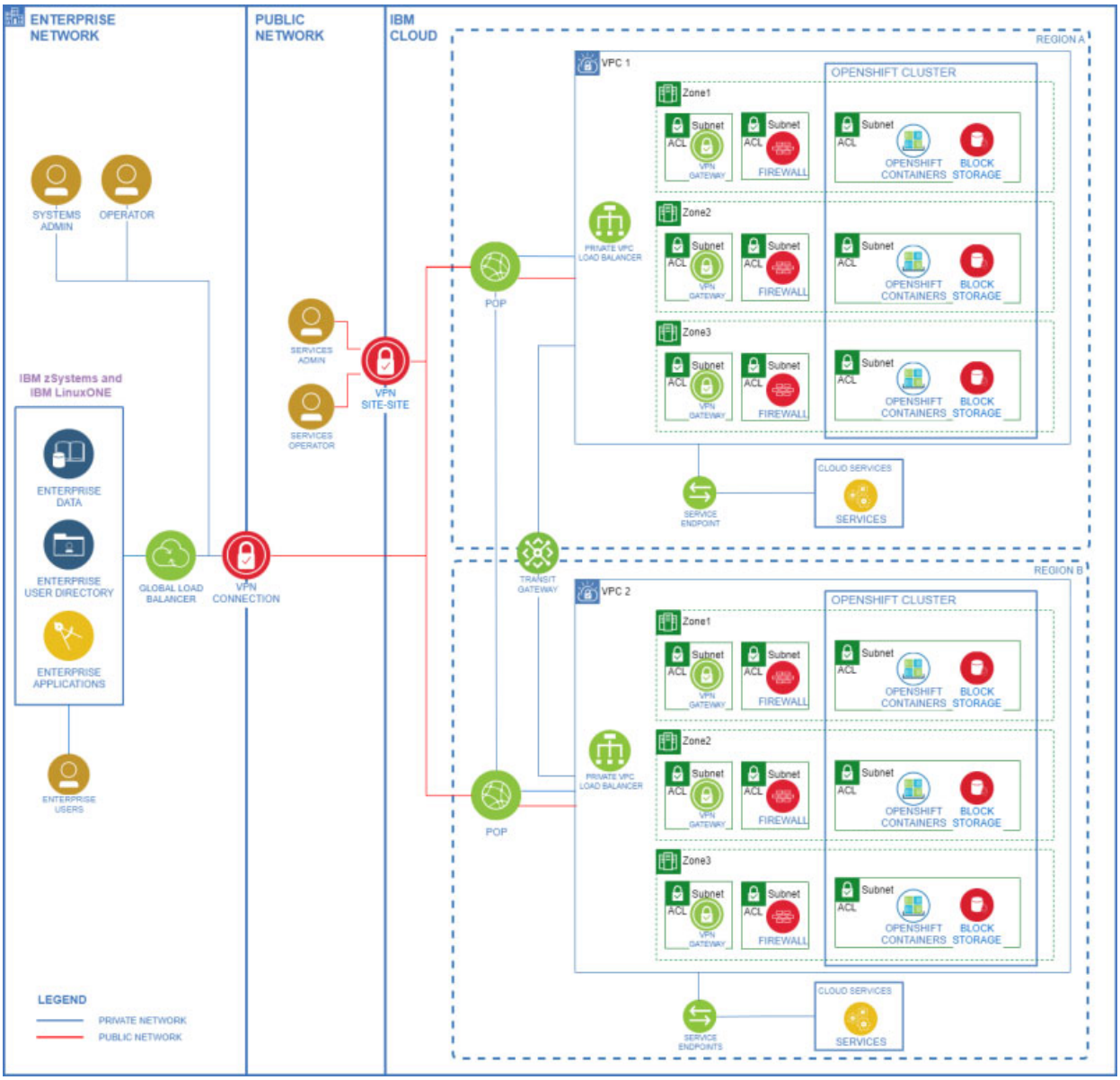


Figure 1-1 Sample topology overview

An application deployed on the public cloud may not be cloud native, and conversely an app deployed on-premises in a heritage environment (e.g. VMWare, z/OS), could be cloud native. Cloud native implies the set of characteristics versus specifying a specific underlying platform/infrastructure as a prerequisite - though picking the right underlying platform could reduce friction when it comes to execution and operations.

Multi-cloud is the term used for the consumption of multiple clouds (either a vendor for public cloud or on-premises). On average 93%[2] of organizations use multi-cloud as part of their enterprise IT strategy. This includes SaaS, FaaS, PaaS and IaaS. Multi-cloud (or hybrid multi-cloud) provides challenges around data sovereignty, normalized security across all the vendors, identity, and access management, standardized DevSecOps, mismatched SLAs etc.

[2] https://www.nutanix.com/theforecastbynutanix/technology/why-the-public-vs-private-cloud-debate-rages-on-in-the-hybrid-multicloud-era

For core business critical applications, SLAs including throughput, latency, availability, and uptime become important. While throughput can often be improved by horizontally or vertically scaling, general latency is typically a function of the network and cannot always be solved by delegating more resources to a task. Placement of the business presentation layer relative to the integration and database layer become very important - too far and the steady state latency or the occasional latency spike could infringe on SLAs. The problem is compounded in a hybrid cloud context as it is rarely possible to lift and shift an entire application stack in one step and layers are moved incrementally, often known as the strangler pattern - which negatively impacts business SLAs.

In cases where latency impacts user experience, it often makes sense to modernize to cloud native in-place vs lift and shift incrementally to a new location. This offers the best balance of agility, time to market and SLAs while minimizing operational or compliance risk. In cases where latency isn't as critical, the strangler pattern may be a good fit.

This IBM Redbooks publication provides a use-case driven approach to architecting various hybrid cloud solutions to optimize for enterprise KPIs while maintaining risk. While most books assume the same underlying architecture (typically amd64), modern cloud designs include alternative architectures like arm64, s390x, ppc64le etc. as each ISA/hardware architecture offers unique benefits.

Modernization on IBM zSystems typically benefits from in-place modernization to best leverage decades of investment by organizations into IP critical to their operations on this platform. There is also program-ming language modernization which could be as simple as upgrading compilers from COBOL 4 to COBOL 6[3]. to benefit from modern programming constructs and improved performance[4], or more involved like migrating a heritage language to Java, JavaScript, Go etc. In both scenarios, the consumption of a developer friendly IDE that supports the source and target language is beneficial. Tools like the products under the IBM WAZI[5] portfolio enable portable development and testing of application on-premises or in the public cloud. There are several other techniques like API extension of applications, data virtualization, caching, event-based architecture etc. that are also performed to modernization applications. To accelerate modernization, IBM has created an all-inclusive stack named the IBM zSystems and Cloud Modernization stack that greatly simplified procurement of the tools necessary for modernization and the following chapters will describe how do implement a subset of those patterns.

1.2 The value of the Hybrid Cloud approach

The journey to the Cloud is no longer a trend. It is now the center of most IT strategies, to companies of all sizes in every industry. However, there are many ways of adopting Cloud computing, as well as considerations to be made when deciding on the best approach, such as:

- Should I use Public or Private Cloud?
- How do I move the application or workload to the Cloud? Lift-and-shift, contain-and-extend or refactoring?
- What are the required integrations or dependencies?
- What are the non-functional requirements?

[3] https://www.ibm.com/docs/en/developer-for-zos/14.1.0?topic=documents-enterprise-cobol-zos-v62
[4] https://www.ibm.com/support/pages/enterprise-cobol-zos-documentation-library
[5] https://www.ibm.com/cloud/wazi-as-a-service

There is no single solution for all business problems and it is typically defined at the application level, depending on its unique requirements.

This is where a Hybrid Cloud approach excels. It provides a standard and consistent experience for developers to build and test applications across the various platforms of the enterprise, while giving flexibility for decision makers to choose where to deploy and run production workloads based on business needs. In this context, IBM Z is typically the best fit platform for mission-critical applications, that demand high availability, scalability and high levels of security, while maintaining low latency even when processing large volumes of transactions and data.

The aspects and characteristics of different Cloud approaches are discussed in the *Why IBM Hybrid Cloud for Your Journey to the Cloud?*, REDP-5653 IBM Redbooks publication, as well as in the "*The Cloud Adoption Playbook*" e-book, available on the following website:

https://www.ibm.com/cloud/architecture/adoption/the-cloud-adoption-playbook/

1.3 Application modernization

For applications to be part of the Hybrid Cloud enterprises leverage a set of capabilities, that are now becoming an industry standard:

- Agility through automated DevOps practices
- Simple integration by using RESTful APIs
- Ability to shift workloads around different enterprise platforms, including on-premises infrastructure and various Cloud providers.

These capabilities can help driving better business performance and can be applied to most applications that run in the IBM Z platform:

- For new or recently deployed workloads, they are easily adopted as part of the development cycle using modern tools and practices.
- For legacy workloads, typically written many years ago, this may require applications to be modernized using certain patterns, that will be described next.

1.4 What are the modernization patterns

Application modernization involves cost, risk, change impact, and multiple variables that need to be considered when deciding on the best approach. Patterns are designed and published to provide guidance and best practices to address specific business challenges. It is also important to mention that modernization is a continuous journey, with incremental progress. Technology is not static and will keep evolving, therefore these patterns need to be constantly adjusted and updated to remain current.

IBM's published modernization patterns are divided into four categories:

1. *Application centric:* Drive enhanced function and flexibility through new application components that are developed as cloud-native functions, which either extend or enhance application components. New application components can either invoke applications on IBM Z or the applications can also invoke these new cloud-native functions.

2. *Data-integration centric:* Gain data-driven business value as applications share system-of-record (SOR) data either through direct access, replication, caching, or data virtualization concepts that combine data assets across the enterprise.
3. *Event driven:* Leverage dynamic interaction between loosely coupled applications, applications, and new cloud-native applications, exchanging information through events to improve performance and experience.
4. *Lifecycle enablers:* Derive deeper insights about the environment and enable agile solution development through techniques and tools that support DevOps practices and software pipelines.

In the next section these patterns are applied into the IBM Z context.

1.5 How to apply modernization patterns on IBM Z

As previously mentioned, there is no single solution for all applications or business problems. As such, the best modernization pattern will depend on the desired outcome for each application. Understanding application and data interdependencies, affinities, performance, security and availability goals are a prerequisite to identifying the best modernization strategy. It is also important to mention that multiple modernization patterns may apply to a single application, depending on its requirements.

The modernization patterns for IBM Z provide guidance and are described next in this section.

1.5.1 Application centric

The Application centric approach focuses on enhancing application functions, by applying one or more of the following patterns:

Expose applications through APIs

Access mainframe applications and data by using standards-based REST APIs with IBM z/OS Connect EE. Manage APIs by using industry-standard API management solutions, including solutions by IBM.
https://www.ibm.com/cloud/architecture/architectures/z-expose-apis-pattern

Extend existing code applications on z/OS with Cloud-native applications

Augment core applications on IBM z/OS with new cloud-native components that are integrated through REST APIs. The cloud-native application can be pre-existing or developed by using enterprise DevOps and containers that run on IBM z/OS or IBM LinuxONE.
https://www.ibm.com/cloud/architecture/architectures/z-extend-cloud-native-pattern

Co-locate applications with existing IBM Z applications and data

Collocate applications on IBM Z in a container that can access existing data or applications with order of magnitude reduced latency to meet SLA objectives.
https://www.ibm.com/cloud/architecture/architectures/z-collocate-applications-pattern

Enhance selected functions by incrementally rewriting as Cloud-native

Incrementally rewrite a part of a mainframe application driven by an immediate business need. Use cloud-native principles, enterprise DevOps, modern languages, and container technology on IBM z/OS or IBM LinuxONE. Integrate new functions by using APIs to or from assets by using the co-existence model.
https://www.ibm.com/cloud/architecture/architectures/z-enhance-cloud-native-pattern

Refactor elements of an existing IBM Z application into discrete services

Refactor functions into reusable components for agile development and sharing by applications.
https://www.ibm.com/cloud/architecture/architectures/z-refactor-discrete-services-pattern

1.5.2 Data Centric Integration

The Data Centric Integration approach focuses on eliminating the existence of data silos within organizations. In other words, this means allowing data that reside on IBM Z to be accessible by applications regardless of their platforms, respecting data security practices. To do so, one or more of the following patterns can be used:

Enable modern access to existing IBM Z data

Provide modern support for accessing IBM Z data through SQL-based query and through REST APIs. Simplify new application development by using this modern data access without disrupting data management and recovery processes on IBM Z to maintain data consistency.
https://www.ibm.com/cloud/architecture/architectures/z-enable-modern-access-pattern

Virtualize IBM Z data to provide in-place access across data sources

Access data across IBM Z and other data sources, including joining data, without the need to copy and replicate that data. Deliver more current and accurate data with in-place data access to consuming applications, including analytics.
https://www.ibm.com/cloud/architecture/architectures/z-virtualize-ibm-z-data-pattern

Cache SOR data residing on IBM Z for creation of new data

Improve application response time and scalability by storing optimized copies of IBM Z data. Free compute resources and increase throughput by offloading application logic to the caching layer, especially for read-heavy applications.
https://www.ibm.com/cloud/architecture/architectures/z-cache-ibm-z-data-pattern

Transform SOR data residing on IBM Z for creation of new data

Incrementally build new modernized systems-of-record (SOR) data stores by tapping into IBM Z data traffic through a data adapter to transform to the modernized data format.
https://www.ibm.com/cloud/architecture/architectures/z-transform-data-pattern

Replicate SOR data residing on IBM Z using change data capture

Replicate data in real time by capturing change log activity to drive changes in the target. Enable newer applications to access a broader range of data stores and access methods through replication.
https://www.ibm.com/cloud/architecture/architectures/z-replicate-data-pattern

1.5.3 Event driven

The Event driven approach allows IBM Z applications to interact in near real-time with others, regardless of their platform. The following patterns can be applied:

Respond in near real-time to events occurring within IBM Z applications:

Share events that are generated in IBM Z applications so that new application logic can be developed to respond to such events without introducing risks in core applications. Analyze data as part of application logic to generate an event.
https://www.ibm.com/cloud/architecture/architectures/z-respond-to-z-app-events-pattern

Respond to external events in near Real-time leveraging IBM Z assets:

Share events that are generated by applications that are external to IBM Z to drive the invocation of IBM Z application logic. Develop flexible logic without introducing risks in core applications.
https://www.ibm.com/cloud/architecture/architectures/z-respond-external-events-pattern

Optimize CQRS - delivering Core systems Integration:

Deliver an efficient CQRS system that is based on IBM Z to optimize the synchronization between the command access, which is SOR data that is updated by online and batch applications, and the query access, which is an information model that is aligned with the needs of the new applications. Optimize by using IBM Z Digital Integration Hub to deliver a non-disruptive, low-latency, high-throughput, and cost-attractive solution.
https://www.ibm.com/cloud/architecture/architectures/z-optimize-cqrs-pattern

1.5.4 Lifecycle Enablers

The Lifecycle Enablers approach focuses on enabling IBM Z applications to take advantage of common agile practices and tools that are used by modern development frameworks. As such, the following patterns can be applied:

Application discovery for Business Alignment

Extract consumable information about your software assets and build an inventory of applications and their resource usage and dependencies. Visualize information flow across application components, perform impact analysis, and generate reports to act on your modernization strategy and plan increments with confidence.

https://www.ibm.com/cloud/architecture/architectures/z-application-discovery-pattern

Implement Enterprise DevOps including IBM Z application environment (Agile/CI/CD)

Provide a cloud-native developer experience for IBM Z by fully integrating IBM Z development into enterprise continuous integration, continuous delivery (CICD) pipelines and embracing consistent open-source tools that are familiar to all developers.

https://www.ibm.com/cloud/architecture/architectures/z-enterprise-devops-pattern

1.6 Overview of network architecture

In this section we provide an overview of a hybrid cloud architecture. While discussing the network architecture is outside of the scope of this IBM Redbooks publication, in our environment, we worked with the following three main topologies:

- An application running in a non-z/OS environment accessing database or files residing on z/OS
- Application running on z/OS, typically leveraging IBM CICS®, IMS, MQ, WAS or equivalent middleware subsystems, accessing or storing data in an external data repository, such as MongoDB, Postgres or equivalent databases.
- Both application and data repository running on OpenShift, either on zCX or Linux on Z.

1.7 Security

Although security is an important topic it is not covered in this book at the time of this writing. However, the IBM Z platform brings with it several built-in features that can be explored by workloads running in the platform:

- *Encryption everywhere:* Stop choosing what to encrypt. Encrypt faster and without app changes.
- *Quantum-safe protection:* Act now to future proof your business. Start protecting your data, apps and infrastructure from future quantum threats.
- *Plan for crypto agility:* Discover where and what crypto is used in applications to build and maintain your crypto inventory.
- *Preserve privacy with zero trust:* Protect and control access to sensitive data as it is shared throughout your hybrid cloud.
- *Centralize keys:* Manage keys efficiently and securely for IBM z/OS data set encryption on IBM Z and public cloud key management systems.
- *Protect data in flight:* Protect and encrypt data flowing on IBM FICON® and Fibre Channel links from IBM Z to DS8900F, or between Z platforms.

For more information please visit the *IBM Z Enterprise Security* portal at the following website:

https://www.ibm.com/it-infrastructure/z/capabilities/enterprise-security

2

Modernized application architectures

IT environments are now fundamentally hybrid and multi-cloud in nature.

Companies adopting cloud applications view application modernization as a key component to harmonize business processes across their hybrid cloud applications.

As part of the next critical step in their digital transformations, organizations are building new applications and modernizing legacy applications to leverage cloud native technologies, which enable consistent and reliable development, deployment, management and performance across cloud environments and across cloud vendors.

In this chapter, we'll explore how to accelerate application modernization using the following architectural patterns designed and published by IBM, you can use them to learn how to implement and deploy them in an IBM Z environment, whether z/OS or Linux, and determine which circumstances are best for your application modernization initiatives. For more details, please refer to https://www.ibm.com/cloud/architecture/architectures/application-modernization-mainframe/

Application centric: [1]

- Expose through APIs

  Expose applications and data through APIs.

  Access mainframe applications and data by using standards-based REST APIs with IBM z/OS Connect EE. Manage APIs by using industry-standard API management solutions, including solutions by IBM.

- Extend with cloud native

  Extend core applications on IBM z/OS with cloud-native applications.

[1] The Application centric patterns are part of IBM Z application modernization patterns which are indtroduced and discussed on https://www.ibm.com/cloud/architecture/architectures/application-modernization-mainframe/patterns. Contributors are: Asit Dan, Yves Tolod, Elton DeSouza, Catherine Moxey, William Alexander

Augment core applications on IBM z/OS with new cloud-native components that are integrated through REST APIs. The cloud-native application can be pre-existing or developed by using enterprise DevOps and containers that run on IBM z/OS or IBM LinuxONE.

- Colocate applications

  Colocate applications with existing IBM Z applications and data.

  Colocate applications on IBM Z in a container that can access existing data or applications with order of magnitude reduced latency to meet SLA objectives.

- Enhance as cloud native

  Enhance selected functions by incrementally rewriting as cloud native.

  Incrementally rewrite a part of a mainframe application driven by an immediate business need. Use cloud-native principles, enterprise DevOps, modern languages, and container technology on IBM z/OS or IBM LinuxONE. Integrate new functions by using APIs to or from assets by using the co-existence model.

- Refactor into discrete services

  Refactor elements of an IBM Z application into discrete services.

  Refactor functions into reuseable components for agile development and sharing by applications.

In following sections, we'll talk about more on the first two patterns: 2.1, “Expose through APIs pattern” and 2.2, “Extend with cloud native pattern”, and describe a sample application architecture and demostrate how to deploy it onto IBM Cloud® and IBM Z environment in 2.3, “Sample application architecture”.

2.1 Expose through APIs pattern

New initiatives often impose significant demands on applications and data because timely access to data drives new business processes and better customer experiences.

Many Organizations continue to rely on core applications and data on IBM Z. To accelerate digital transformation, organizations need to embark on a strategy to modernize core applications and build new ones.

However, challenges often arise when it comes to updating mainframe-based monolithic applications to support new business initiatives. Risks are involved, and the effort required to develop the new features and test the application is often significant. Alternatively, a full rebuild also faces multiple challenges, such as high development costs, lack of documentation and understanding of the business logic, exposure risks to business-critical data, and poor performance and availability.

A good starting point for modernization is to leverage core business-critical applications on IBM Z by using APIs that are consumed by new cloud-native application logic, such as mobile or cognitive applications. You need to develop APIs to expose your applications on IBM Z. You also need a robust and comprehensive runtime environment for APIs that is scalable, highly available, secure, and that covers all subsystems.

Solution and Architecture

The following Figure 2-1 on page 14 shows the components that are involved in invoking a mainframe application that is exposed as APIs. In the simplified flow example, a cloud-native application invokes an API that is managed, secured, and exposed by an enterprise API management system that uses an API gateway. The API gateway, upon receiving a request, checks to see whether it is an authorized request. If the request is authorized, the gateway routes the request to a corresponding API that is deployed on z/OS Connect EE that runs on IBM Z. The z/OS Connect EE server transforms a REST/JSON based API request into a payload according to the prespecified copybook format. It also invokes a z/OS application that runs on a subsystem such as CICS, IMS, or IBM Db2®. Similarly, the z/OS Connect EE server transforms the response from the application into the results format that the API definition specifies.

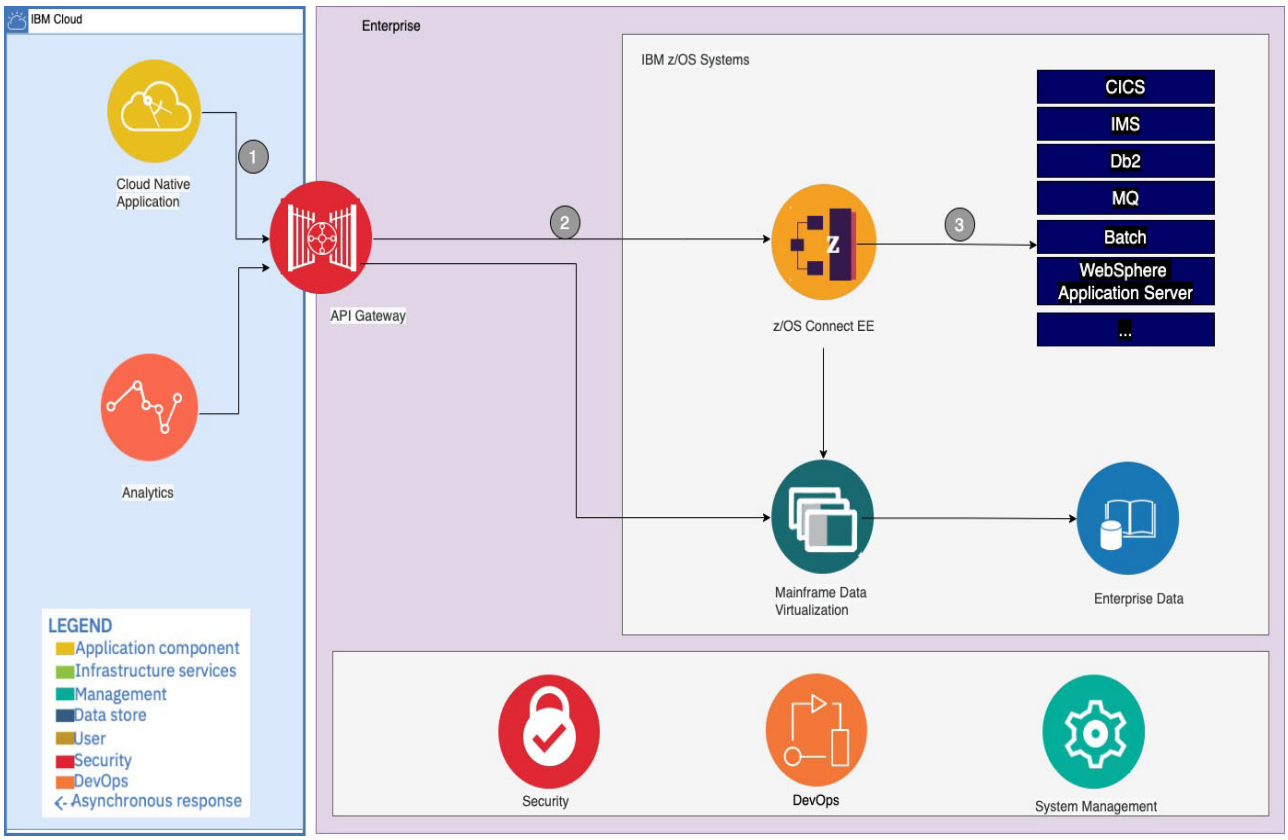


Figure 2-1 Expose through APIs on z

A key business benefit is that no new coding or changes to mainframe assets are required for newly developed cloud-native applications to access core business-critical applications and associated data.

Considerations

- Use IBM z/OS Connect API toolkit.

  The ability to consume an API from a z/OS application starts with the use of an API description format to document and understand the interface to the cloud-native application. Such a format might be based on the OpenAPI specification. From the OpenAPI document, you can use the IBM z/OS Connect API toolkit to build the artifacts to enable a z/OS application that is written in COBOL or PL/I to call the API. The following artifacts that are generated:

  - An API runtime file (API requester archive) that contains the transformation logic to convert the request payload from binary format to JSON format and convert the response payload from JSON format to binary format
  - An API information file that contains information such as the path and method of operation that is supported by the API
  - The request and response data structures that are used for each operation in the API

  Note: More information on IBM z/OS Connect can be found at the following website: `https://www.ibm.com/docs/en/zos-connect/zos-connect/3.0?topic=welcome-overview`

- Use IBM Application Discovery and Delivery Intelligence.

To discover and understand the code to be exposed as APIs, use this tool to analyze assets on IBM Z. You can understand the impact and interdependencies of extending a core application. For more information, please refer to https://www.ibm.com/cloud/architecture/architectures/z-application-discovery-pattern

- Enable monitoring and management.

  Consider integrating the monitoring of APIs with the monitoring of the end-to-end solution, spanning the cloud-based application that invokes APIs and the API enabling runtimes and core applications that run on IBM Z. Please refer to Chapter 6, “Managing your applications” on page 109 for more details.

- Consider your deployment topology.

  Consider supporting the full lifecycle of applications, APIs, including development and testing, deployments, and a highly available and scalable production runtime environment. Please refer to Chapter 7, “Deployments of production applications” on page 125 for more details.

2.2 Extend with cloud native pattern

As you drive digital transformation and enable hybrid cloud solutions, you face these challenges:

- Using core assets that can't be re-created without a huge upfront cost, a significant effort, and a long delay
- Avoiding the risks of rebuilding business-critical applications without thorough due diligence, as doing so can lead to high development costs, lack of documentation and understanding of business logic, exposure of business-critical data, and poor performance and availability
- Enabling agility and innovation by supporting a new cloud-based application ecosystem through enabling multi-speed IT with cloud-native applications
- Using open standards-based languages and tools to enhance applications with new microservices and container technology

Solution and architecture

The "Extend with cloud-native" pattern as shown in Figure 2-2 on page 16 shows the process and components that are involved to extend a core application on z/OS, whether CICS, IMS, or batch, by writing new functions as cloud-native applications. Communication between the core application on z/OS and the cloud-native application takes place by using a well-defined API interfaces.

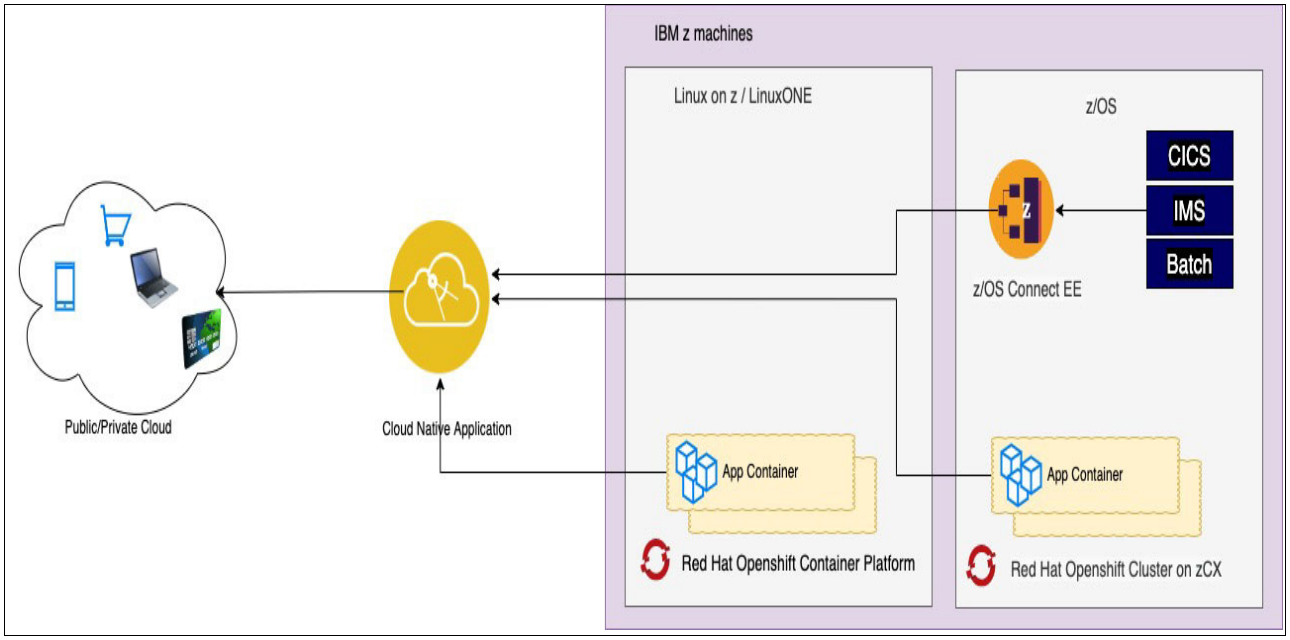


Figure 2-2 Extend core applications on IBM Z with cloud-native

A key business benefit of using this pattern is that enterprises don't need to abandon their investment in core applications on IBM Z. Instead, they can breathe new life into those applications by extending their capabilities with cloud-native applications. The use of cloud-native technologies can enable organizations to build highly scalable applications in a modern environment with private, public, and hybrid clouds.

The ability to use cloud-native applications with core applications on z/OS provides many advantages:

- Enables speed to develop and deploy new capabilities, responding to market demands by using modern development tools and processes
- Addresses skills shortages of older technology by using open standards-based languages
- Takes advantage of security frameworks that are available with modern applications to meet compliance standards
- Avoids the high risks of completely rebuilding business-critical applications
- Provides a way to deploy updates without redeploying the entire application

Considerations

- The use of containers with cloud-native applications offers flexibility of deployment for a hybrid cloud solution:
  - Deploy containers inside z/OS and closer to the z/OS application using the z/OS Container Extensions Technology.
  - Deploy containers on Red Hat OpenShift Container Platform on premises on IBM Z or a distributed platform.
  - Deploy containers on Red Hat OpenShift Container Platform on public cloud.
  - Deploy containers on any Kubernetes platform.
- The key decision criteria include reducing latency with the colocation of cloud-native applications closer to z/OS based applications and data as well as meeting stringent SLAs on security, availability, and scalability. Please refer to the "Colocate applications" pattern for additional details.
  https://www.ibm.com/cloud/architecture/architectures/z-collocate-applications-pattern

2.3 Sample application architecture

In our sample application architecture, we are deploying an open-source, lightweight and microservices based application called the “Voting app”. It is cross platform and can be deployed on any architecture. We are using this application as an example to demonstrate how to leverage the application modernization pattern to learn how to implement and deploy applications in a hybrid cloud platform, which combines the public cloud, Red Hat OpenShift on IBM LinuxONE and Red Hat OpenShift on zCX.

You can find the public repository with the application source code at the following website: https://github.com/OpenShift-Z/voting-app

In original source code, some programs are using a Postgres database, we made some changes to these programs to support IBM Db2, please refer to Appendix A, “Voting App changes needed to support IBM Db2 database” on page 139 for more details.

2.3.1 Application architecture

Our application architecture is shown in Figure 2-3

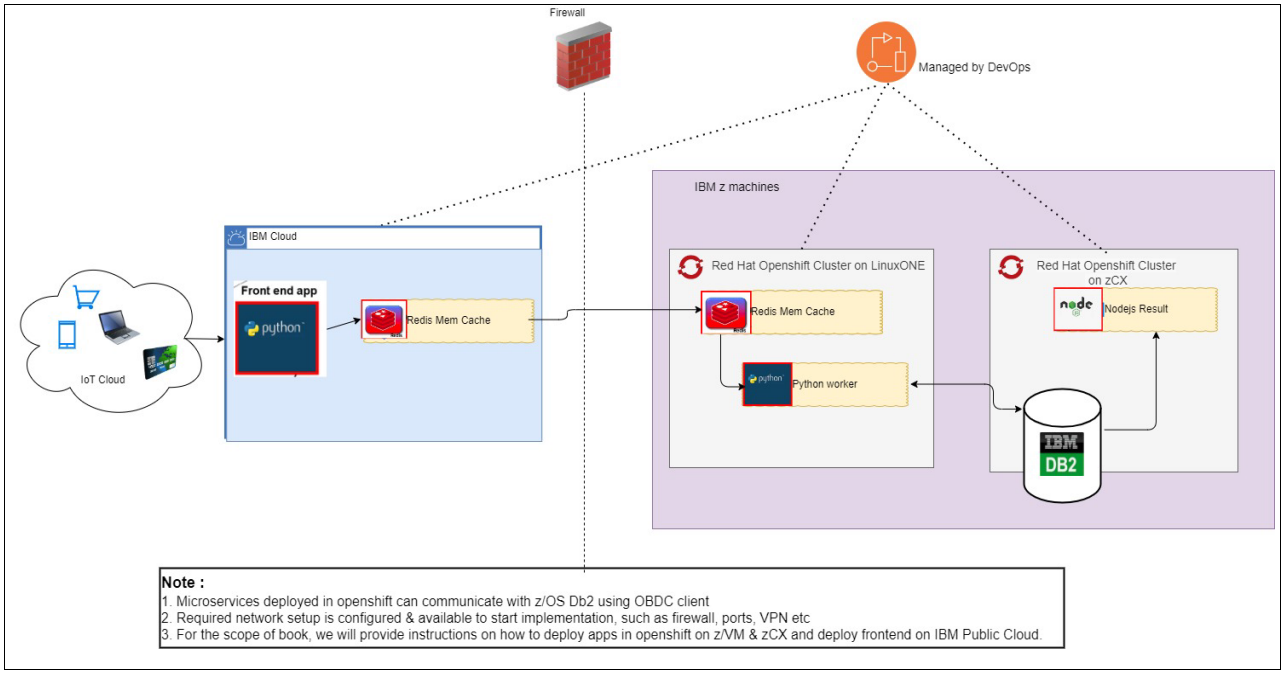


Figure 2-3 Sample Voting applicaton architecture

The application provides the user with a choice to vote for any of two given options (such as Coca-Cola vs Pepsi).

On IBM Public Cloud:

The web front end is comprised of a Python microservice that displays the options for users to choose from. When the user interacts with the application, the information (votes) is sent to the Redis microservice

On Red Hat OpenShift Cluster on LinuxONE:

Redis serves as an in-memory cache holding the votes received by the Python web microservice.

Running in a background there is another microservice, also written in Python, that takes the votes from Redis and stores them in a IBM z/OS Db2 database.

On IBM z/OS:

Running Db2 database, where the votes are stored at.

On Red Hat OpenShift Cluster on zCX:

Finally, there's a Node.js microservice, that shows the voting results as they are accumulated in the Db2 database.

The application follows the notation of microservices-based architecture and its components can be scaled individually; in fact each component can be switched with an alternative one, without affecting the overall application.

2.3.2 Deployment

Environment:

- Red Hat OepnShift cluster on IBM Public Cloud
- Red Hat OpenShift cluster on IBM LinuxONE
- IBM zCX Foundation for Red Hat OpenShift (zCX for OpenShift)
- Db2 on z/OS

To deploy the application, you will need to get the GitHub personal access token that will be used for generating a secret. More detailed information can be found at the following website: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-apersonal-access-token

Procedures:

We performed the following steps to start the deployment:

1. Deploy z/OS Db2 database

```
DB2 SSID            D2B1 , D2B2
DB2 member name     D2B1 , D2B2
IRLM SSID           I2B1 , I2B2
z/OS System         SC74 , SC75
Command Prefix      -d2b1 , -d2b2
DRDA Port           38010
Security Port       38011
Resync Port         38012, 38013
Location            DB2B
```

On All Red Hat OpenShift clusters:

2. Set the following environmental variables: $PROJECT, $GIT_REPO, $GIT_TOKEN
3. Create the new project by using the following command.

```
oc new-project ${PROJECT}
```

4. Create the secret by using the following command.

```
oc create secret generic git-token --from-file=password=${GIT_TOKEN}
--type=kubernetes.io/basic-auth
```

5. Import images with the following commands.

```
oc import-image rhel8/nodejs-12 --from=registry.redhat.io/rhel8/nodejs-12
--confirm
oc import-image ubi8/python-38 --from=registry.redhat.io/ubi8/python-38
--confirm
```

On Red Hat OpenShift cluster on LinuxONE:

6. Deploy the Redis service with the following commands:

```
oc new-app --name new-redis --template=redis-persistent \
--param=DATABASE_SERVICE_NAME=new-redis \
--param=REDIS_PASSWORD=admin \
--param=REDIS_VERSION=latest
```

Example 2-1 Sample output for Redis Pod

Name	Status	Ready	Restarts	Owner	Memory
P new-redis-1-pzkvv	Running	1/1	0	RC new-redis-1	12.8 MiB

7. Deploy the Python worker by using the following commands:

```
oc new-app python-38:latest~${GIT_REPO} \
--source-secret=git-token \
--context-dir=worker-python \
--name=voting-app-worker-py \
-e DB_NAME="DB2B" \
-e DB_USER="admin" \
-e DB_PASS="admin" \
-e HOST_NAME="wtsc75.pbm.ihost.com" \
-e PORT_NO="38010" \
-e REDIS_PASSWORD="admin"
```

Example 2-2 Sample output for Python woker Pod

Name	Status	Ready	Restarts	Owner	Memory
P voting-app-worker-py-6d8c95dd57-cbbbp	Running	1/1	0	RS voting-app-worker-py-6d8c95dd57	20.6 MiB

On Red hat OpenShift cluster on IBM Public Cloud:

8. Deploy the voting app Front-end with the following commands:

```
oc new-app python-38~${GIT_REPO} \
--source-secret=git-token \
--context-dir=/vote \
--name=voting-app-py \
-e REDIS_PASSWORD="admin"
```

Example 2-3 Sample output for Voting app frontend Pod

Name	Status	Ready	Restarts	Owner	Memory
voting-app-py-7b59f4d4d6-jwt5m	Running	1/1	0	voting-app-py-7b59f4d4d6	46.0 MiB

9. Create the voting app Front-end route by using the following commands:

```
oc create route edge demo-py --service=voting-app-py --port=8080
```

Example 2-4 Sample output for Voting app frontend route

Name	Status	Location	Service
demo-py	Accepted	https://demo-py-voting-app-db2.apps.rdbkvmocp.pbm.ihost.com	voting-app-py

On Red Hat OpenShift cluster on zCX:

10. Deploy the Nodejs result front-end application by using the following commands:

```
oc new-app nodejs-12:latest~${GIT_REPO} \
--source-secret=git-token \
--context-dir=result \
--name=voting-app-nodejs \
-e
DB2_CONNECT_STRING="database=DB2B;hostname=wtsc75.pbm.ihost.com;port=38010;prot
ocol=tcpip;uid=admin;pwd=admin"
```

Example 2-5 Sample output for nodejs result frontend Pod

Name	Status	Ready	Restarts	Owner	Memory
voting-app-nodejs-7b75f85db4-fszfj	Running	1/1	0	voting-app-nodejs-7b75f85db4	40.4 MiB

11. Create the Nodejs result front-end route by using the following commands:

```
oc create route edge demo-nodejs --service=voting-app-nodejs --port=8080
```

Example 2-6 Sample output for nodejs result frontend Route

Name	Status	Location	Service
voting-app-nodejs	Accepted	https://voting-app-nodejs-voting-app-db2.apps.ocpzcx1.rdbkocp.pbm.ihost.com	voting-app-nodejs

Access applicaiton via Web Browser:

Voting App Frontend: https://demo-py-voting-app-db2.apps.rdbkvmocp.pbm.ihost.com

Voting App Result: https://voting-app-nodejs-voting-app-db2.apps.ocpzcx1.rdbkocp.pbm.ihost.com

2.3.3 Consideration

- Latency is always an important aspect to address in any cloud-native applcation.
  - Platform matters.

    Hybrid cloud with IBM Z technologies, allowing the application to scale in three dimensions (horizontally, vertically or both), achieving superiror performance and SLAs (Service Level Agreements) to a similar deployment in a hybrid cloud environment.
  - Colocation of cloud-native applications and data within or closer to IBM Z

    By colocating supporting applications on the same IBM Z platform as core systems-of-record (SOR) applications, you can leverage the data gravity and reduce latency, as well as better security, availability, and scalability. Please refer to the "Colocate applications" pattern for additional details.
    https://www.ibm.com/cloud/architecture/architectures/z-collocate-applications-pattern
  - Introduce memory cache

    Caching is one of the key implementation when it comes to production level deployment of services, which help to increase the application performance by acting as a middle layer between application component and the persistence sytem. In this sample application, we are using redis memory cache as an example.
- Enterprise DevOps.

  While integrating IBM Z into your hybrid cloud, it is imperative that developers and IT operations understand that the same, agile processes can also be performed on IBM Z, by using the same DevOps tools and having the same DevOps experience as other platforms. A range of solutions helps integrate systems, empowering developers with an open and familiar development environment with enterprise-wide, platform agnostic standardization, in turn helping developers build, test, and deploy code faster. Please refer to Chapter 5, “Modernize enterprise DevOps” on page 61 for more details.

3

Modernized data access architectures

In this chapter we describe how to modernize the way data accesses are done today. The following topics are covered:

- Data fabric as the basis for modern data access
- Data-centric integration patterns[1]
  - Enable modern access to IBM Z data pattern
    Application development by using modern data access to maintain data consistency.
  - Virtualize IBM Z data
    Virtualize IBM Z data to provide in-place data access across data sources
  - Cache IBM Z data
    Cache data on IBM Z for higher performance
  - Transform IBM Z data
    Transform SOR data on IBM Z for the creation of data

[1] The data modernize patterns are part of the IBM Z application modernization patterns which are introduced and discussed on https://www.ibm.com/cloud/architecture/architectures/application-modernization-mainframe/patterns. Contributors are: Maryela Weihrauch, Sueli Almeida, Daniel Martin, Paul Cadarette, Greg Vance

3.1 Data fabric as the basis for modern data access

Thanks to the recent acceleration of digital transformation, every business is facing significant change. In this transformation process, only those companies that treat the data they accumulate as a strategic asset will gain a competitive advantage.

Most of today's successful businesses already understand this and many companies do indeed view data as a strategic asset. Unfortunately, this does not yet mean that they are taking steps to extract more actionable information from their data. They are not planning any data strategy and are not dealing with the additional costs of poor data quality.

At the heart of a typical IT solution is the system that supports the core business. This is complemented by other sub-systems, sometimes created as an afterthought, specialized in specific subtasks, exploiting new technologies, cloud computing, mobile and social technologies, security, and, finally, the unstructured data generated by IT systems that pervade all aspects of life (called Big Data) and the analysis of the totality of these.

Thanks to traditional processing, there are many data sources everywhere, and each data source can have hundreds of tables, each with dozens of columns. This data must be used to serve a relatively large number of users or use cases, where the end-users usually require slightly different data. The amount of data is now so vast that centralizing it is an impossible task. Data is stored in many places and virtually everyone will use it everywhere.

One of the most serious challenges to maximizing the use of data is that it is constantly changing. Every company is using increasingly complex and diverse data structures and data types that are in a state of almost constant change to meet the ever-changing business needs.

As a first - and most obvious - step, most companies are already distinguishing between structured and unstructured data and are really starting to think about how to link them together. But there is still the challenge of data diversity.

A typical example of this is when you need to run reports and analytics on mixed-source data. Combining different data sources is not easy, but processing is also complicated by inconsistent data from different sources. It is easy to see that the more data sources a company has, the more likely it is that data quality will eventually become a problem.

In addition, recent events have brought a new challenge, namely the significant organizational transformation of companies. In the wake of the COVID-19 pandemic, masses of office workers have started to work remotely and many may continue to do so in the future. We are also seeing more and more mobile workers and even more who work without a permanent office. Of course, their data needs must also be met. Most of these employees with changing work styles generally want to use data in a self-service way.

A new approach called **data fabric** can provide answers to these challenges.

What exactly is a data fabric, how is it different from previous architectures, and what are the benefits for companies?

3.1.1 What is a data fabric?

A data fabric is not a single product or a single platform. It cannot be purchased or installed. A data fabric is an architecture that can incorporate all your existing data sources and facilitate the integration of these data sources using automated solutions.

The data fabric is a modern, distributed data architecture that includes distributed data tools and optimized data management and integration processes to manage today's data challenges in a unified way.

The hope is that the data fabric will allow users to spend more time analyzing data than fiddling with it. Data consumers will have access to integrated, high quality and usable data.Figure 3-1is a high level representation of data fabric integration.

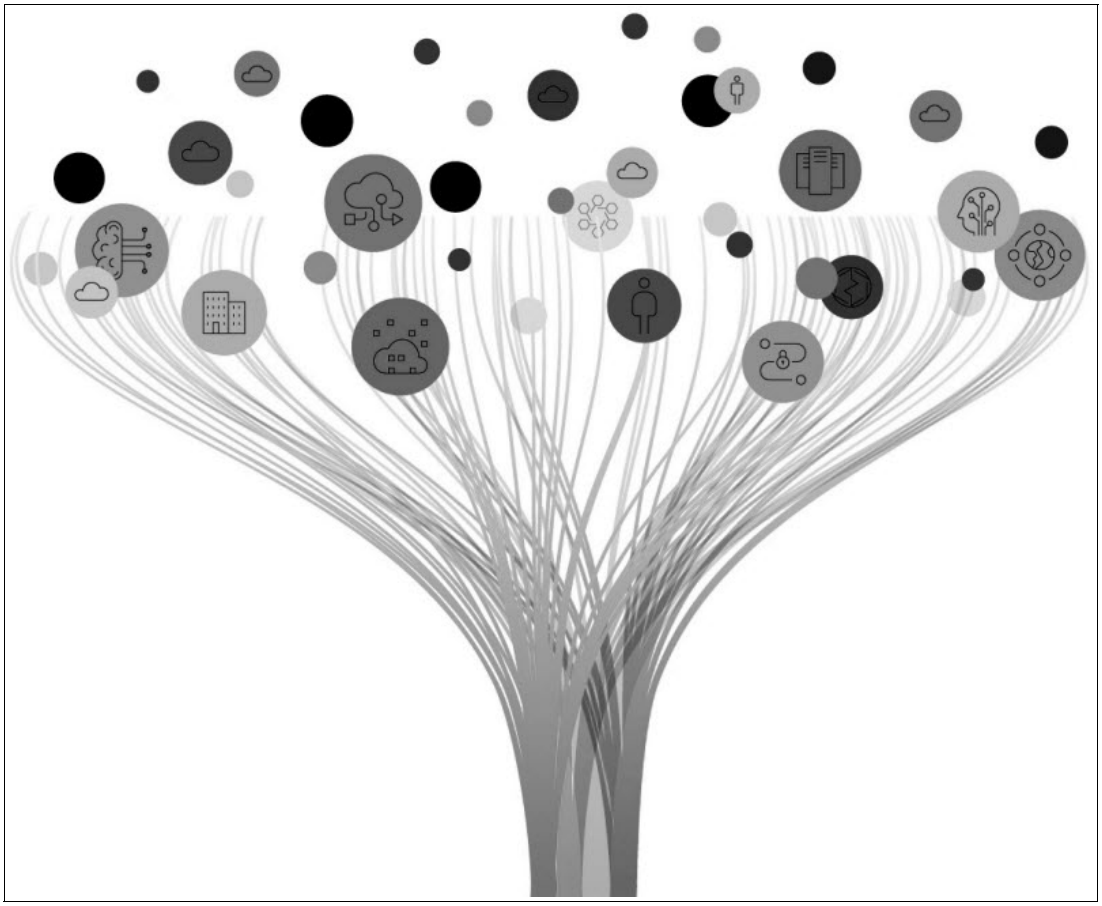


Figure 3-1 Data fabric

The data fabric architecture is needed to make it easier to find data in a reliable and accessible way for anyone. The data fabric can enable decision-makers to look at data from different sources in a unified way to better understand customer problems and make connections between data that did not exist before. By closing the gaps in understanding customers, products, and processes, data fabrics accelerate digital transformation and automation initiatives in enterprises.

3.1.2 Data fabric architecture

As you can guess from the above, the data fabric can bring together data from legacy systems, data lakes, data warehouses, relational and non-SQL databases, and applications. The aim is to achieve greater integration between data environments as opposed to individual data warehousing systems. At the same time, it tries to avoid the problem that data becomes more difficult and costly to move and transform as it grows (commonly known as the problem of data gravity). The data fabric tries to make all data available across the enterprise

wherever possible.

Let's be clear: there is not much that is new in the components that make up the data fabric. We continue to use the existing elements that are constantly evolving. Continuous evolution is particularly true in those application areas where the cloud is involved. It is a new combination of these existing and changing elements that creates this new approach we call the data fabric.

An example of a data fabric architecture is in a multi-cloud environment, for example, accounting is done in one cloud, a CRM system is run in another cloud, and data cleansing and transformation are done on another platform. Furthermore, on another platform, for example, IBM Cloud Pak® for Data, analytics services may be used.

The data fabric architecture can integrate these environments to give decision makers a single view and see relationships between data that were previously unknown.

But this is just a general example. Different businesses have different needs so there is certainly no one-size-fits-all approach.

At the same time, different data fabric architectures have common elements:

- **Data organization**

  This is where some of the most important tasks of the data fabric take place: data transformation, integration, and cleansing. It makes data usable by different units across the enterprise. It enables the categorization and access of enterprise data from multiple data sources while implementing strong access management. It can include centralization, control, and management of master data management or data describing other data (metadata).

- **Data management, data access**

  Enforce data policies and maintain data quality. Helping users establish policies, processes, and accountability and ensuring that data quality remains satisfactory. Enabling data use, ensuring that users in each group or department have the right to access the data they need, and controlling that everyone has access only to the data that is relevant to them.

- **Data preparation and data quality**

  Analyze information and identify incorrect, incomplete, or improperly formatted data. Data quality tools clean or correct these data according to defined rules.

- **Data integration, data processing**

  Takes data from different sources and combines it into a single view. This layer refines the data so that only the relevant data is extracted. Integration starts with data entry and includes cleaning; Data integration enables users to apply analytical tools to produce actionable business intelligence.

- **Data discovery**

  Discover new opportunities by integrating different data sources. For example, you can find previously hidden opportunities to link your accounting system database with CRM system data, opening new possibilities for developing new personalized offers for your customers.

- **Data analysis, data visualization**

  This layer is where the data are examined to identify trends and draw conclusions about the information they contain. Data Visualization is a way of seeing what the data tells us. Rather than being presented in numerical, tabular, or other formats, data are presented

graphically in charts and graphs. These can make it much easier to understand trends or messages in the data.

3.1.3 Advantages of data fabric architectures

The key achievement of data fabric architectures is that they enable action at the speed of business, often in real-time. Data only brings business benefits when it is made available to any user in the organization. When properly implemented, the data fabric also helps to reveal hidden relationships between data, enabling the value of data to be accessed in the most efficient and automated way across the organization.

- **Intelligent Integration**

  The implementation uses metadata, machine learning, and artificial intelligence to unify data of different data types and endpoints. This helps data management teams to group related data sets, but also helps to eliminate data silos and improve data quality. This activity involves capturing, consolidating, and making data available through different systems so that it can be analyzed where it is needed.

  Intelligent Integration helps to automate any data access or data delivery process, without the need for a tedious or error-prone coding process.

  Data integration optimized with automation speeds up data delivery. An automated process of real-time data capture ensures continuous data quality. Machine learning can automate specific data discovery and classification processes, resulting in faster time to value. Continuous analysis can be performed automatically in real-time, wherever the data lives.

- **Democratizing data, self-service data retention**

  Data fabric architectures facilitate the use of self-service applications, extending the range of data consumers beyond data engineers, developers, and data analysis teams. The data fabric gets data quickly into the hands of those who need it. It helps business users use the data themselves - enabling them to make faster business decisions - and frees up their experts to focus on solving problems that better leverage their skills.

  Business users find and consume data through a unified access point. Self-service data access can help businesses collaborate with other users.

- **Better privacy and security; using active metadata**

  Widening access to data should in no way mean compromising data security and privacy. In fact, it means introducing more data management barriers around access control, ensuring that certain data is only available to certain roles. A data fabric architecture can automatically enforce all data access policies to ensure a high level of data protection and compliance. The use of artificial intelligence and machine learning technologies can increase the level of automation. This will enable organizations to create and implement data governance policies that ensure the ethical use of data wherever it resides, orders of magnitude faster than ever before.

  Data fabric architectures enable technical and security teams to discover and encrypt data around sensitive and proprietary data, reducing the risks of data sharing and system breaches.

- **Provides a total picture of customers**

  By connecting all your company's data sources, you can get a single, reliable and comprehensive view of your customers. With centralised master and metadata management, you minimise the risk of incorrect data entry, increase accuracy and improve decision-making speed. New ways of accessing data create previously unavailable decisioning and reporting capabilities for companies to better understand their customers' needs and better position their products and services from a new, comprehensive view.

Using the data fabric

- Enables the creation of customized and reliable customer views.
- Standardize management policies and process execution with workflow automation features.
- Provinces more valuable and accurate reports on customer interactions.

- **Data fabric functionality supported by AI**

  AI can be used to track the lifecycle of the data fabric. It can enable greater transparency and automate lifecycle documentation. Artificial intelligence can also be used to determine which corporate policies should be enforced during the development and deployment lifecycle.

  Using artificial intelligence:

  - Enables automatic monitoring and, where appropriate, automatic re-learning of different models.
  - Controls rules for automated lifecycle monitoring.
  - Helps to create intelligent recommendations.

3.1.4 How IBM help to realize the data fabric

IBM Cloud Pak for Data is a product that makes the concept of a data fabric a reality.

IBM Cloud Pak for Data is a platform that simplifies and automates data collection, data organization, and analysis, and accelerates the flow of artificial intelligence throughout the enterprise.

IBM Cloud Pak for Data is able to connect data from disparate data sources and can run workloads in hybrid cloud environment. Designing, deploying, and managing AI in hybrid cloud environments enables enterprises to accelerate digital transformation and implement a data fabric architecture.

The IBM Cloud Pak for Data platform provides seamless integration across the enterprise as well as the following:

- Uses the services available in IBM Cloud Pak for Data.
- Integrates with external applications and data sources.
- Provides advanced AI-based capabilities for data management, data centralization, and data governance.

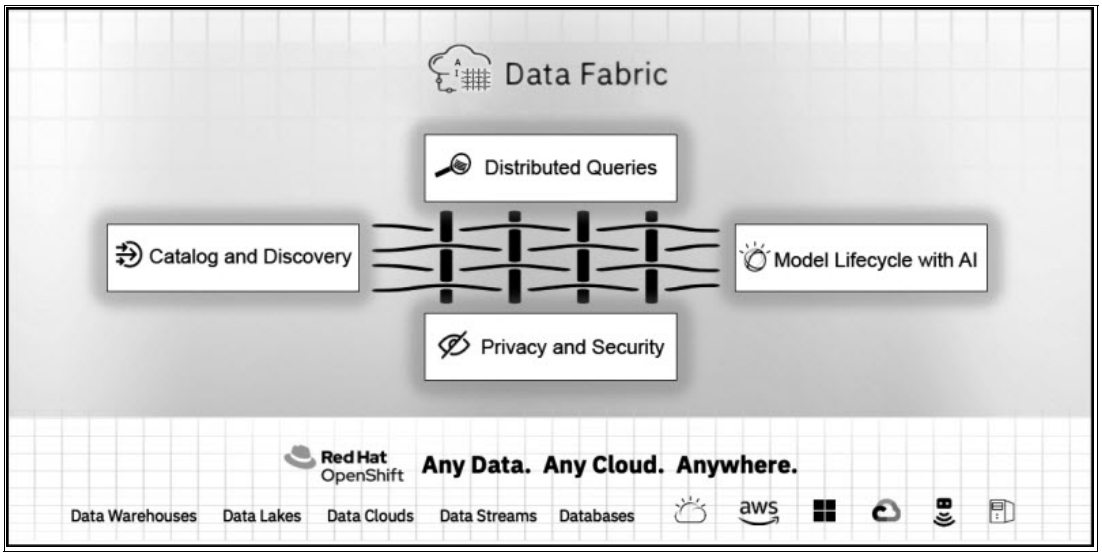


Figure 3-2 IBM Data Fabric approach

This platform provides users with the foundation to always have up-to-date, curated data with an optimal balance of performance and compliance.

It delivers data processing by intelligently tuning and managing workloads based on data location and data management policies.

In most cases, IBM Cloud Pak for Data automates the provisioning of business-appropriate data required by data fabrics. The following capabilities support the design and implementation of data fabric architectures:

- **Metadata-based knowledge core**

  Facilitates the discovery of data sources and catalogues; enriches datasets.

  Performs various analyses using AI to help automate and extract insights. The knowledge core is used to power the data marketplace through semantic searches.

- **Self-service data marketplace**

  A next-generation data catalogue that helps data consumers, such as business analysts, to retrieve data in a unified way across all data sources in the enterprise.

- **Smart integration**

  Enables data consumption by extracting and overutilizing data. Connects to the knowledge core to automate data integration and has the intelligence to decide which integration approach is best based on workloads and data management policies. It can also be used for data preparation as part of data editing workloads or to create data products. Finally, it provides the possibility to publish updates to data products.

- **Governance**

  Catalog and maintain metadata, define privacy policies, maintain data, record data provenance, and perform other tasks related to security and compliance.

  This layer is able to understand the different data formats (e.g. structured or unstructured data) and the meaning of the data (e.g. is it public or proprietary data). Apply the appropriate security policies to each piece of data and each user. Rather than having to manually apply standards and rules to data, this integrated capability means that they will be applied at the organizational level and to the appropriate data sources. Analysis

models in different tools can communicate with each other; enforcement of data policies at the elementary level can be highly automated.

- **Unified development and operations**

  Enable unified lifecycle tracking across all components of the data platform, automate configuration and run in production.

3.2 Enable modern access to IBM Z data pattern

For decades, companies have tried to copy data from different operational systems into central data stores for various business cases such as operational business transactions and analytics. Establishing and maintaining data replication pipelines is expensive, time consuming, and it creates data quality and data latency challenges for consuming applications. Accessing data in place can accelerate transformation and improve its opportunity for success. It also preserves the existing data management and recovery processes.

Accessing consistent data avoids application design complexity by eliminating the need for compensation logic. It is a powerful foundation to satisfy complex information needs, such as infuse AI models and historical data, within service processing. You can dramatically simplify application development by using broad API support through SQL, REST, or both.

3.2.1 Modern data access solution and pattern for IBM Z

IBM Z supports modern access to real-time transactional data in IBM Db2, IMS, and other data sources. Db2 and IMS can be accessed through SQL and REST API by using IBM z/OS Connect EE. Other data sources can be accessed through SQL or REST API by using IBM Data Virtualization Manager for z/OS along with z/OS Connect EE.

With Java Database Connectivity (JDBC) support, new cloud applications that consume information from core Db2 and IMS systems can use SQL to detect the underlying data format and contexts from systems of record, which is often required.

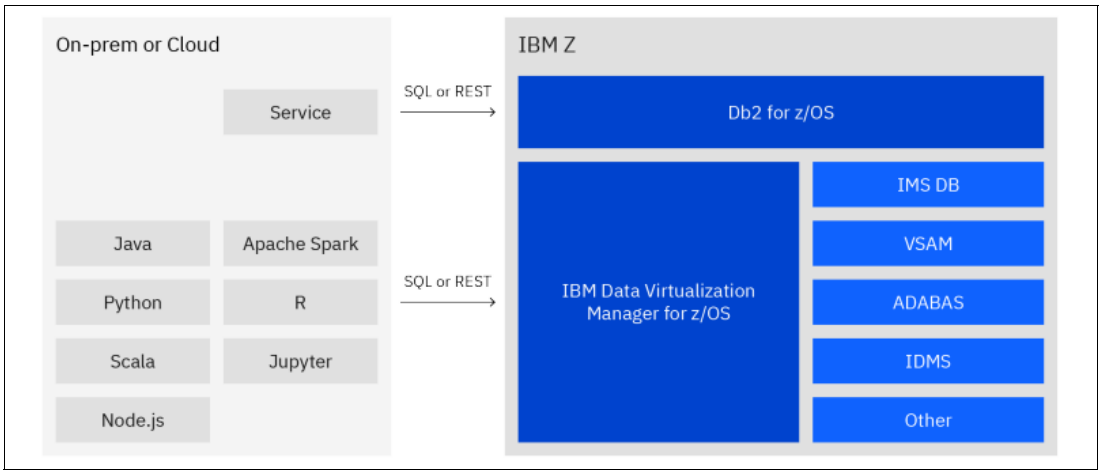


Figure 3-3 Real-time access to transactional data

You can access data that is stored in Db2 from anywhere by using SQL. Db2 for z/OS also supports native RESTful services to expose SQL and stored procedures as REST APIs when

combined with z/OS Connect EE. You can invoke Db2 native RESTful services from z/OS Connect EE by using the z/OS Connect EE REST Client Service Provider.

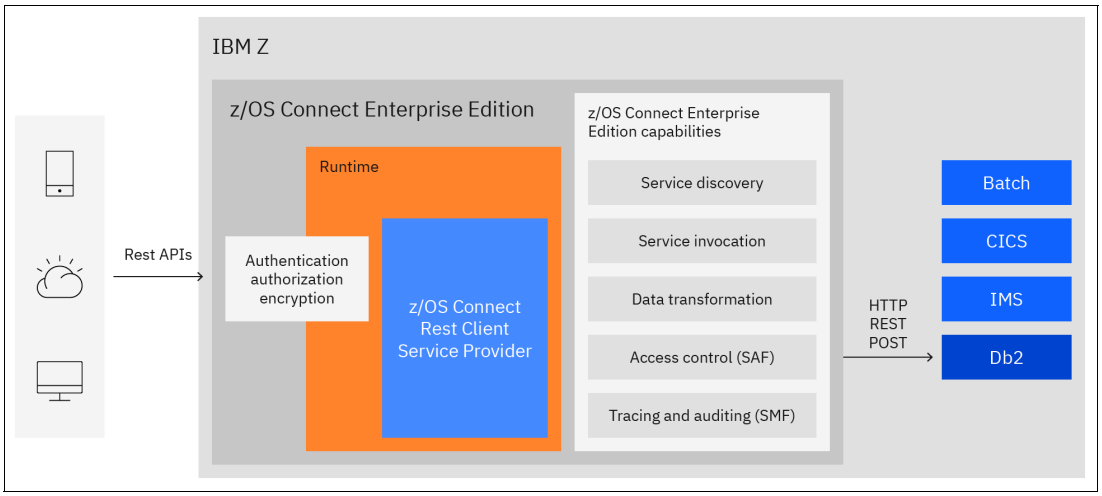


Figure 3-4 z/OS Connect capabilities

With z/OS Connect EE, you can consume IBM Z data that is stored in Db2, IMS, or data sources through REST API. Applications anywhere can consume data that is stored on z/OS. It is a common interface for cloud-native applications. IBM Data Virtualization Manager for z/OS extends SQL access to data sources other than relational databases.

Advantages

Accessing IBM Z data in place provides several critical business benefits. among which:

- Reduces the risk of data integrity
- Reduces the cost that is involved in data movement
- Increases data quality
- Preserves the existing data management and recovery processes
- Allows cloud applications to access the data at its underlying format and context
- Supports all popular access methods, such as SQL and REST APIs
- Benefits from higher performance by accessing data that is on IBM Z

Considerations

This data access pattern satisfies many data consumption needs. For example, you can run complex Db2 queries or resource-intensive queries through SQL by using the IBM Db2 Analytics Accelerator for z/OS. In fact, Db2 for z/OS can parse and resolve complex SQL statements. However, some of those queries might be resource-intensive Db2 queries. Those queries are often offloaded to be run by using the IBM Db2 Analytics Accelerator for z/OS. For example, “select * from TABLENAME where last_name like UCASE(“ALM%“) might be a resource-intensive query, as it might cause a table scan in certain situations. That example would not be an analytical query, but because it is a table scan, the accelerator would run it.

By using this pattern, you can deliver modern applications, as it facilitates and simplifies access to relational and non-relational IBM Z transactional data and combines that data with off-platform data. The pattern allows access and updates to live IBM Z data through traditional APIs such as SQL and, when combined with IBM z/OS Connect EE, to modern RESTful APIs. It also reduces the cost and delay of moving data to non-IBM Z platforms.

3.3 Virtualize IBM Z data pattern

Data is an integral element of digital transformation for enterprises. New services need simplified access to IBM Z data for business operations that require read and updates through APIs. Frequently, IBM Z data also needs to be combined with other data sources.

But as organizations seek to use their data, they encounter challenges that result from diverse data sources, types, structures, environments, and platforms. Those challenges apply equally to data that is stored on IBM Z, which contains most operational data in large organizations. A common concern is that data on IBM Z is difficult to access and transform.

One approach is to move all data into a single data store, such as an ODS or a data lake, which can create more challenges. The complexity of data copy processes results in data latency, poor data quality, increased cost, risks, and security challenges. With data virtualization, you can access data across many data sources without the need to copy and replicate data.

3.3.1 Virtualization solution and pattern for IBM Z

The foundation for consuming IBM Z data through data virtualization across data sources is the implementation of the Enable modern access to IBM Z data pattern. That pattern supports access to real-time transactional data in IBM Db2, IMS, and other data sources. You can access Db2 and IMS through SQL, Java Database Connectivity (JDBC), and REST API by using IBM z/OS Connect EE. IBM Data Virtualization Manager for z/OS can provide SQL access to all IBM Z data sources. For through REST API, you can add z/OS Connect EE.

The term data virtualization is overloaded. The main adopted use case for Data Virtualization Manager for z/OS is the mapping of traditional IBM Z data sources such as VSAM, IMS, or Adabas into relational views for modern access through SQL or API. In contrast, the main use case for data virtualization in IBM Cloud Pak for Data is to gain a single view of disparate data without data movement and to manage data with less complexity and risk of error.

The foundation for accessing data across disparate data sources is the IBM Watson® Knowledge Catalog in IBM Cloud Pak for Data. It is more feasible and less costly to maintain metadata across different data sources instead of constantly moving terabytes of changing data. Watson Knowledge Catalog is a data catalog tool that powers the intelligent, self-service discovery of data structures, models, and more. You can access, curate, categorize, and share data, knowledge assets, and their relationships wherever they are, backed by active metadata and policy management. The cloud-based enterprise metadata repository also activates information for AI, machine learning (ML), and deep learning. As shown in the following diagram, in Watson Knowledge Catalog, you can discover, govern, and catalog the metadata of IBM Z data that is stored in Data Virtualization Manager for z/OS and Db2 for z/OS.

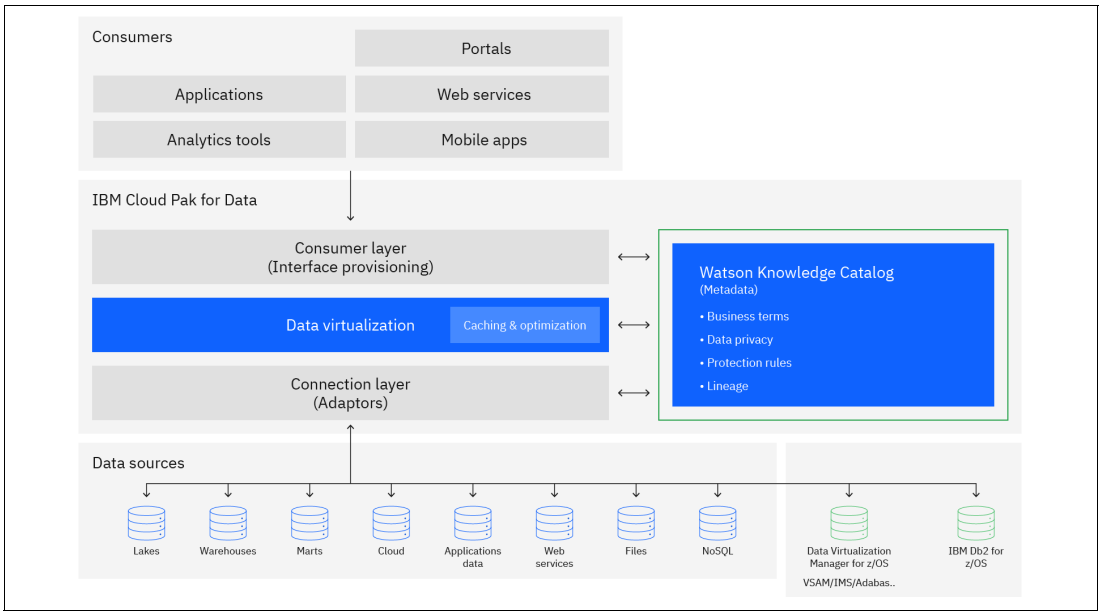


Figure 3-5 Data Virtualization Manager

IBM data virtualization is designed as a peer-to-peer computational mesh, which offers a significant advantage over a traditional federation architecture. By using innovations in advanced parallel processing and optimization, the data virtualization engine can rapidly deliver query results from many data sources. Collaborative highly paralleled compute models provide superior query performance compared to federation, up to 430% faster against 100 TB data sets. IBM data virtualization has unmatched scaling of complex queries with joins and aggregates across dozens of live systems. IBM Z data can be accessed through SQL.

Data virtualization can simplify development of consuming applications, including infusing AI into business applications. It also allows those applications to access current and accurate data at its source.

Advantages

Accessing IBM Z data in place provides the same critical business benefits as discussed in “Modern data access solution and pattern for IBM Z” , “Advantages” on page 31.

Considerations

Data virtualization in IBM Cloud Pak for Data is the foundation for rapid ML model development and deployment, infusing AI into business applications.

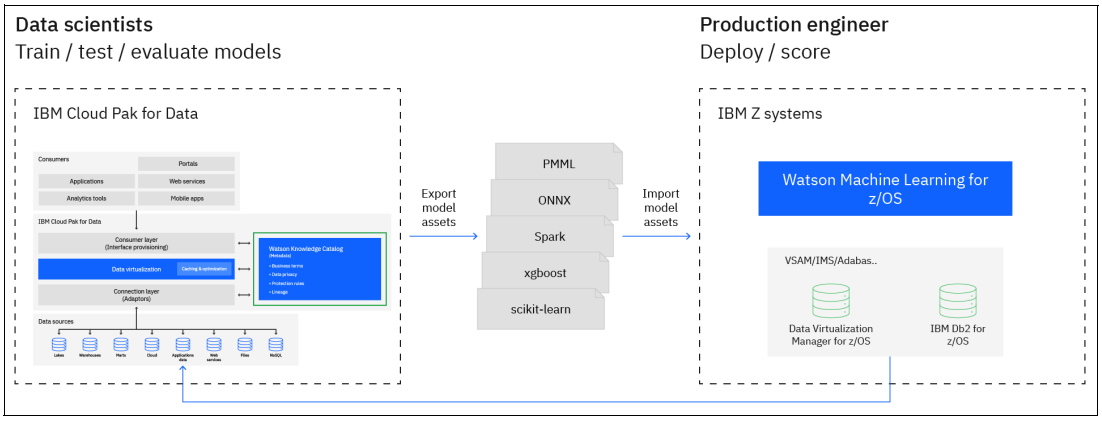


Figure 3-6 Data Virtualization and Machine Learning

With a centralized view of data, including IBM Z data, within Watson Knowledge Catalog, you can build, test, and train ML models on the platform of your choice. You can then deploy AI models to Watson Machine Learning for z/OS to address more complex information needs within business services that run on z/OS.

Deploying, and using the „IBM Data Virtualization Manager for z/OS (DVM for z/OS or just DVM) please study the following IBM Redbooks publication:*IBM Data Virtualization Manager for z/OS*, SG24-8514**.**

3.4 Cache IBM Z data pattern

The surge of new digital applications is driving huge growth in data access. Those new applications typically run read-only queries of up-to-date data, but aren't necessarily associated with generating revenue for the organization. Examples of such applications include mobile banking, retail online browsing, and insurance open enrollment. The characteristics of those applications can make them difficult to plan for and size:

- High, unpredictable volume
- Massive, sharp spikes in activity
- Updates are possible, but aren't propagated back to the source (read-only)
- Expensive to maintain
- Complex to implement
- Often compromised data currency
- Prone to instability

Historically, to address those challenges, organizations used several methods to extract data for use on other platforms. It didn't matter whether the applications were analytic or simple query-type access, the typical approach was to take the data off platform. Organizations used that approach because website developers who were accessing the data were sometimes unfamiliar with the mainframe and because of concerns that too much read-only activity might conflict with operational applications.

Traditional incremental copy and ETL approaches are unpredictable and can be associated with data latency. By using general-purpose incremental copy and ETL technologies, you can limit efficiency and performance improvement opportunities. Data extraction and incremental

copy from IBM Db2 for z/OS can use considerable resources, increase software-related costs, and compete for the same resources that are used for operational processing.

As these applications are often customer facing, the data must be up to date—only moments behind a transactional system. This requirement drives the need for more complex application logic that ensures data currency. On top of the processes to refresh the read-only data store, some organizations use customized code to ensure this consistency. Doing so adds more complexity, instability, and cost to many environments. The diagram in Figure 3-7 shows the common approach to new, highly intensive read-only applications.

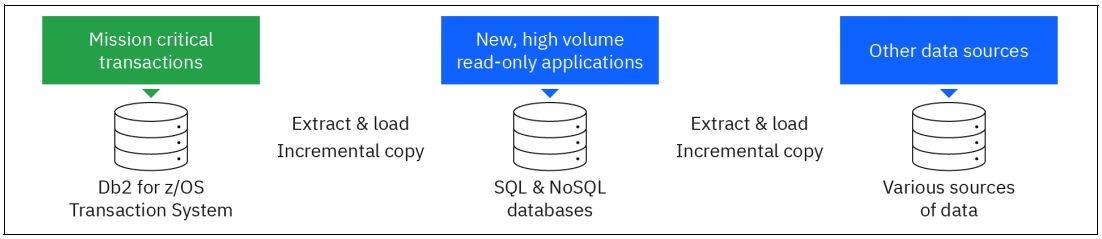


Figure 3-7 Cache IBM Z Data

Many IT organizations consider these applications necessary to support customer service but want to minimize their associated cost. Yet, as these applications are often customer facing, the data must be up to date and data access must be resilient.

3.4.1 Cache support, solution and pattern for IBM Z

Caching support on IBM Z improves application response time by storing copies of data that is on IBM Z. IBM Z offers several products that implement variations of this pattern. The IBM Db2 Analytics Accelerator, IBM Z Digital Integration Hub, IBM Db2 for z/OS Data Gate, IBM Z Table Accelerator, and IBM Data Virtualization for z/OS with Cache Option include an implementation of the cache. Some of the products include a synchronization component to maintain the cache.

Table 3-1 indicates the appropriate solution based on the application needs of the data access type.

Table 3-1 Which solution does my application need?

Data access type	Db2 for z/OS	Db2 Analytics Accelerator	Db2 Data Gate	IBM Z Table Accelerator
Operational processing on rapidly changing data	Y			N
Ad-hoc analytic processing on data that is stored in the Accelerator via Db2 for z/OS		Y		N
Access to Db2 for z/OS (and other) data that is outside of IBM Z			Y	N
High speed access to relatively static data from within IBM Z infrastructure (CICS, Cobol, etc.)				Y

Data access type	Db2 for z/OS	Db2 Analytics Accelerator	Db2 Data Gate	IBM Z Table Accelerator
Offers general processor offload	(requires zIIP)	Y	Y	Y

The two main differentiators of the cache are as follows:

- Pull versus push maintenance of the cache. In push maintenance, the cache is always kept in-sync with the source regardless of actual use. Pull maintenance involves a lazy update and invalidation of the cached values based on access.
- Cache data structure. The data structures of the cache are optimized for the consumption pattern, such as columnar or in memory. The cache itself might also contain derived or precomputed results.

Table 3-2 summarizes the key differentiators among the available options.

Table 3-2 Key differentiators

Description	Db2 Data Gate	Db2 Analytics Accelerator	Data Stage	Data Virtualization	Change Data Capture
Use case	Use IBM Z current and consistent IBM Z data on modern platform	Accelerate analytical queries on Db2 for z/OS	Automating ETL	Data integration, virtualization, make IBM Z data accessible and consumable for new users	Stream data from data sources into data lake and warehouses
Is data copied or kept in place?	Copy	Copy In place if Accelerator on IBM Z	Copy	In place/in memory	Copy
If copy, what is the typical latency?	1-10 seconds	1-10 seconds	Hours/days	No copy	Depends (30 seconds for WH, 1-10 for OLTP)
Query performance expectation	Depends on target	Optimized for transactions and analytics (HTAP)	Depends on target data store	Data is moved/re-accessed on every query	Depends on target
Data ownership and access control	Target store	Db2 on z/OS	Target store	Source	Target store
zIIP eligibility	Integrated synchronization zIIP eligible	Integrated synchronization zIIP eligible	Depending on the workload	High percentage of workload is zIIP eligible	Lower zIIP eligibility rate (about 50%)
Addresses data transformation requirements?	N	In database transformation	Y	Y	Y
Data access type	Read	Read	Read	Read/Write	Read
Target	IBM Cloud Pak for Data Db2 or Db2 Warehouse	Db2 Analytics Accelerator	Data warehouse, data lake, flat files	In-memory virtual tables	Data warehouse, data lake, flat files
Continuous replication	Y	Y	N	N/A	Y

Figure 3-8 shows how the approach shown in Figure 3-7 can be improved.

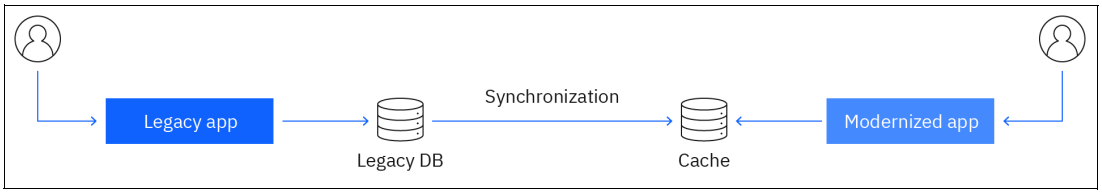


Figure 3-8 Cache IBM Z Data - improved approach

The IBM vision is for modern applications to share an integrated infrastructure with mission-critical transactional applications, without the need to build custom integration. That vision supports an integrated infrastructure that doesn't require access to the same copy of the transactional data. Db2 Data Gate delivers an integrated approach for your digital transformation without the need to be concerned about potential cost or workload implications on your operational systems.

Figure 3-9 shows an integrated approach to enterprise digital transformation.

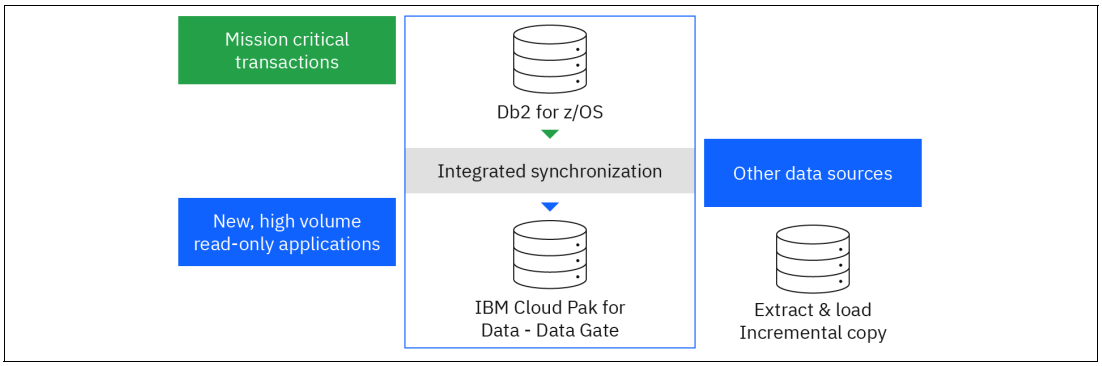


Figure 3-9 Integrated approach to enterprise digital transformation

IBM Db2 for z/OS Data Gate can help you with this specific challenge regarding Db2 data on IBM Z. You can derive value faster from data that is generated through mission-critical applications that run on Db2 for z/OS. The solution uses technology to manage the replication and synchronization between the source and target. The source data always remains secure on IBM Z, and all insert, update, and delete actions are completed in Db2 for z/OS. You can define instances of Db2 Data Gate on IBM Cloud Pak for Data and use the IBM Cloud Pak for Data platform to build new applications and analytical models from Db2 for z/OS data without impacting the source system. In this sense, data needs from lines of businesses are fulfilled while transactional workloads on IBM Z remain secure and stable.

You can access Db2 for z/OS data in other ways, such as through z/OS Connect, REST API, and Java Database Connectivity (JDBC) or Open Database Connectivity (ODBC). But nothing comes close to Db2 Data Gate in terms of performance, simplicity, and cost-effectiveness. Db2 Data Gate eliminates that complexity, providing timely, fast cloud access to your Db2 for z/OS data. Compared with traditional replication methods, Db2 Data Gate performs magnitudes better at a fraction of the CPU cost.

Advantages

Optimizing application performance by accessing cached IBM Z data can provide several critical business benefits:

- Achieves service level agreements (SLAs)
- Improves performance in scenarios where data is read repetitively and at high frequencies

- Provides a good compromise between cost and complexity
- Improves efficiency by avoiding hitting database or other data sources every time for the same request
- Integrated, lightweight data synchronization
- Excellent performance and resiliency
- Lower latency and better data currency

While caching is commonly used to improve application latency, a highly available and resilient cache can also help applications scale. By offloading responsibilities from the application's main logic to the caching layer, you free up compute resources to process more incoming work. Read-intensive applications can greatly benefit from implementing a caching approach.

Considerations

Applications can tolerate various levels of latency depending upon the nature of the specific application. For example, a data warehouse reporting system that runs a balance sheet and income statement after a close doesn't need up-to-the-second data. However, a trading application that depends on exact information has no tolerance for latency. In fact, even microseconds might mean a different decision. Algorithmic trading depends on a futures price and the price of a basket of stocks that the option represents. Any significant variation from those prices means that the trade might be unprofitable. The more latency in data, the greater the risk that your decision is wrong.

If you're leaving the Db2 for z/OS environment, the only security that you can depend on is the security of the target system that the data is being moved to. You need to control database- and application-specific access for that database because none of the security attributes are passed on with the data.

3.4.2 Examples of customer scenarios for data caching

The following are examples of typical data caching scenarios.

1. Replicate once (to cache), use many
   - Ideal use case: read-intensive applications that can tolerate some level of latency. Usually analytics, dashboards, customer summary records, batch processes, notification and mailing use-cases
   - Isolate/protect original source system from new read-intensive workloads
   - Save processing cost on mainframe
   - One-way synchronization, to avoid data integrity problems and the need to update conflict resolution
   - Achieve data and function co-location
   - The low latency data access is especially important for hybrid cloud scenarios where the on-prem system of record must stay as is
2. Low-risk modernization
   - Keep existing data source and surrounding application landscape as is
   - Add a selective cache of the tables that new applications need
   - The cache is synchronized one-way, from source to target
   - Writes go to original source/system of record, to avoid any form of data integrity issue

3. “Self-service access to use-case and/or application-specific data” for cloud consumption
   - Data mesh architecture: separation from the product team and a team that provides data “as a service”
   - Data is served in form of a replicated cache, maintained by the data team, and provided “as a service”
4. Mobile applications need current/low latency data but have high variability in workloads (day/night/weekend)
   - Extend transactional system with a new mobile app where the backend is developed cloud-native, but data access is to the system of record as current data (second accurate) is needed
5. Data Cache as a data delivery method for a data fabric
   - Even though data is copied, fabric helps to still enforce lineage, security, and policy rules to make sure that e.g. only in-country access to data is possible
   - Also, since cached data is registered in the catalog including relation to original source (connection), this helps “smart data integration” use-cases where applications decide if direct access or cached access is better.
   - Data discovery: represent cached data in a data catalog. assign business terms. policy rules can be applied to the discovered metadata (i.e. this column is PII, mask PII with every access that is not in-country)

3.5 Transform IBM Z data

The transformation of systems-of-record (SOR) data by software processes to create a wholly new data set is a specialization of a broader copy-and-access use case. That use case also includes extract, transform, and load (ETL) processes, software replication processes, and virtualization and federation processes. All of those broader processes include some transformation capabilities, and it's common to require light transformations during otherwise routine copy-and-access use cases.

This pattern involves heavy set-based transformations that create a data set that is composed by external feeds, derived transformations, and source SOR data sets, such as aggregations or consolidations and summations. Currency requirements dictate the use of either real-time mechanisms, such as virtualization, near-real-time mechanisms, such as software replication, or periodic batch mechanisms, such as ETL or extract, load, and transform (ELT).

You might need combinations of those three mechanisms if set-based transformations, which are typically the province of ETL products, are required in addition to real-time or near-real-time currency. While this pattern applies to SOR data on any platform, its use with SOR data on IBM z/OS is relevant and valuable because many large enterprises use z/OS as an SOR.

In the following simplified depiction, the transformation processes are indicated by the Data Adapter box. In practice, these transformations can span both processes and time.

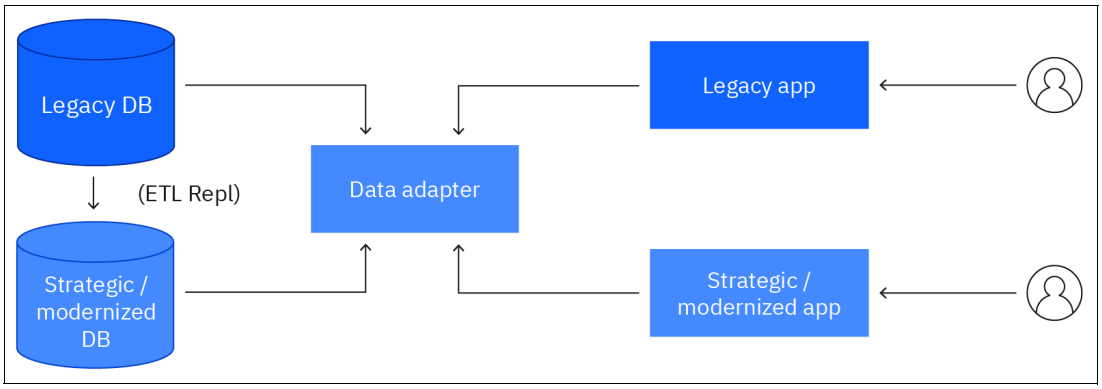


Figure 3-10 Transforming data

3.5.1 Data transformation solution and pattern for IBM Z

On z/OS, it's typical to have solutions that maintain logically related data across different stacks, such as IBM Db2, IMS, VSAM, or sequential files. The ability to view this data through federated queries or from an aggregated copy has value in and of itself. Add the ability to extend the aggregation to derive data (summations, transformations) and to add sources (distributed databases, external feeds), and the value of the original z/OS SOR data grows without disrupting the original workloads.

Beyond data creation, you can use this pattern to create schemas over data. One common use case is to convert normalized data models from the SOR to a de-normalized data model. A de-normalized data model might be one that is used in data warehouse solutions and dimensional data marts that are optimized for statistical and analytical processing. Another common use case is to create user-oriented schemas from what might be a product orientation at the SOR. This use case enables consumption by people who are less familiar with the internal view of the technology or products that underpin the data.

IBM has industry-leading product capabilities in the data transformation space. One such product is Data Stage, which is available in on premises, on IBM Cloud Pak for Data, and in cloud implementations.

Advantages

Creating data sets with extra attributes that are derived from production data sets allows for extensibility with minimal disruptions. It also reduces the need to maintain the same data in multiple stores without synchronization.

- No changes are required to the original SOR data sets. All of these methods apply transformations downstream from the original data.
- These methods all provide remote access to SOR data in addition to the transformation of that SOR data.
- Downstream data sets can be created or modified without impacting the source SOR data that they are derived from.
- New applications can be created outside of the context of the original SOR data.
- The impact to core SORs from downstream workload requirements is mitigated.

Considerations

When you create transformed data from SOR data, you need to consider a few factors. Any changes to the source SOR schema or content can have a downstream impact on the new,

derived workload. Modify SOR procedures and processes to account for the impact on the new, derived workloads. Make sure that lineage and provenance are discoverable to ensure that the downstream processes are maintained correctly.

4

Event-driven architecture with z/OS

In this chapter, we discuss even driven architectures with event-driven solutions. This chapter describes the following:

- Overview of event-driven architecture
  - Reference Architecture
  - Flavors of Event Processing
- Overview of event-driven architecture patterns with z/OS
  - Respond to IBM Z events
  - Respond to external events
  - Optimize CQRS
- Introducing event-driven architecture in z/OS Ecosystem
  - Deep Dive - Respond to IBM Z events
    - Sample use case,
    - Architecture
    - Benefits
    - Instruction set
  - Deep Diver - Optimize CQRS
    - Sample use case
    - Architecture
    - Implementation guide
- Conclusion

4.1 Overview of event-driven architecture

As internet capabilities continue to grow and bringing the world closer than ever, users are empowered to perform business functions with the tap of a few buttons through mobile or web applications. It has become necessary for businesses to reinvent themselves in a competitive world by developing inter-operable, loosely coupled, high throughput systems that offer better, reliable, near real-time user experiences and hence, event driven architecture.

From a non-technical perspective, an event can be considered as anything and everything happening or changing around us. Event driven architecture is building a system which can sense, detect, and capture such events, put them in context and apply intelligence over those events and determine the next best action or decision.

Event-driven architecture is rising to popularity, mainly because of its ability to complement microservice architecture style and agile practices. Together, microservice architecture styles and agile practices overcome limitations such as inflexibility in adding or modifying components, expensive and slower development and an inability to scale up or down quickly that traditional monolithic systems are having.

For additional information on event driven architectures, see the following website: https://www.ibm.com/cloud/architecture/architectures/application-modernization-mainframe/patterns

4.1.1 Simplified reference architecture

Figure 4-1 describes the basic components and their connectivity with each other that are involved in event driven architectures.

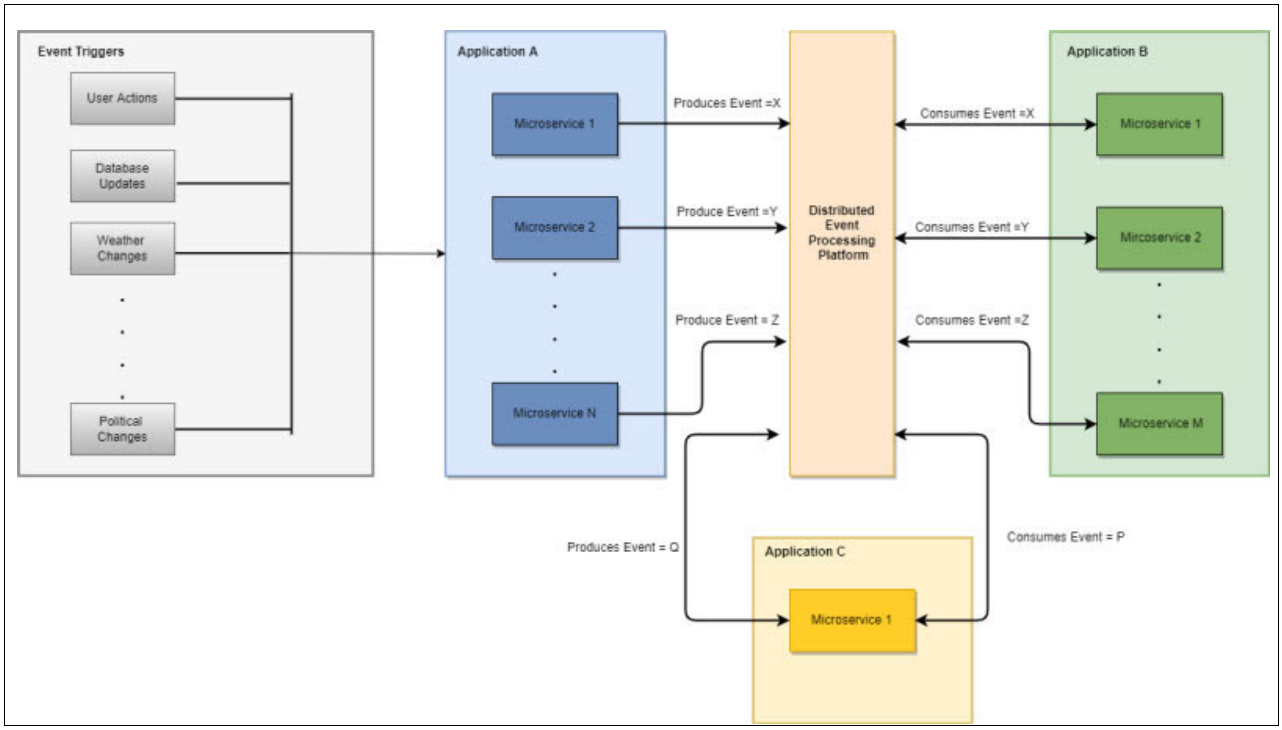


Figure 4-1 Simplified event-driven architecture

The basic components of simplified event driven architectures include:

- Event triggers, may or may not be part of the technical system, they cause change in state for a subject of interest in real world. Triggers could be technical, political, environmental, essentially, anything under the sun. Depending on the scope of application/project, we can accommodate which triggers we want to cover.
- **Producers**, hosted either at same or distributed servers, capture real life events, translate them in technical format and emit to event processing platform, understandable for other technical systems.
- **Event processing platform**, is distributed system for high availability, acts as temporary storage for event, helping event navigate to required consumers. It also can be visualized as message broker among disparate systems.
- **Consumers**, hosted either at same or distributed servers, continuously poll event processing platform. Generally it is designed in such a way that, each consumer is a microservice and is triggered only when event of interest is found. Sometimes, consumers also can act as producers, to publish separate events, causing chain of events, example is shown in Figure 4-1, microservice inside Application C

Typically, events are captured and processed by using any of the following, depending on the use cases. The following are general categorizations and are not a holistic view:

Message Topics: are generally used in publish-subscribe model, where each message can be consumed by multiple subscribers. For example: Business publishing a message to all customers subscribed for promotional events, regarding holiday offers.

Kafka: is used when we expect high throughput, i.e. processing high load of data in less time. The processed data is used to feed analytical engine, to build intelligence around information. For example: Depending on videos seen by users in past, build a user profile and recommend new videos.

File System: are used to automate processes, to avoid manual intervention. For example: Receiving an invoice at FTP location, triggers a bot to send an e-mail to vendors.

Message Queues: are generally used when we need to maintain sequence of each message while processing.For example:In supply chain management, the shipment goes through different statuses, such as departed, booked, landed, delivered etc. These events need to be processed in order, without jumbling.

4.1.2 Types of event processing

To appropriately discover an use case and design, we need to categorize events as follows:

1. **Simple Event Processing** or SEP, typically detect change in state, in context of specific workflow or business process, using which next step is determined. The scope of event is limited to current and next state only. For example, In payroll system, employee submitting leave application, triggers simple event, which are consumed by approval process, salary distribution process.
2. **Complex Event Processing** or CEP, typically detect and correlate change in state, to other past or future events, and establishes pattern to determine next best action. For example: In smart home systems, whenever water is used, an event is generated. When it is detected that water consumption is more than user's typical water consumption in last 30 days, by analyzing all the events received in past, the system shuts down the water supply, and notify user about possible plumbing issue.
3. **Event Stream Processing** or ESP,is typically used when high throughput is expected to build an intelligence at real time, over streams of events received. For example: In Global Positioning System, when all cars on road slow down due to traffic, GPS in each car emit

an event about current speed. The system, correlating all these events modifies route color from blue to yellow or red, signifying delays, on GPS user interface.

4.2 Introducing event-driven architecture in the z/OS ecosystem

In this section, we will take an opportunity to deep dive into a couple of patterns. We will take a sample scenario and navigate through current vs. proposed architectures, their benefits, some considerations and provide a brief implementation guide. The patterns we will discuss in this section include:

- Section 4.2.1, “Deep Dive - Pattern: Respond to IBM Z application events”:

  Share events that are generated in IBM Z applications so that new application logic can be developed to respond to such events without introducing risks in core applications. Analyze data as part of application logic to generate an event.

- Section 4.2.2, “Deep Dive - Pattern: Optimize CQRS”

  Deliver an efficient CQRS system that is based on IBM Z to optimize the synchronization between the command access, which is SOR data that is updated by online and batch applications, and the query access, which is an information model that is aligned with the needs of the new applications. Optimize by using IBM Z Digital Integration Hub to deliver a non-disruptive, low-latency, high-throughput, and cost-attractive solution.

An additional pattern, not discussed here is Respond to external events. This pattern allows you to share events that are generated by applications that are external to IBM Z to drive the invocation of IBM Z application logic. You develop flexible logic without introducing risks into the core applications

4.2.1 Deep Dive - Pattern: Respond to IBM Z application events

To demonstrate the different possibilities of hybrid cloud with z/OS and to be consistent with agile principles, we will start small and then scale, using the following sample scenario.

Sample scenario

A bank having thousands of customers, processing millions of transactions a day, decides to extract meaningful information from credit card transactions that customers are making to serve customers better and increase customer satisfaction, or CSAT, without disrupting business as usual. This includes establishing a prototype, which could ingest credit card transactions in real time to build an intelligence model. It is meant to be a foundation, which could further be extended, enriched and enhanced for the following use cases, as an example.

- Campaign management, to build a user profile, so the right customer gets the right product offers.
- Categorizing and tracking expenses made through a credit card.
- Credit card fraud detection.

Hence, in scope of design and implementation, we will talk about different alternatives to integrate event driven architecture with z/OS, provide instructions to set up the infrastructure suitable for one of the designs, deploy services to connect with the event processing platform and provide sample code producing and consuming events technically. We will not provide any business logic in this example.

Typical as-is (current) architecture

The simplified version of the existing system, shown in Figure 4-2, demonstrates credit card transactions as business as usual (BAU).

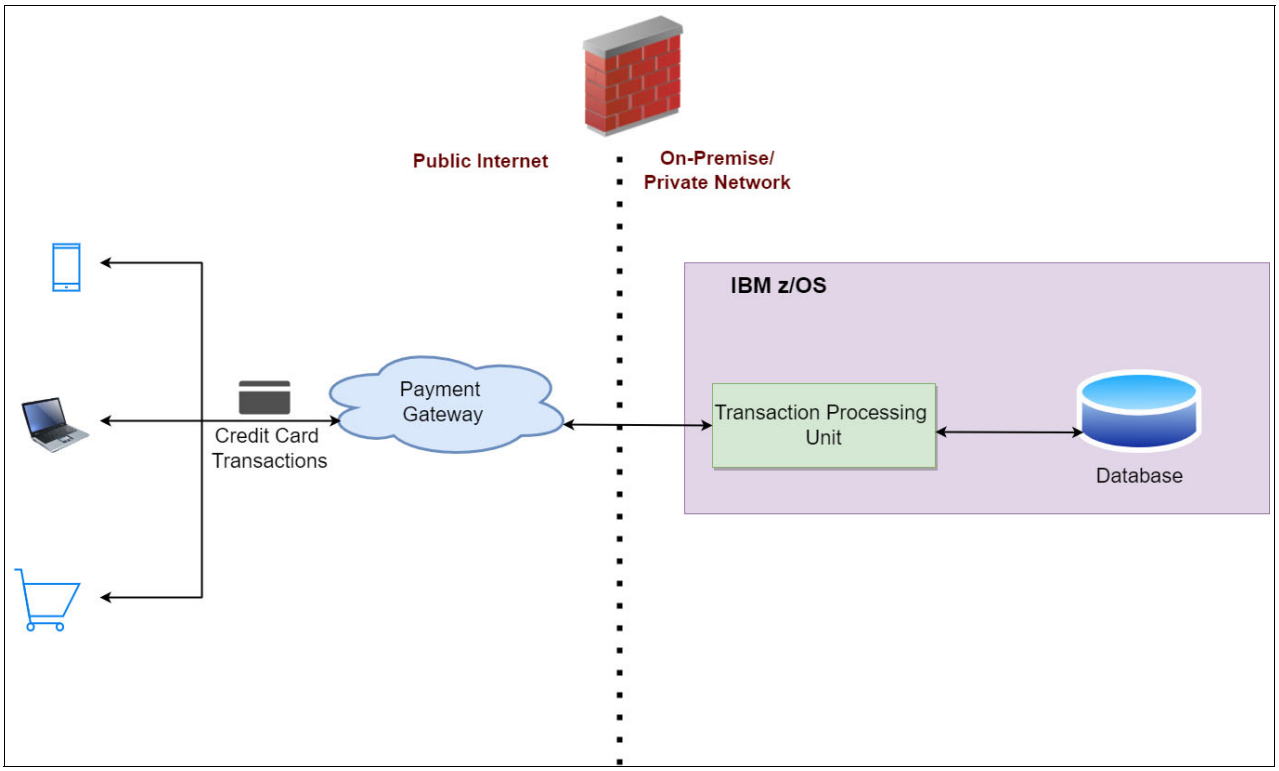


Figure 4-2 Typical as-is architecture

The following is the simplified processed used in a typical as-is architecture:

- A user makes a transaction by using a mobile, laptop or any other means, over the public internet.
- The Payment Gateway sends the transaction to the bank to process it.
- The Transaction Processing Unit, hosted on IBM z/OS, receives, processes and approves or rejects the request.
- The result is returned to the payment gateway as well as stored in a database.

We will enhance this system by feeding data received by the transaction processing unit to additional components in subsequent sections.

Proposed to-be architecture: Phase 1

To leverage the power of z/OS with a hybrid cloud, we will introduce new components step by step to ensure minimal or no impact on the existing system. This process allows businesses to adapt to modern development practices, such as agile and devops, while enjoying the benefits of z/OS computing power.

Figure 4-3 shows the proposed architecture for Phase 1.

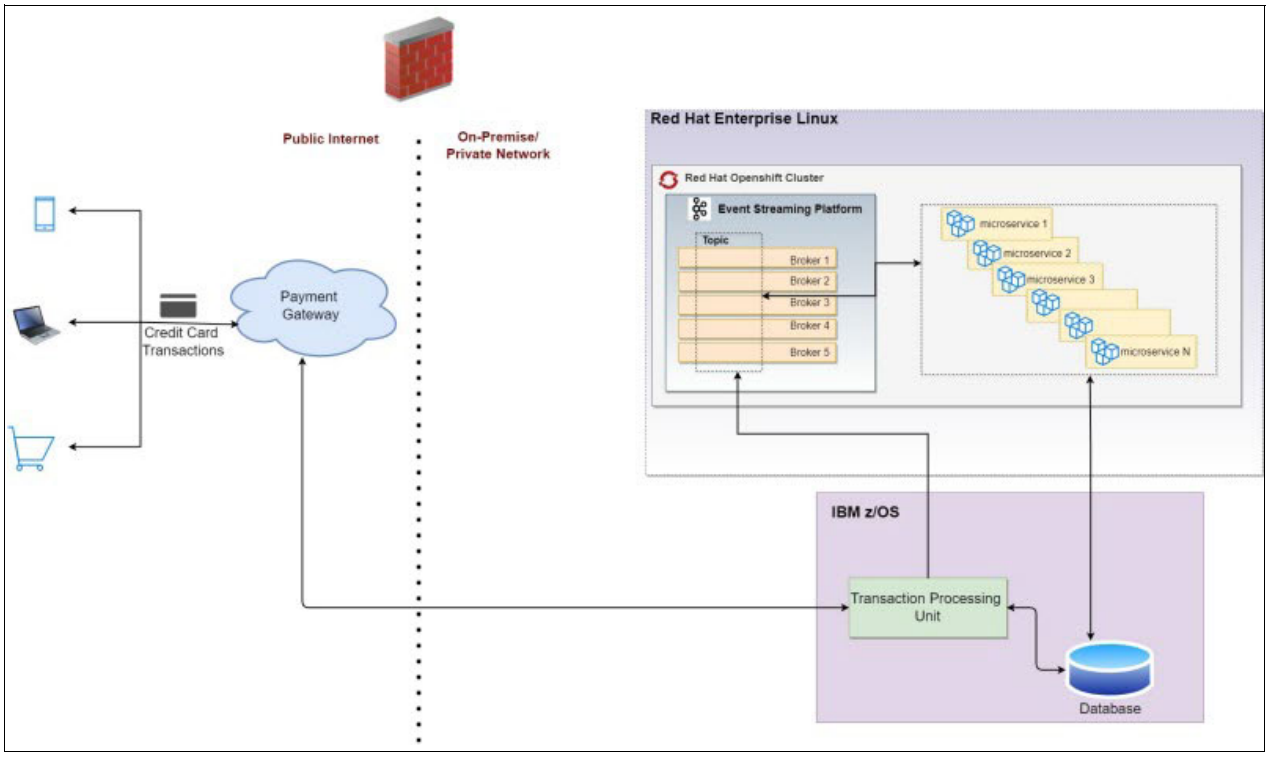


Figure 4-3 Proposed to-be Architecture, Phase 1

Figure 4-3 introduces component to ingest existing data. To start with, the new setup is made on a separate Red Hat Enterprise Linux box, either on premises or on a private network and decoupled from the existing system, adhering to our principle of not disrupting the existing system or business-as-usual. It will also help us to measure the performance matrix separately for a new implementation, along with separate application management (deployments, logging, monitoring etc) if needed. The containerization of the event streaming platform and microservices in OpenShift will make implementation platform independent, for example, the solution provides flexibility to deploy OpenShift on-premise or on cloud, if needed with the cloud provider of the customer's choice.

From a technical perspective, in addition to existing functionality happening as-is, this approach sets up an event streaming platform and set of microservices, containerized in OpenShift and installed on the Red Hat Enterprise Linux machine.

The existing Transaction Processing Unit is extended to emit events for each transaction made on the event streaming platform. Containerized microservices are continuously polling the event streaming platform. As soon as an event is received, the respective microservice of interest triggers and runs the business logic.

Proposed to-be architecture: Phase 2

Once we have functioning prototype, it would be "lift and shift" of components from one OpenShift cluster from Red Hat Enterprise Linux to another on z/OS, as shown in Figure 4-4.

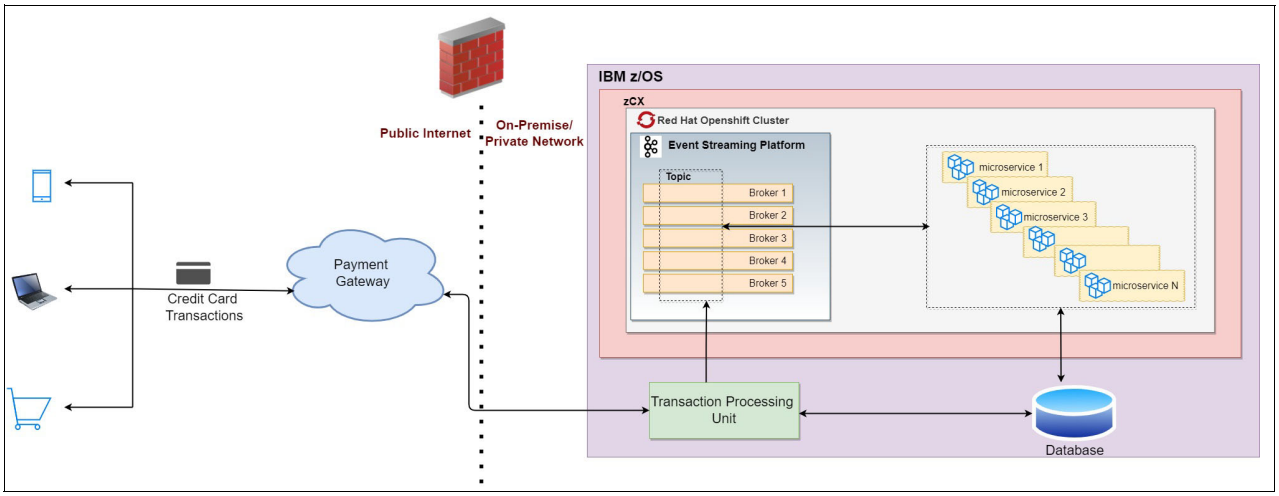


Figure 4-4 Proposed to-be architecture, final state

However, z/OS Container Extensions (zCX) is pre-requisite in order to install OpenShift on z/os, instructions of which are shared in “Installing Red Hat OpenShift Container Platform” on page 50. From technical perspective, the interfaces among components are unchanged, but getting OpenShift closer to existing system, will help to reduce latency and provide closer control to developer on existing and new implementation.

Benefits of proposed architecture

Event-driven architecture used above, combined with microservices and agile methodologies provides following benefits:

- **Asynchronous processing**: provides efficient use of resources, loosely coupled resilient services and capability of replaying events if needed.
- **Cross-technology/platform integration**: allows developers to choose different languages for different services and still integrate seamless with other modules.
- **Continuous integration/continuous deployment**: automates the process from source code development to deployment on server/runtime, helping scalability, reliability and availability.
- **Rapid Go to Market strategy**: helps businesses to react on events happening around and take actions in quick time, without compromising any of the existing applications/systems.
- **Cost effective**: solutions/components can be rolled out in phases, additionally technical resources are used/scaled only when events occur and not blocked otherwise.
- **Segregation of responsibilities**: among independently deployed components, help in running system, for area of problem, to be detected, corrected and deployed, without affecting other components/systems.
- **Maintaining skillset in team**: is easier, with compared to monolithic systems, as components are not technology/platform dependent

Considerations

Some things to keep in mind when using this proposed architecture:

- The design is technology agnostic giving reader flexibility to choose technologies as per his/her choice, microservices can be in java or python or node or any other language, database can be db2 or sql and so on.

- Though design depicts connectivity to database from technical components in zos, the implementation of same is already proven and hence, not included in the implementation guide.
- The same architecture can be mirrored, if events are sourced from external components, by putting event streaming platform and microservices in public cloud.
- Other existing systems, functionalities, non-functional parameters are kept out of design intentionally, to focus on event driven architecture. It is well-supported by proposed design and expected to be integrated in real time applications

Implementation

This section provides implementation level details to achieve the proposed architecture. The main focus area will be on event driven components, such as implementing, deploying and testing microservices using event streaming platform.

For the purpose of our use case, we will use the following:

- IBM Event Streams as our event processing platform,
- Red hat OpenShift Container Platform, as our hosting platform
- Java microservices, as our consumers of events
- Transaction processing unit which will act as producer. (which will be simulated with another Java microservice).

Installing Red Hat OpenShift Container Platform

The detailed instructions are provided on following website:
https://docs.openshift.com/container-platform/4.11/installing/index.html

The instructions to install OpenShift on z/OS through zCX can be found in the following IBM Redbooks publication:

- *Red Hat OpenShift on IBM Z Installation Guide*, REDP-5605

Installing an event processing platform on Red Hat OpenShift

Application team has choice to install flavor and provider of event processing platform of their choice as-a-service on Red Hat OpenShift, including IBM MQ, Red hat AMQ, Apache Kafka, IBM Event Streams, Strimzi Kafka, to name a few, from OperatorHub on OpenShift console. For the purpose of our use case, IBM Event Streams is used,

The instructions to install IBM Event Streams can be found on the following website:
https://ibm.github.io/event-streams/2019.4/installing/installing-openshift/

Similarly, instructions to use Strimzi Kafka can be found on the following website:
https://strimzi.io/blog/2019/03/06/strimzi-and-operator-lifecycle-manager/

Deploying microservices on Red Hat OpenShift

This being a containerized solution, an application team can choose various programming languages without worrying about environment, deployment, networking rules etc. In fact, developers can enjoy continuos integration/continuos deployment with in-built support for various buildpacks by OpenShift, and focus more on business functions.

For the purpose of our use case, we chose to deploy the following Java microservices developed in Spring Boot, which is an open source Java-based framework used to create microservices.

kafka-demo-producer

This would simulate the events the Transaction Processing Unit emit, on the topic configured in the Kafka service. The following website provides the Git repository for our kafka-demo-producer:
https://github.com/IBMRedbooks/SG248532-zos-hybrid-cloud-examples/tree/main/Chapter04/kafka-demo-producer

kafka-demo-consumer

This would be the listener application, polling a Kafka topic, where actual business logic should reside when we extend functionalities. The following website provides the Git repository for our kafka-demo-consumer.
https://github.com/IBMRedbooks/SG248532-zos-hybrid-cloud-examples/tree/main/Chapter04/kafka-demo-consumer

Prerequisites

The following are the prerequisites needed to deploy microservices.

- OpenShift cluster is provisioned and developer has sufficient access permissions/role, on the cluster
- If cluster is on-premise or private network, user is connected to network with VPN or similar
- Client tools to access cluster from command line are installed on developer machine, please find installation instructions on following website:
  https://docs.openshift.com/container-platform/4.8/cli_reference/openshift_cli/getting-started-cli.html
- The server certificate is installed on client machine, to enable secured connection. Please procure same from your cluster administrator
- Since the example is developed in java spring boot, having familiarity with same, along with Maven is beneficial.
- Personal Access Token is setup, in case source code is referred from github. Please find instructions on following website:
  https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

Deployment Instructions

The following are the step-by-step instructions to deploy Java microservices.

1. Log in to the Red Hat OpenShift cluster by using the following command from a command line:

   $oc login -u <userName> --certificate.authority=<path_to_certificate> --server=<server_url>

   You will be prompted for the password, after which, output similar to Example 4-1 will be displayed

Example 4-1 Login to cluster

```
Authentication required for <server_url> (openshift)
Username: <userName>
Password:
Login successful.
You have access to 78 projects, the list has been suppressed. You can list all
projects with 'oc projects'
```

2. Create a new project. It is always good idea to segregate services per functional domain for better maintenance. The following command will create a project in the container.

   $ oc new-project <project_name> --display-name="<display_name>" --description="<description>"

   Output should look similar to that as shown in Example 4-2.

Example 4-2 Create a project

```
Now using project "<project_name>" on server "<server_url>"
```

3. Build and deploy the microservice. This is where things start getting more interesting. OpenShift supports an array of options, such as using containerized images based on Docker. The following are ways you can build your Dockerfiles and push the image to a repository.
   - Build- for more details, see:

     https://docs.docker.com/engine/reference/commandline/build/
   - Quay.io: Supports the ability to build Dockerfiles and push the resulting image to a repository. For more details, see:

     https://docs.quay.io/guides/building.html
   - Native build or source to image (S2I) build using a variety of technology stacks.

     https://docs.openshift.com/container-platform/3.11/creating_images/s2i.html

We used an S2I build for our use cases. This is a multiple-step process.

1. First, pull a buildpack compatible with Red Hat OpenShift, needed to build an image, in our case Java, by using the following command:

   $ docker pull <JDK_BUILD_TAG>

   For example:

   $ docker pull registry.access.redhat.com/ubi8/openjdk-11:1.14-3

   Output should be similar to that shown in Example 4-3.

Example 4-3 Pull prerequisite images

```
Digest: sha256:8054b2aac795530eea7e0053343c624c96b661f38d99c51997dad91a4c32e094

Status: Downloaded newer image for
registry.access.redhat.com/ubi8/openjdk-11:1.14-3

registry.access.redhat.com/ubi8/openjdk-11:1.14-3
```

2. Next, run the build to create an application in OpenShift project. The following command will fetch source code from a centralized repository, build the deployable unit of source code using buildpack downloaded and finally, push the deployable unit to OpenShift project as an application and start it

   $ oc new-app <JDK_BUILD_TAG>~<GIT_URL> --context-dir=<NAME_OF_DIR_IN_GIT> --name=<APP_NAME>

   For example:

$ oc new-app registry.access.redhat.com/ubi8/openjdk-11:1.14-3~<GIT_URL> --context-dir=/kafka-demo-producer --name=kafka-demo-producer

Output of command should be similar to Example 4-4.

Example 4-4 Create an application

```
--> Found container image 790ba43 (4 weeks old) from registry.access.redhat.com for
"registry.access.redhat.com/ubi8/openjdk-11:1.14-3"

    Java Applications

----------------

    Platform for building and running plain Java applications (fat-jar and flat
classpath)

    Tags: builder, java

    * An image stream tag will be created as "openjdk-11:1.14-3" that will track
the source image

    * A source build using source code from
https://github.com/amey0309/kafka-demo.git will be created

      * The resulting image will be pushed to image stream tag
"kafka-demo-producer:latest"

      * Every time "openjdk-11:1.14-3" changes a new build will be triggered

--> Creating resources...

    imagestream.image.openshift.io "openjdk-11" created

    imagestream.image.openshift.io "kafka-demo-producer" created

    buildconfig.build.openshift.io "kafka-demo-producer" created

    deployment.apps "kafka-demo-producer" created

    service "kafka-demo-producer" created

--> Success
```

3. Validate the status of the application by using the following command:

$ oc status

Output similar to Example 4-5 should be displayed.

Example 4-5 Validate status

```
In project <project_name> on server <server_url>
svc/<APP_NAME> - <IP> ports <PORT_NUMBERS>
  deployment/<APP_NAME> deploys istag/<APP_NAME>: <TAG_NAME>
    bc/<APP_NAME> source builds <GIT_URL> on<BUILDPACK>
      build #1 running for 36 seconds - bd02973: should trigger build in quay
(<GIT_USER_NAME>)
```

```
deployment #1 running for 37 seconds - 0/1 pods growing to 1
```

4. Create a route in case you want to expose a REST endpoint for the application. In our use case, for consuming events, we did not need to create a route. However, we will be simulating producing events by posting a message on an end point. Hence, the route will be needed for the producer component. We expose the route by using the following command:

   $oc expose svc <APP_NAME>

   For our example:

   $oc expose svc kafka-demo-producer

   The output should be similar to Example 4-6.

Example 4-6 Create route

```
route.route.openshift.io/kafka-demo-producer exposed
```

Perform the steps from step 3 on page 52 to step 4 on page 54 for kafka-demo-producer as well as kafka-demo-consumer.

5. Testing the Kafka connection. Now your application is ready to be tested. We will produce an event using a REST endpoint, which should be consumed by the consumer application in real time. The business logic to be built either to process simple, complex or stream events is to be placed in the consumer application, using the respective APIs.

   The following command can be used to emit a simulated event to Kafka:

   **$curl -X POST -H “Content-Type: plain/text” \ -d 'Simulated message for kafka-demo' **

   <SRVC_ROUTE>

   Where SRVC_ROUTE is the URL of the REST endpoint we created for the kafka-demo-producer.

   Output should be similar to Example 4-7.

Example 4-7 Output of producer

```
Message posted successfully!
```

At the same time, this message should be processed in kafka-demo-consumer, and should appear in logs to show that data is flowing from the producer to Kafka to the consumer successfully by using Red Hat OpenShift in the z/OS environment.

Important: The properties of kafka that are used in codebase are provider dependent. Ensure accurate properties are set as per kafka provider documentation.

6. Monitoring the application is possible by using a command line as well as administrator console. The following website describes the steps to do this:

https://docs.openshift.com/container-platform/4.11/applications/application-health.html

Possible Alternate Solutions

The solution outlined here is designed specifically keeping in mind that the client business is very new to the cloud journey. There are other possible solutions to achieve a similar outcome

that you can choose. We provide a brief overview of some other possible solutions, which were considered while writing this chapter.

Alternative 1

Figure 4-5 is a slight variation of the design that we discussed in detail and shown in Figure 4-3. The difference is that the event streaming platform is not containerized, instead, it is installed directly on Red Hat Enterprise Linux, or for that matter any machine on premise. It decouples the business logic in microservices from the infrastructure.

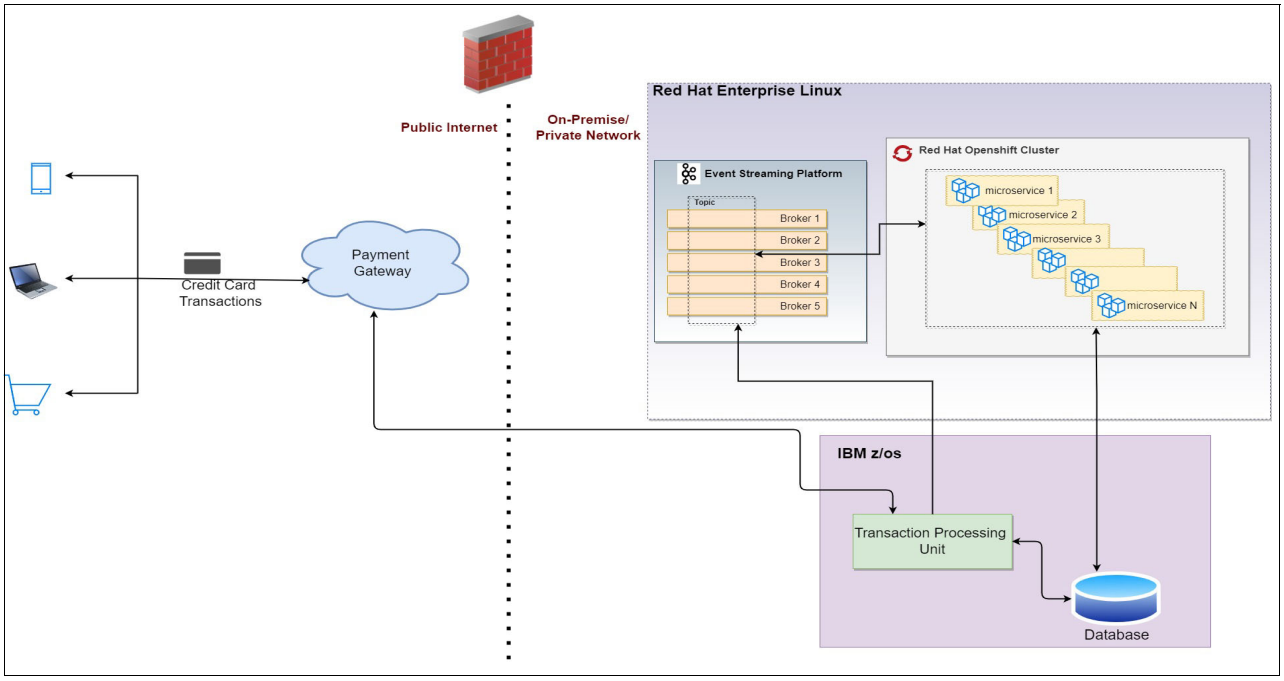


Figure 4-5 To-be Architecture - Alternative 1

Alternative 2

IBM also offers Cloud Pak for Integration, Cloud Pak for Automation and Cloud Pak for Data, which can be installed on any cloud environment, to multiply the benefits that z/OS and event driven architecture provide, including seamless integration and a better user experience.

For this alternative use case, you can choose to install IBM Cloud Pak for Integration on Red Hat OpenShift deployed on z/OS.

More details can be found on the following websites:

https://www.ibm.com/cloud/cloud-pak-for-integration

https://www.ibm.com/cloud/cloud-pak-for-business-automation

https://www.ibm.com/products/cloud-pak-for-data

4.2.2 Deep Dive - Pattern: Optimize CQRS

Enterprise organizations are responding to both business pressures and opportunities through digital transformation and modernization initiatives. Although these efforts can yield significant positive results, effective transformation for enterprise clients is often underpinned by how well core systems of record integrate and interact with hybrid cloud deployments.

As clients move to more open and modular architectures based on industry aligned references, such as the Banking Industry Architecture Network (BIAN) for banking, the efficient flow of information between systems of record and cloud applications becomes even more essential.

In addition, trends across many industries are creating the need to transform business processes into real-time or near-real-time by using Command Query Response Separation (CQRS). This trend includes even those industry use cases that traditionally are satisfied with latent information. Typical CQRS patterns leverage event-based mechanism to achieve the real-time delivery of relevant information to business processes.

For those environments that have a high volume of transactions and low latency needs for real-time information, what is needed is a performant, secure, cost-effective, modern way to share information with hybrid cloud. The Optimize CQRS pattern is a specifically optimized pattern for systems of record (SOR) which are on IBM z/OS. For IBM zSystems enterprise clients, this includes application systems such as core banking, claims adjudication, wealth management, card processing and many more.

Sample Scenario

Efficient flow of business process information and inquiry-driven interaction between cloud applications and z/OS systems of record, using a non-disruptive, progressive modernization approach is valuable for many scenarios across industries. One example industry where this is most relevant is banking and a category of scenarios that can leverage an optimized CQRS pattern for greater return-on-investment (ROI) and includes ecosystem expansion.

Across the banking industry, many FinTech (technology used to support or enable banking and financial services) and RegTech (the management of regulatory processes within the financial industry) companies are making their capabilities available in a Software-as-a-Service (SaaS) model deployed in public cloud environments. There is an increased need to share relevant, real-time core banking application information at scale with this expanding ecosystem. At the same time, there is also a need to ensure that systems of record, which perform mission critical operations for the bank, are not exposed to unpredictable inquiry traffic.

Core systems for enterprise banking clients often handle significant volumes of transactions and generate a lot of high value data. Using an approach that generates an event for each data level change can cause significant challenges in terms of performance and latency as well as the ability to achieve functional requirements such as sharing composed information.

As a more specific example, one set of information that could be highly valuable for downstream hybrid cloud ecosystem applications is running balances or account balance and summaries from various banking products. Account balances are composed with business logic and not typically represented in copies that are populated from change captured data. Reconstructing this kind of information on the receiving side of a typical CQRS implementation is usually not achievable in real-time, hence the need for an Optimized CQRS approach. The high-level scenario is shown in the Figure 4-6.

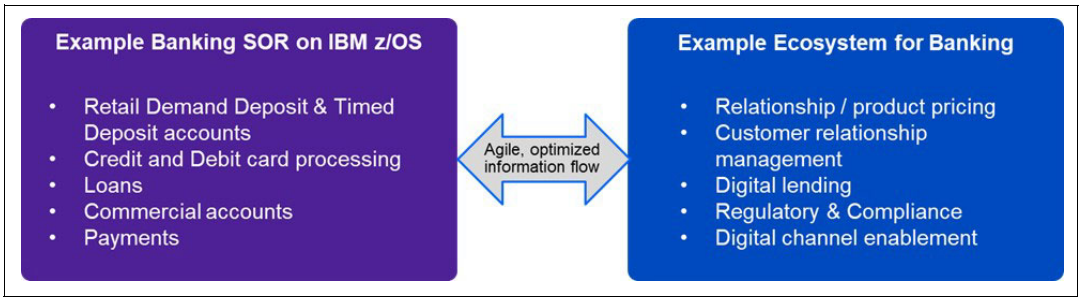


Figure 4-6 High Level Scenario

Typical As-is Architecture

Typical CQRS architectures are often implemented through data level capture of changes from core systems data stores sent as events to streaming architectures such as those built on Kafka. Though leveraging event streaming platforms can be a valuable approach for communication with a broad spectrum of consumers, there are challenges with this methodology if it is not optimized for the event content and volume. A typical as-is structure is shown in Figure 4-7.

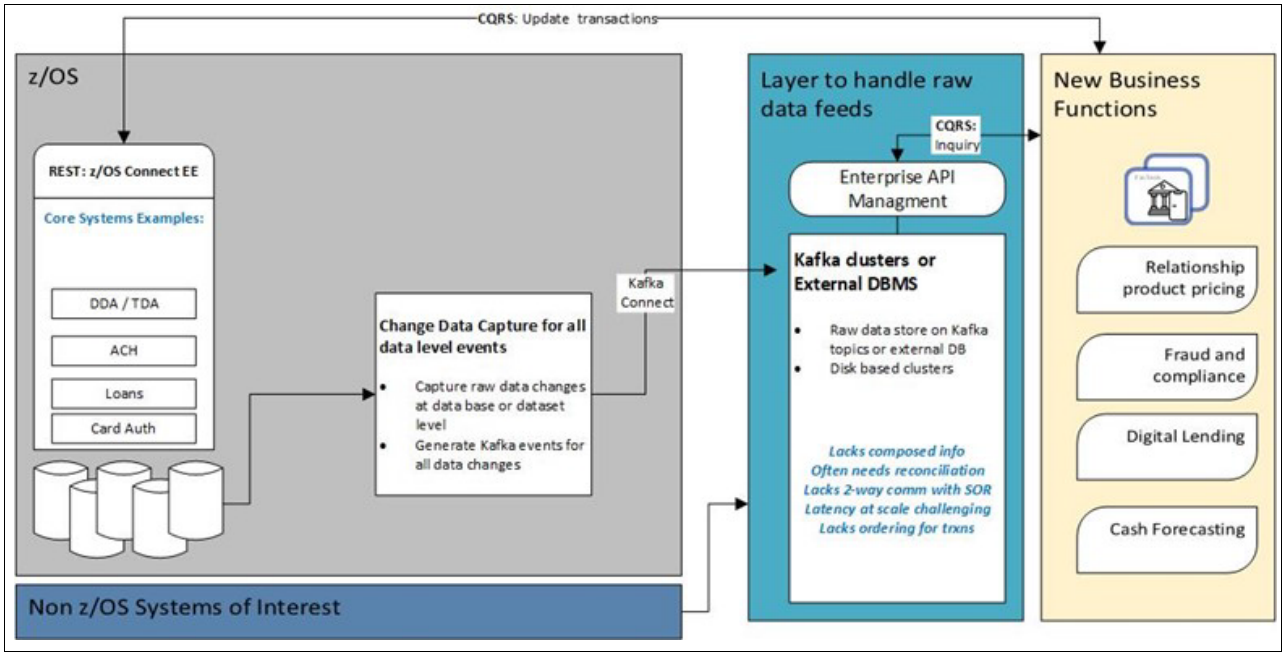


Figure 4-7 As-is Architecture

In this diagram, data associated with core banking applications on z/OS such as demand deposit (DDA), timed deposit (TDA), Automated Clearing House (ACH) payments and others is captured with change data capture technologies and streamed off platform via Kafka topics. Typically, these data feeds then go to a layer that ingests the events (for example, via Kafka based topics), and either populates in a different data store, or is held in the Kafka cluster itself. From this point, the broader ecosystem of SaaS applications shown in the yellow box in Figure 4-7 consumes the data for inquiry purposes. If the SaaS ecosystem applications wish to transact for an update to the core systems, they would communicate directly with the core systems of record application via REST interfaces, for example.

The following are some key challenges with this approach.

- For systems of record that have a high transaction volume, the latency or lack of sufficient currency can pose problems for real-time needs.
- The events are driven from changed data and typically do not contain composed information such as balances.
- When composed information is required, reconciliation techniques are occasionally built into the environment to reconstruct the missing information, however it can be difficult to maintain this reconstructed information accurately.
- Maintaining ordering for information, such as transaction history in this environment, particularly across a multi-partition clustered environment, can be challenging.
- In many cases, the SaaS ecosystem solutions will produce results, recommendations, scores, etc. that should be shared back with core systems. In many cases, clients prefer that communication to be handled in an asynchronous manner for minimal disruption. The as-is CQRS approach is a one-way flow of events from z/OS and there is no efficient mechanism for a two-way asynchronous sharing back of SaaS ecosystem results with the system of record.

Proposed to-be architecture

In the to-be architecture, the typical CQRS pattern is optimized for high volume, transactional information that needs to be shared in real-time. While event streaming capability is leveraged, the content of these events is composed, aggregated, and a subset of information rather than every raw data event. The information to populate the event is derived from selective integration with core systems at the application level, rather than from a data perspective. In addition, there is a mechanism to optimize the handling and sharing of the information via an in-memory set of caches on z/OS to facilitate both low latency needs as well as asynchronous two-way communication. The to-be architecture is depicted in Figure 4-8.

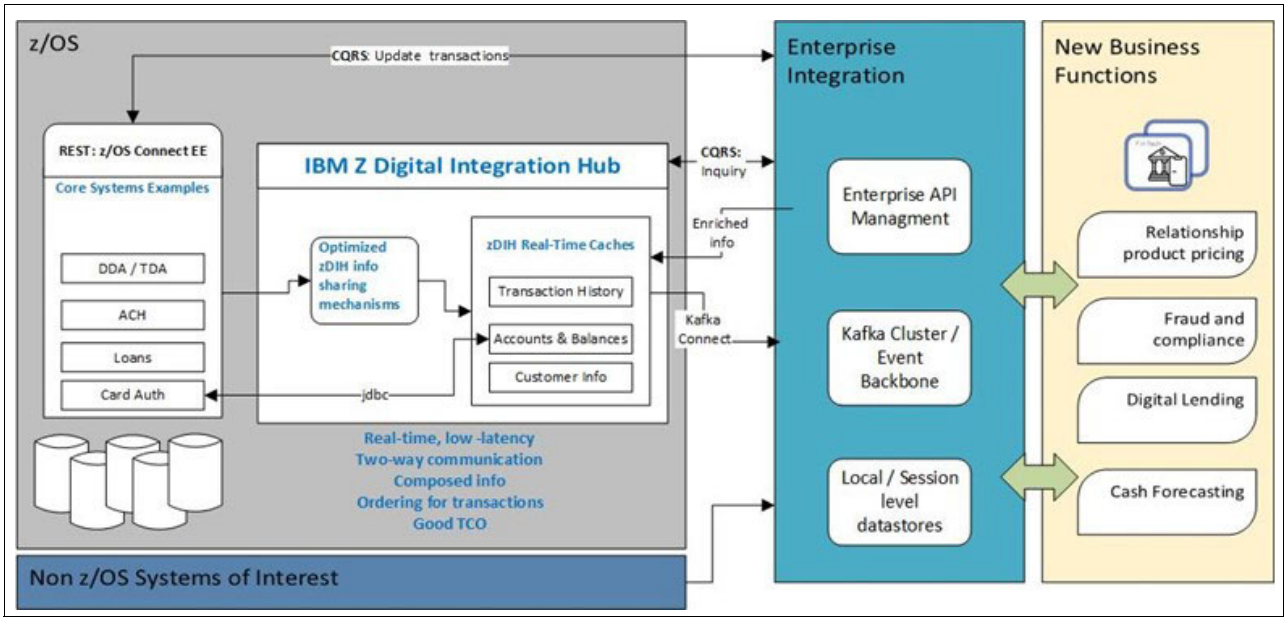


Figure 4-8 To-be Architecture

In this model, the SaaS solutions leverage the 'Query' parts of the architecture via the real-time in memory caches that are on z/OS. These caches are kept current in real-time with only the necessary information to be shared rather than with all the raw data. The 'Update'

part of the flow would still be driven to the core System of Records (SORs) via tools such as z/OS Connect EE for REST based interaction. The architecture facilitates:

- Extremely fast performance with sub-second currency of information at scale.
- Ordered handling for use cases such as transaction history where information from applications that span multiple systems in an IBM Parallel Sysplex® can still be preserved in order on the z/OS cache environment.
- Sharing of selected, composed information such as balances via application-level integration.
- Cost advantages from avoiding re-computing of information if it has not changed as well as from leveraging of z/OS specialty engines (zIIP processors).
- Asynchronous two-way communication where enriched information from the SaaS ecosystem applications can flow efficiently back to SORs.

Implementation

From an implementation perspective, the key component to achieve an optimized CQRS pattern for core systems of record on z/OS is the IBM Z Digital Integration Hub (zDIH). IBM zDIH addresses these needs for clients that run core systems on z/OS through several key features, among which are the following:

- Java-based, fast and flexible in-memory caches and runtime to accelerate compute and query
- The zDIH Developer Kit auto generates java applications which leverage z/OS information sharing mechanisms such as logstreams, create cache structures and keep the caches current in real-time using low code or no code techniques
- A pre-built set of templates for ease of integration with core systems of record on z/OS
- Standards-based interfaces including REST, JDBC and Kafka for event-based architectures

Figure 4-9 shows a technical component overview of zDIH.

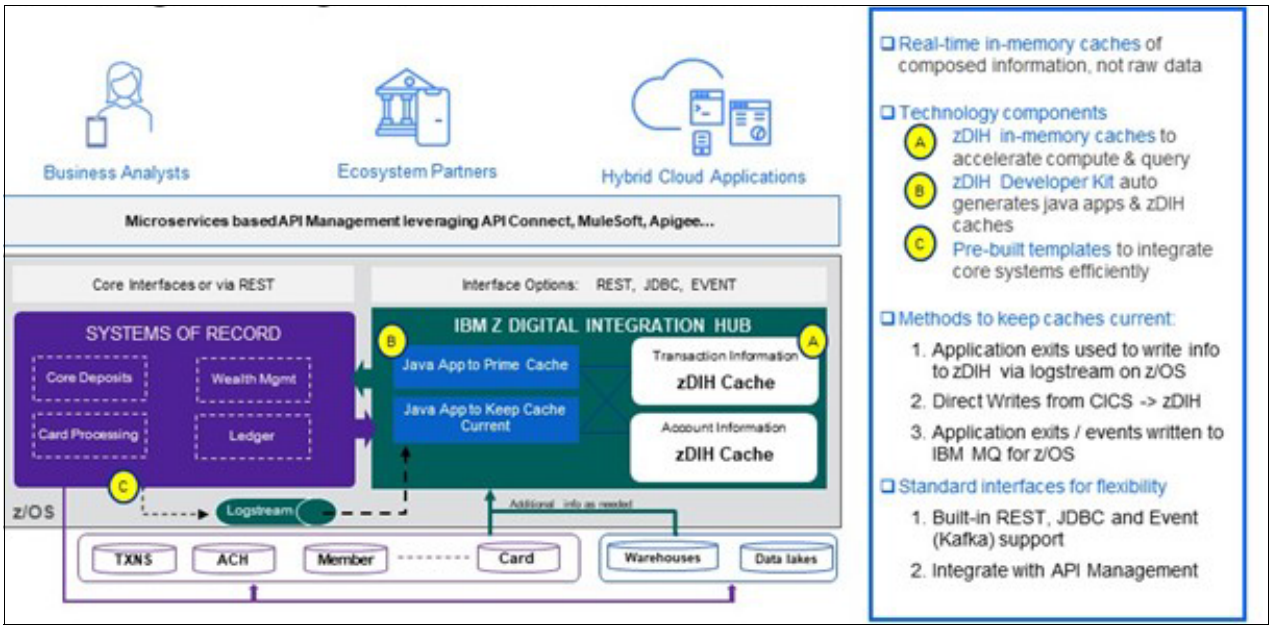


Figure 4-9 Overview of zDIH

IBM zDIH can be used for a variety of use cases in addition to Optimized CQRS and delivers value through its implementation natively on z/OS. Specifically, the zDIH:

- Integrates via application level exits with core systems of record.
- Leverages z/OS capabilities such as logstream for z/OS to share real-time information at scale and in order.
- Uses z/OS functionalities of Parallel Sysplex enabled logstreams for capturing information consistently across multi-system workloads.
- Reduces skills barriers with low-code zDIH applications using zDIH Developer Kit and Java.
- Delivers information in a more consumable manner via a flexible cache infrastructure.

More information about IBM zDIH as well as how you can get started is available at following link: https://www.ibm.com/support/z-content-solutions/z-digital-integration-hub/

4.3 Conclusion

The support of niche cloud technologies such as the Red Hat OpenShift event streaming platform, zDIH in z/OS, with event driven architecture opens up an exciting future. It not only allows businesses to be more competent and prepared in ever-changing competitive markets, but also empowers technical teams to be creative, innovative and focussed on business solutions by providing them richer technical stacks and automating part of their workload. The beauty of having a containerized system is that it provides a “lift and shift” approach. Depending on the use cases, we could shuffle around these containers on-premises, on a private cloud or a public cloud, with the provider of the customer's choice, either using the same provider or a mix of providers, making solutions truly hybrid.

5

Modernize enterprise DevOps

After looking at many different use cases and solution examples for three groups of IBM Z application modernization patterns in this book, we are now exploring in this chapter the Enterprise DevOps Patterns, also known as the Enabler Patterns for the other three pattern groups in the framework. These Enablers define best practices and solution patterns for organizing the development teams' ways of working as well as development infrastructure and tools. In particular, we look at what IBM calls Enterprise DevOps or IBM Z DevOps and its recent journey to the Cloud to support the development of the hybrid applications, comprising of Cloud-native as well as z/OS application components, that were presented as examples in the previous chapters.

In this chapter, we present Z DevOps in three sections:

1. We will walk through important best practices for DevOps and how they apply to hybrid development projects in section 5.1, “Core practices of Z DevOps for hybrid enterprise application development” on page 62.
2. We will explore key tools and technologies that help enable development teams to realize these best practices in section 5.3, “IBM Z Cloud and Modernization Stack: A layered development tool architecture adding incremental capabilities” on page 71.
3. We will discuss an end-to-end example project utilizing these practices and technologies in section 5.4, “A next-generation developer end-to-end development example” on page 89.

5.1 Core practices of Z DevOps for hybrid enterprise application development

When development organizations start with Enterprise Modernization and adopting DevOps, a first thing to review and reform are the development practices performed in the organization. Every development team follows some kind of process. For creating a hybrid development team that can work on hybrid applications, this process or the various processes of the previously siloed teams, needs to be assessed and compared to industry best practices and technical solutions that enable them.

The practices described here have been collected by the IBM Z DevOps offerings group led by Rosalind Radcliffe over the last few years. Additional information can be found in the following:

- DevOps from APIs to z Systems® For Dummies
- Rosalind Radcliffe, Enterprise Bug Busting: From Testing through CI/CD to Deliver Business Results, Accelerated Strategies Press, July 2021
- Enterprise DevOps pattern
- IBM Garage™ Method

They can serve as a great reference for an assessment. They were also the driving force behind many of IBM's development tool offerings such as the products included in the IBM Z Cloud and Modernization Stack that we cover in later sections. These products can directly support and sometimes even enable the adoption of these practices. All of the practices are based on what we perceive as industry standards and can be implemented for enterprise application development as well as hybrid, distributed applications. However, we focus on the specifics of enterprise application development covering specific challenges and how we aim to unify the developer experience between the two.

There is a strong focus on continuous integration and continuous deployment (CI/CD) with the practices covered in this chapter and the tool examples shown mainly focus on those.

5.1.1 Standardize and automate your development setup

Before any CI/CD automation can take place, it is important that all developers in the hybrid team have the same baseline for working on application changes. This relates not only to access to artifacts such as source code and scripts to build and debug the application components that they are working on, it also relates to standardizing the setup and configuration of the actual development environments, for example, the editors and tools they use and how they are configured to ensure common ways of working and optimal collaboration capabilities.

The practices for standardization that we recommend in this book are not along the lines that all developers need to use Mac, Linux, or Windows, or need to use the same editor. These choices are made by each developer based on their skills and technical preferences as these contribute significantly to a developer's job satisfaction.

However, standardization has to happen on the artifact formats produced by these developer environments as they must be uniform and platform independent. For example, if the team is comprised of a mix of Windows and Mac users, then an important standardization and rules to follow would be around eliminating any differences between these platforms, such as the file encoding used for source code files (such as UTF-8 without any special characters allowed outside of strings), as well as the required control characters (such as only LF and

not CRLF line break characters) that must be used in source code files. Tools such as Git can be used to enforce these standards and even perform automatic conversions.

Another example would be to standardize on platform independent tools when developers work with different editors around source code formatting, which is essential for readability of source code files and compare-and-merge operations when bringing the code of multiple developers on the same files together. If every developer formats their code with different rules provided by different editors, then difference-views used for merging would show many changes that are just based on formatting and not actual code changes. A simple thing such as having trailing whitespaces and tabs-vs-spaces can become a huge productivity killer for a development team. An editor independent tool such as Prettier can be used here to format code within many editors or outside of the editor with command line operation. It can even be used in a build pipeline to check for and reject incorrectly formatted code by failing the build. Unfortunately, such solutions are not always available for all languages and technologies and teams need to find the right compromises to ensure such standards in different ways.

Another recommendation is for a hybrid development team to provide a set of recommended well-documented, proven standard configurations for a developer to choose from; at least as a starting point. Here it would be important that such configurations do not enforce any silos, such as a setup for COBOL-only developers, or a setup for Java-only programming, but rather that they all cover all technology platforms so that anyone can bring up a COBOL program although they are mainly working on Java.

To facilitate these reference configurations, ensure that they can be set up with as much automation as possible, because a new developer joining the team might have never worked with some of the technologies and related tools used by the team before; or, a developer is interested in trying a new editor, but would shy away from a long list of installation and configuration instructions.

In the subsequent sections, in which we are exploring the IBM Z Cloud and Modernization Stack as well the End-to-End Next Generation Developer Walkthrough, we will discuss some of such configuration and automation examples for hybrid development teams.

5.1.2 Maintain a single source code management system

This is the core practice for enabling CI/CD in a development organization, especially for hybrid and more complex projects that build applications that are comprised of many components that need to be individually built, tested, and integrated. The overall goal is to have a fully automated pipeline to do all of these steps and the main prerequisite for that is to maintain all of your code, scripts, configuration files and other non-binary assets "as code" and manage it as such a code repository so everything can be retrieved on demand in a consistent way.

As these applications are complex, many humans will be involved working on many different aspects of the application in parallel. All of these changes need to be continuously integrated and tested, so an important requirement is that a common source code management system can enable a consistent baselining, branching, and retrieval of old versions in a consistent way. For a hybrid application, this has to be a technology that can work on all participating platforms and in all development and automation environments. Having a special solution for z/OS will come with significant risk of breaking integrations and impeding parallel development.

One technology that addresses all of these requirements and runs well on z/OS is Git. For more information on Git, see the following website: `https://git-scm.com`

For more information about open-source languages and tools for z/OS, see: https://www.rocketsoftware.com/zos-open-source

Additional information from the z/OS Open Source Tools community can be found at the following website: https://zosopentools.github.io/meta/#/

Surveys, such as from the Stack Overflow web site, show a larger than 90% usage of Git in the industry. The same survey also shows that commercial and open source version control platforms, such as GitLab and Github, are also almost 100% dominated by Git-based technologies.

In the Cloud development space, the 2021 Eclipse Foundation Cloud Developer Survey shows that all the major Cloud-based integrated development environments (IDEs) and IDEs that integrate well with the Cloud are centered around Git-based workflows.

To summarize some important recommendations:

- Use a Git-based software repository system that can be applied in a fully distributed way, with developers cloning Git repositories to their development machines, as well as a centralized management server such as GitLab or GitHub that can be used for build orchestration.
- Store all your source code written in all languages in these repositories.
- Store everything that is needed for testing, building, deploying, and managing your applications as code in your repositories.
- Use clearly defined branching and baselining rules to go from one consistent state of your hybrid application to the next. For more information on the Git branching model and branch based workflows, see the following websites:
  - https://nvie.com/posts/a-successful-git-branching-model
  - https://www.atlassian.com/blog/git/the-essence-of-branch-based-workflows

5.1.3 Incrementally build a fully automated pipeline

Automation is not something that happens overnight for a development organization and projects that try to implement it, but rather something that gradually evolves depending on experience, skills, as well as careful planning and adjusting plans continuously. The end goal is to completely build and deploy an entire hybrid application end-to-end based on any change in any of its components; with the desired optimization to only rebuild the changed parts.

There can be many strategies to achieve this goals. Probably the most popular approach is going bottom-up for automating the builds of various components and gradually adding automation for integrations and deployments. The downside will be that the different components might have evolved out of siloed teams that had chosen all different technologies to accomplish their goals, which might be hard to integrate. A top-down approach assumes that some plan or overall strategy was defined, for example, as part of an enterprise modernization initiative. The downside here is that rarely all automation can be reinvented from scratch and therefore, in most cases, a mix between a bottom-up and top-down implementation will be the result in most modernization projects.

The other complication is that many components of a hybrid application do require different technology stacks to be used for building and have to be integrated. For example, a hybrid application might be made up of the following:

- CICS transaction programs written in COBOL that need to be compiled on z/OS using a build toolkit such as IBM Dependency Based Build (DBB) and unit tested with zUnit

programs that run on the IBM Virtual Test Platform (VTP), and deployed with scripts written for DBB in Groovy or alternatively Urban Code Deploy.

- Java programs running on z/OS, such as an application backend, can be compiled anywhere by using Java automation frameworks such as Gradle or Maven, but if they utilize z/OS capabilities, such as MVS, they need to be tested on the platform, perhaps using a testing framework such as Galasa.
- A Web front-end might be written in Node.js with TypeScript and built using a package manager such as NPM or Yarn that runs unit tests and packages the application.

An integrated build pipeline would perform all these tasks running, perhaps, the build execution on multiple systems, collecting all the results, packaging the application components, publishing them to an artifact repository, baselining the code versions used for the build in Git together with the build's log files, then deploying the application to a staging system and then finally running a set of integration tests against that deployment.

Realizing such an integrated pipeline requires the top-down thinking and the selection of a pipeline orchestration technology that can perform all the various steps described above. In this chapter we will look at a few examples of such technologies, additionally, there are other publications available that explore such technologies in much more detail, among which are the following web sites:

- Develop Mainframe Software with Opensource Source Code Managers and IBM Dependency Based Build
- zAppBuild Introduction and Custom Version Maintenance Strategy
- Integrating IBM z/OS platform in CI/CD pipelines with GitLab

5.1.4 Fully automated tests

Test automation will need to run as part of the automated build pipelines to ensure that changes do not introduce, for example, regressions, and ensure that the application components still perform their functions as intended. Tests need to be performed at many levels of the application ranging from unit tests that test the functionality of solution components in isolation to integration tests that test many parts of the application end-to-end in a real staging or emulated environment with representative data.

The creation and maintenance of tests should not be a dedicated activity performed by specialists, but by the hybrid development team themselves as part of the overall development activities. Depending on the technology stack, different methods for creating and maintaining tests need to be employed. In the distributed development space for languages such as TypeScript, Java, or Python, Test-Driven Development or its more holistic refinement Behavior-Driven Development, are popular methods that focus on writing the tests first as an expression of the requirements collected via user stories. Developers run and rerun these tests while creating the implementation until they pass. The results are a more or less formal specification of the requirements, but the downside could be that you have to continue maintaining a large number of tests slowing down build times with potentially redundancies with tests that are not useful catching regressions. So unit tests needs to be carefully curated and augmented with integration tests.

For impact analysis, IBM has available an application suite called IBM Application Discovery and Delivery Intelligence that can scan your entire application code and artifacts on z/OS and populate databases (relational and graph) that can be used by many tools for analysis. The analysis tools available are made up of a powerful search engine, visual call-graph navigators, business logic visualization and business rule extraction tools, Git history analysis, static-analysis report generators, as well as many more. All these tools allow you

then to rapidly analyze an existing application and trace along control or data flows to assess the impact a potential change will have end-to-end. This information will allow you to define the tests that you need to create to ensure the correctness of the application before and after change.

For test writing, IBM has available a framework called zUnit (originally based on the xUnit framework, but heavily modified since then) that is comprised of a method for test writing and execution as well as supporting tools. These tools can generate test code for existing program code in COBOL or PL/I. These tests are generated as COBOL or PL/I programs themselves and can be executed in the same environments. In combination with the IBM Z Virtual Test Platform these tests can run as unit tests, as well as tests that emulate integration tests by recording and replaying network interaction with external systems such as IBM CICS or the IBM Db2 database. The framework can also be used for full integration tests running against a full infrastructure setup.

Once the tests have been written, you will need to be able to automatically execute them as part of a build pipeline as well as on a component level which is discussed in section 5.1.5, “Every change pushed to the source code management system is automatically built and tested” on page 66.

5.1.5 Every change pushed to the source code management system is automatically built and tested

In the last two sections we focused on best practices for automation and tests that run as part of the automation. In this section, we explore when to actually run the automation. The short answer is: as often as possible. Ideally, the full build pipeline should run as soon as you push a change from your local copy of your Git repository to a Git management system such as GitLab. These systems can trigger build events on a push and integrate with various systems that coordinate the builds. Among examples for such systems are GitLab CI, Jenkins, and Travis.

For large hybrid applications the practical question is how feasible is it to run builds that often and run them in parallel for larger teams working on many changes at the same time. As described in Rosalind Radcliffe’s Enterprise Bug Busting: From Testing through CI/CD to Deliver Business Results, Accelerated Strategies Press, July 2021, the builds have to be designed and optimized to run fast, focusing on giving quick feedback to the developer and the code reviewers. If that is not enough, you can break your builds into parts that build individually to run unit tests and mock integration tests that use technologies such as Mockito in the Java and Spring Boot world or the IBM Virtual Test Platform in the Enterprise Application world. Development teams can then run fast CI builds for their localized components, rapidly reviewing, rapidly fixing (in the case of regression causing builds to break), and merging code changes, which can then be built in larger integration builds. Such builds could be nightly builds that then run the entire test suites including CI/CD integration tests on real staging systems. This approach would require more work on maintaining tests for these different stages of builds.

5.1.6 Clearly define your builds as a consistent set of artifacts

The result of an automated pipeline build should be a set of artifacts that is not only made up of the deployable application with all of its installation and configuration components, but also allows a full trace back to all the sources that were used for the build. This also includes the log files and build artifacts produced with the build. Examples, for such artifacts are as follows:

- Log files for building individual application components as well integration build activities.

- Dependency information linking specific component builds, e.g. when not all components require a re-build then there needs to be trace report of which artifact versions were used from previous builds.
- Quality artifacts such as code coverage reports, static code analysis results, security code scans, test results, etc.
- Versioning baselines from the SCM that define the version used for each and every file used in the build. For Git this could be as simple the as the SHA Commit ID of every Git repository that was used in the build, which includes the Git repositories of dedicated build and integration testing scripts. With these IDs the exact version of all files in the repository can be easily retrieved. In addition to commit IDs some teams use Git Tags as a way to have more expressive baseline names as well as to label special builds such as end-of-iteration builds or release builds.
- Application binaries used and produced by the build.

With this information it is then possible to go back and reproduce any previous build to perform the following tasks:

- Branch off, e.g. reproduce the release build three released versions back to fix a bug in that particular old version of the application that might be deployed at a specific site or customer.
- Compare any old build with another, to answer questions such as why did the application work back then and is now broken: What has changed since then? When was the problem introduced, and did the log files report anything unusual then?
- Perform trend analysis of quality metrics captured by the builds or not; restore the last release build and run a static code analysis against it and compare it against the current state of the code.

Git already provides all the capabilities needed for baselining source code. Moreover, there are many tools that integrate with Git for creating baselines, such as SonarQube. It can store a lot of the quality data types mentioned such as test results, code coverage reports, and static code analysis data in a database that is linked to one or more Git repositories storing these results for every build and branch.

In addition to using the built-in capabilities of CI/CD tools, many organizations create additional Git repositories for baselining build artifacts such as Compliance Evidence Lockers, which are critical to achieve various compliance standards such as Sarbanes-Oxley, ISO-27001, etc.

For more information on baselining build artifacts, see the following websites:

- Compliance in a DevOps Culture: Integrating Compliance Controls and Audit into CI/CD Processes
- Evidence in the IBM Cloud DevSecOps reference architecture for tool chains

The goal here is to provide a full trace of any change to all build and application assets for an audit of all the activities such as scans and tests that have been performed for the change. These evidence lockers are often implemented as a set of versioned reports that are stored as baselined Git artifacts as well as databases or general storage that maintains binary artifacts.

Finally, you need to provide a place in which binary build artifacts such as the compiled and packaged application components can be stored for every build. Dedicated artifact management systems or artifact repositories are a popular choice. There are many commercial offerings available such as JFrog's well-known Artifactory product that you find referenced by many of IBM's DevOps offerings. This solution is attractive for hybrid

development teams as it supports a large set of artifact types from many different technologies that can then be accessed in a technology native way. For example, it provides NPM repositories for storing JavaScript/TypeScript packages the application might use for its front-end, Maven repositories that can be used for Java libraries with Maven or Gradle, as well as generic repositories that be used to store enterprise application artifacts such as compiled COBOL modules and CICS configuration files that can then be directly deployed to z/OS staging and production systems. An example of this can be seen that is using IBM UrbanCode® Deploy is shown on the following website:
https://www.ibm.com/docs/en/urbancode-deploy/7.2.3?topic=integrating-zos-considerations-urbancode-deploy

5.2 The vision for a cloud-native developer experience for z/OS enterprise applications

After listing some core practices for hybrid Z DevOps development teams, we now want to tell the story of how we created the IBM Wazi brand of Z DevOps development tools and solutions driven by these practices.

Wazi was chosen as a brand name for IBM products that embody the Z DevOps vision outlined in this chapter. Wazi is a term in the Swahili language that stands for open and clear, such as in open minded and clear vision, which is what we want to express when thinking about our solutions.

Wazi started out as a project to provide COBOL editing capabilities in a browser and its first technology preview was running an Eclipse Theia editor with a COBOL and Zowe extension on a developer laptop from a local Docker container. This preview evolved into a VS Code extension called IBM Z Open Editor that was published for the first time to the VS Code Marketplace in September 2019. Since then it has been installed over 65,000 times. The editor evolved further by making it available in a Cloud-based editing experience based on the Red Hat CodeReady Workspaces environment. A Debugger has also been added as well as User Build capabilities that use the IBM Dependency Based Build product.

Wazi is now providing three different editing experiences: Wazi for VS Code, Wazi for Dev Spaces (Dev Spaces, being the new name for Red Hat CodeReady Workspaces), and Wazi for Eclipse. The Wazi brand also includes z/OS virtualization solutions such Wazi Sandbox that runs a z/OS emulator on x86 hardware in Red Hat OpenShift, as well as Wazi as a Service (WaaS), which runs z/OS on real hardware in the IBM Cloud as a virtual server instance in a virtual private cloud. In 2021, the Wazi products (except for Wazi as a Service) were repackaged into the IBM Z and Cloud and Modernization Stack that is made up of several other components for Cloud-native development for z/OS that we cover in more detail in this chapter.

We are envisioning the hybrid development team as a group of people without major skill gaps, or skill gaps that can easily be bridged if required, that work with very similar development tools and a common pipeline for hybrid applications that are made up of components running on z/OS as well as distributed/cloud-based applications.

5.2.1 Role of z/OS for hybrid development projects

A key goal for making this vision a reality is to make the mainframe just another node in your pipeline, for example, integrating z/OS just like any other operating system into an organization's hybrid infrastructure. IBM Systems, as an organization, has been working on the following three major guidelines for achieving this goal for many years now:

1. z/OS has been transitioning to remove the differences that serve no purpose. z/OS has a set of qualities of service that provide value, allow you to build applications that take advantage of the reliable hardware, but that does not mean the development and operations processes have to be different.
2. Open source works on and for z/OS. Nothing about building COBOL or PL/I makes it different than building any other language. Strive towards using the same core practices and tools to break down the silos.
3. DevOps defines cultural principles that apply to any system development project and requires breaking down silos. Remove the silos between the mainframe and the rest of the organization by transforming the mainframe development process and tools to today's standards.

Along these lines we envisioned and created solutions along the following two major guiding vision statements:

1. We envisioned z/OS to be a widely available and vibrant choice of platform in a hybrid-cloud development environment.
2. To provide a modern, cloud native developer experience for IBM z/OS that is consistent and familiar to all developers.

5.2.2 Personas of the hybrid development team

Before we go into the details about the architectural decision we made for our Z DevOps pipeline and development tools, let's review the audience, i.e. the personas representing developers of a hybrid development project to guide our decision making, and the assumptions we had about these personas provisioning and using our tools.

By using IBM's variant of the Design Thinking method, we defined several personas (Figure 5-1) for using our tools and solution framework. We will, in this section in particular, focus on Deb, Kathleen, and Todd, and Zach.

Figure 5-1 The Personas for the Cloud-Native Enterprise Application Development Team

Deb's background

Deb is an early tenure z/OS application developer. As a recent graduate entering the workforce today, Deb has limited system programming capabilities and little, if any, exposure to the 3270 terminals (or emulators) used for traditional mainframe application development.

Instead, she learned to develop on a modern integrated development environment (IDE). Although her instructor in school may have suggested a specific IDE to use for her classes, Deb has also has tried out other IDEs and is used to being able to pick her IDE of choice for programming assignments.

In addition to learning how to code, Deb was also introduced to the DevOps and Agile methodologies that many distributed development teams use today. To support these development approaches, she learned best practices such as writing unit tests for her code to make it more robust and easier to maintain going forward. These best practices were facilitated by tools such as test coverage detection provided by her IDE and its integrations, along with the Continuous Integration/Continuous Deployment (CI/CD) pipeline technologies that she used in her projects. As she begins working in z/OS application development, Deb would like to continue working in this way using the tools and best practices she is familiar with.

Kathleen's background

Kathleen is an experienced z/OS application developer. Although she has spent the majority of her long career coding on the traditional 3270 interface using a Waterfall development methodology, she is aware of the need to stay current with the industry's evolving technology environment. Thus, she is open to trying new and innovative ideas and technologies not only to continue learning and improving as a developer, but also to modernize her team's z/OS application development processes so that they can continue delivering quality products on time in an increasingly fast-paced world.

Kathleen sees the potential of Z DevOps and its associated tools for enriching her team's development process with additional coding, debug, testing, and analysis capabilities. At the same time, she notes that the automation capabilities of a CI/CD pipeline can streamline traditionally manual processes such as backup and deployment, making them more efficient and less error-prone. As developers new to z/OS are becoming more common in the workforce, Kathleen is interested in adopting these modern IBM Z development approaches and tools that will appeal to newer developers with their familiarity and facilitate the onboarding process for new team members.

Todd's background

Todd is a Cloud operations administrator. He is a Cloud DevOps development expert, with an emphasis on infrastructure, deployment, and operations. He leverages this along with his expertise in Red Hat OpenShift to help hybrid development teams set up their development infrastructure in the Cloud and in on-premises OpenShift clusters. Todd contributes to the development and operation of the overall cloud toolchain and pipeline, and aims to keep the cloud platform maintained and updated so that it runs as efficiently as possible.

Earlier in his career, Todd worked on projects that deployed and ran Kubernetes applications written with various technologies such as Java and Node.js. He also has a background in writing automation for Kubernetes using Helm and Operators written in Go and more recently, Ansible. Although Todd did not have any exposure to mainframe applications in the past, he understands the role of the mainframe in hybrid application architecture. While working on cloud development infrastructure, Todd expects that he can use the mainframe in the same way as any other compute node in his infrastructure setup, rather than having to learn z/OS specifics. He expects services such as Unix and SSH to be available for him in the same way as on any other device.

Zach's background

Zach is a system programmer and z/OS expert with many years of experience. He is responsible for the configuration of many z/OS systems in his organization, and his career experience has primarily been as a traditional z/OS developer using green screen only. Although he strongly believes in z/OS as the best and most robust platform for security, reliability, and availability, he also feels the need to modernize in order to adapt and remain relevant in today's fast-paced technical industry. Thus, one of his goals is to automate processes involved in managing and maintaining z/OS systems and resources, without impacting current day-to-day system operations.

As Zach helps many different teams, he has limited time to work with Kathleen, Deb, and Todd. They therefore try to be as independent from Zach's services as they can. To assist with this, Zach creates standard z/OS development configurations that Deb's team can deploy themselves as virtual server instances and use with minimal setup.

Assumptions for the hybrid cloud-native development team

For this chapter we assume development teams comprised of representatives of the three main personas, as well as limited involvement from Zach, working on a hybrid application that matches one or more of the architectural patterns described in previous chapters. The application under development comprises of code in several languages such as COBOL code running on z/OS, access via REST APIs from OpenShift applications that implement their backend perhaps in a language such as Java, and front-end applications perhaps in TypeScript and Node.js.

The team has decided that all developers should have full visibility into all the components of the application, because they want to work as one development squad using a common agile, rapid-iteration development process for working on all application components together. This means, that although there will be clear experts and owners of specific application components that match their backgrounds, that all team members should be able to understand how the application works in general, as well as every team member should be able to pick up work for any of those components. For example, Deb should be able to Debug and research a defect in the COBOL code and Kathleen should be able to implement changes in the Java code running on OpenShift.

Therefore the team will follow many of the practices defined above to standardize on a set of editors (Deb uses VS Code with Z Open Editor and Kathleen uses Wazi for Dev Spaces), code encoding, conventions, code formatters, etc. They create a fully integrated pipeline that builds all application components owned overall by Todd, but with Deb and Kathleen writing code using test-driven development methods as well as all the build scripts and integration tests.

In section 5.4, “A next-generation developer end-to-end development example” on page 89, we will pick up this scenario again, but keep these personas and their backgrounds in mind when we first walk you through the technical Z DevOps architecture.

5.3 IBM Z Cloud and Modernization Stack: A layered development tool architecture adding incremental capabilities

In this section we want to explore the overall architecture of the IBM Z Cloud and Modernization Stack and in particular its IBM Wazi components. As the name implies with the word Stack, it is a layered set of capabilities that provide the following:

- Give developers and their organizations options by providing layers with alternative implementations using alternative technologies that best match their requirements or reuse the technologies already deployed and in use. Example: use z/OSMF or alternatively Remote System Explorer to access z/OS resources.
- Provide capabilities with an increasing level of deployment and usage complexity to provide choices for the level of tool support they want to implement in their organization; or want to implement incrementally. Examples for such layers could be (a) only CLI operations to access z/OS resources from your development machine, (b) use a graphical navigator in a local VS Code or Eclipse editor, or (c) access z/OS resources from the Cloud.
- Support different ways of working, based on organizational policies, that either centralize development resource management or fully empower developers to provision their own development environments. Example: developers can only use the z/OS system provided to them by their System Programmers vs developers can deploy and configure their own z/OS systems as virtualized Sandboxes that get deployed via Red Hat OpenShift which serves as a central command center with automation for provisioning and configuring such systems. Another example is development teams implement full end-to-end developer responsibilities by managing all build scripts with the source code, maintained by the developers versus all development scripts are maintained by a central team of build engineers that do not allow developers to access and change the build process.

The following sections are organized to describe the layers from the bottom up, starting with the lowest level of abstraction of z/OS access and development capabilities and progressing to increasingly more sophisticated and complex capabilities. Each layer is also representative of a core architectural principle that we have established for Z DevOps that enables these layers.

5.3.1 Layer 1: Establishing connectivity to z/OS

A development environment for enterprise applications requires access to z/OS. As IBM does not produce any physical 3270 terminals anymore, even access via a green screen needs to be based on an established protocol such as 3270. However, for creating common tools for the hybrid development team that wants to access z/OS in the same way they would access a Linux server running a Java backend server, we want to utilize protocols that allow such similar behavior. There are quite a few to choose from, among which are the following:

- FTP (file transfer protocol): as the name implies the protocol is focused on getting files on and off of z/OS. Through extension the z/OS FTP server supports access to subsystems such as USS, MVS, and JES.
- SSH (secure shell): z/OS has an SSH implementation based on OpenSSH that provides encrypted login and command execution facilities, as well as file transfer access to z/OS; effectively providing a more secure FTP implementation. Plus, it provides login shells for interactive command execution as well as for running scripts.
- z/OSMF (z/OS Management Facility): provides system management functions in a task-oriented, web browser-based user interface with integrated user assistance. For many of its capabilities it provides REST APIs that can be programmatically used. Among its many APIs are methods for file transfer, execution of commands and jobs, querying status such as for JES and many more. All REST services are fully documented and are discoverable with Open API/Swagger interfaces.
- RSE API (Remote Systems Explorer REST API): provides capabilities similar to z/OSMF that were introduced with IBM's z/OS Explorer and IBM Developer for z/OS development tools. Whereas z/OSMF focuses on systems management, these services are optimized for development activities and maintained by the same group that develops the Z DevOps

development tools. The REST APIs are also discoverable with Open API/Swagger, plus there are NodeJS and Java libraries available to use them from development tools implementing these languages.

Development teams can utilize any of these technologies or combinations of them to access z/OS directly, e.g. using SSH/SFTP to download and upload program files for maintenance updates, as well as provide connectivity to z/OS for development environments. Teams can create their own tools that utilize these well-documented capabilities or use the ones covered in the following sections describing tool sets that were built with these.

5.3.2 Layer 2: Building a foundational layer with client SDKs, open APIs, and command line interfaces

Instead of asking developers to use the connectivity services directly, IBM and its business partners have invested in additional layers of abstraction that make the creation and maintenance of a development tool much easier. The design principles and how they are applied as capabilities of z/OSMF are really straight forward and are as follows:

- Provide an Application Programming Interface (API) for every new capability.

  z/OSMF not only provides a proprietary web user interface to do that, but also a REST API that is fully documented.

- Follow standards and industry best practices for the APIs specification.

  In addition to the documentation in the IBM Knowledge Center there is also a machine readable specification provided in the industry standard Open API that is accessible through a Swagger interface.

- Create Software Development Kits (SDKs) to utilize these capabilities in many different ways.

  The Zowe open-source project created several SDKs that support various programming languages to develop applications that abstract from the z/OSMF REST APIs to easily create applications that utilize it. SDKs are available for Node, Java, Python and several other programming languages.

- Implement alternative user experiences with these SDKs

  The Zowe SDKs have been used for creating several user experiences for interacting with z/OS and MVS Data Sets in particular. The most well-known is the Zowe CLI, which provides a command-line interface that allows users create a data set with a simple command such as **`zowe zos-files create data-set NEW.DATASET --like EXISTING.DATASET`**, and Zowe Explorer that provides a graphical tree-browser in Microsoft's VS Code editor that allows users to search and display data sets as well as create new data sets with simple right-click operations, such as right-clicking an existing data set and performing an **`Allocate Like`** operation similar to the command line operation.

This abstraction layer of providing SDKs can now be combined with the previous layer of providing alternatives to so-called data provider technologies, because the SDKs listed above do not only support z/OSMF, but other protocols such as FTP as well. Vendors such as IBM also provide extensions to these SDKs called plugins for their own technologies such as the Remote Systems Explorer (RSE API). The ability to create such plugins for extensibility is another core principle that the design of the SDKs is following. The Zowe project often makes the extensibility of its capabilities a core feature. See VS Code extensions for Zowe Explorer as an example of the APIs provided by the Zowe Explorer project for contributing new data providers. The RSE API data provider, in particular, was designed to plugin into Zowe CLI as well as Zowe Explorer to provide end-users with the choices for the user experience that were mentioned at the beginning of this section.

5.3.3 Layer 3: Standardizing on next generation editors and modern languages capabilities

We have discussed flexible architecture for Z DevOps that gives developers and their organizations choices for alternative technologies. We used the example of command-line interface versus graphical editor interface to show how the same capabilities can be provided as different user experiences.

To continue this story, retrieving a COBOL program from MVS on a z/OS system, making changes to it in an editor, then re-uploading it to z/OS for compiling and running the program can therefore be envisioned in either of these scenarios. Deb the developer (the persona introduced in section 5.2.2, “Personas of the hybrid development team” on page 69 as a next generation developer without prior enterprise development experience) could do the following with a command line editor:

1. Use the Zowe CLI to download a COBOL program from MVS with a simple command that retrieves a file by name and converts it to UTF-8 to be placed on her development machine.
2. Use any editor of her choice such a VIM to edit the program.
3. Use another Zowe CLI command to upload the file converting it back to EBCDIC.
4. Use a third Zowe CLI command to submit a JCL that compiles, links and tests the programs.

Deb could also use a more integrated editor such as Z Open Editor and Zowe Explorer that run inside Microsoft VS Code to do the following:

1. Use Zowe Explorer's DATA SETS view to search for the data set with her COBOL programs.
2. Open the file converted by Zowe Explorer to UTF-8 in Z Open Editor's COBOL editor by just a mouse-click on the data set member name and make changes in the editor.
3. Just use the Save menu or shortcut for the updated program to be converted and written back to MVS.
4. Either find JCL in MVS data sets in the same way as she found her program files and do a simple right-click to submit it or, alternatively, use the Z Open Editor user build function to upload, build, and run the program via a right-click in the editor.

Either the command line editor or a more integrated editor are both completely valid alternatives for editing a COBOL program. One option requires more tools to be installed and configured on the developer's machine than the other. One option provides more integrated capabilities than the other, such as the language support of Z Open Editor performing instant syntax checking and advanced editing capabilities such as code completion, code formatting, and navigation along an outline view as well as code references.

However, both scenarios are not really mutually exclusive alternatives, but rather just specific choices a developer can make to support their preferred way of working. The tools mentioned in these examples can be configured in a way that the scenario can become more and more interchangeable. For example, VS Code has an integrated terminal and Deb could run any Zowe CLI command directly from VS Code as well. Or, if the COBOL editing capability was implemented with the right abstractions, then it is feasible that the (almost) exact same user experience for COBOL language features can be available in both editors, but presented in each editor's specific ways.

Such an abstraction exists. It is called the Language Server Protocol (LSP), and is another open API that was created to support interchangeable layers of capabilities as well as

capabilities that can be combined with one another in a plug-and-play kind of way. The key for the success of such APIs is that they need to be popular enough with enough implementations for vendors to provide these choices. In the case of the language server protocol, implementations are not only available for a very large number of programming languages, but also many different and diverse editors including VIM as well as various Eclipse-based editors such as the Eclipse Java SDK and Eclipse Theia. Many of these editors also provide SDKs for anyone to build and integrate language servers.

For a vendor such as IBM who has invested in its language parsers for COBOL and PL/I for over twenty years as part of its IBM Developer for z/OS offering, an API such as Language Server Protocol is ideal because, with a little bit of refactoring, these capabilities can now be offered as part of a language server in other editors such as Microsoft VS Code and Eclipse Theia. On the other hand, language capabilities that were newly created for the Z Open Editor for VS Code, such as Z Open Editor's language server for the REXX language, were also immediately adopted and made available for IBM Developer for z/OS.

In conclusion, when it comes to development organizations giving choices to their developers, a family of editors that utilize the same language server implementations for specific languages will be easier to support and maintain, as well as allow developers to very easily switch between editors, even editors that all behave differently and offer fundamentally different features.

5.3.4 Layer 4: Adding pluggable extensions with specialized capabilities: z/OS access, debug, build, CICS, Db2

In the previous sub-section, we talked about the language server protocol API providing pluggable language capabilities to editors. In addition to the programming languages support, development teams also need integrations with other development tools and connectivity to platforms. These should be integrated with similar APIs to allow supporting alternatives as well.

Continuing the example from Layer 1, different development organizations will want to standardize on different protocols on that list. If an organization used the Eclipse-based z/OS Explorer or IBM Developer for z/OS before, then they have already deployed the Remote Systems Explorer (RSE) servers. If they now want to also offer Zowe CLI and Zowe Explorer for VS Code to their developers as an alternative choice, then it would be likely that they want to utilize the same technology to avoid double-maintenance. Another organization that is not using RSE might prefer using z/OSMF as it is part of the z/OS operation system already.

When designing connectivity tools such as Zowe CLI and Zowe Explorer, which are a key underlying technology of the Wazi offerings, it was important to provide APIs for supporting multiple protocols. Zowe CLI provides the concept of a CLI Plugin that allows users to install additional capabilities. Such new capabilities could be the implementation of commands that interact with z/OS using a different protocol. As Zowe CLI provides z/OSMF out of the box, IBM was able to create a plugin for it that would allow executing all its commands using RSE API instead of z/OSMF and SSH.

Zowe CLI and its plugins not only provide command line operations, but they can also be used as an SDK for other Node.js applications to utilize the implementation of the command line operations. The graphical Zowe Explorer that adds visual tree views to VS Code showing hierarchies of MVS data sets, USS folders, and JES jobs have been created with these Zowe CLI SDKs. When using the graphical tree view to, for example, list all the data sets for a particular search pattern, the Zowe Explorer will just call the respective API method provided by the Zowe CLI SDK.

The RSE API plugin for Zowe CLI provides API calls to execute the exact same commands against RSE API, i.e. it implements the same SDK method for listing data sets for a search pattern, mentioned above. Zowe Explorer itself was designed to allow extending its tree views to utilize alternative plugins using a simple Adapter Design Pattern[1]. So the combination of APIs from Zowe Explorer and Zowe CLI allowed a completely transparent extension of the Zowe Explorer with the RSE API protocol as an alternative to z/OSMF. The same thing was accomplished with the FTP protocol, utilizing the Zowe CLI plugin for FTP with the Zowe Explorer extensibility APIs.

Figure 5-2 shows the relationships of the various extensibility APIs and their implementations that have been utilized by IBM Z Open Editor and Zowe Explorer.

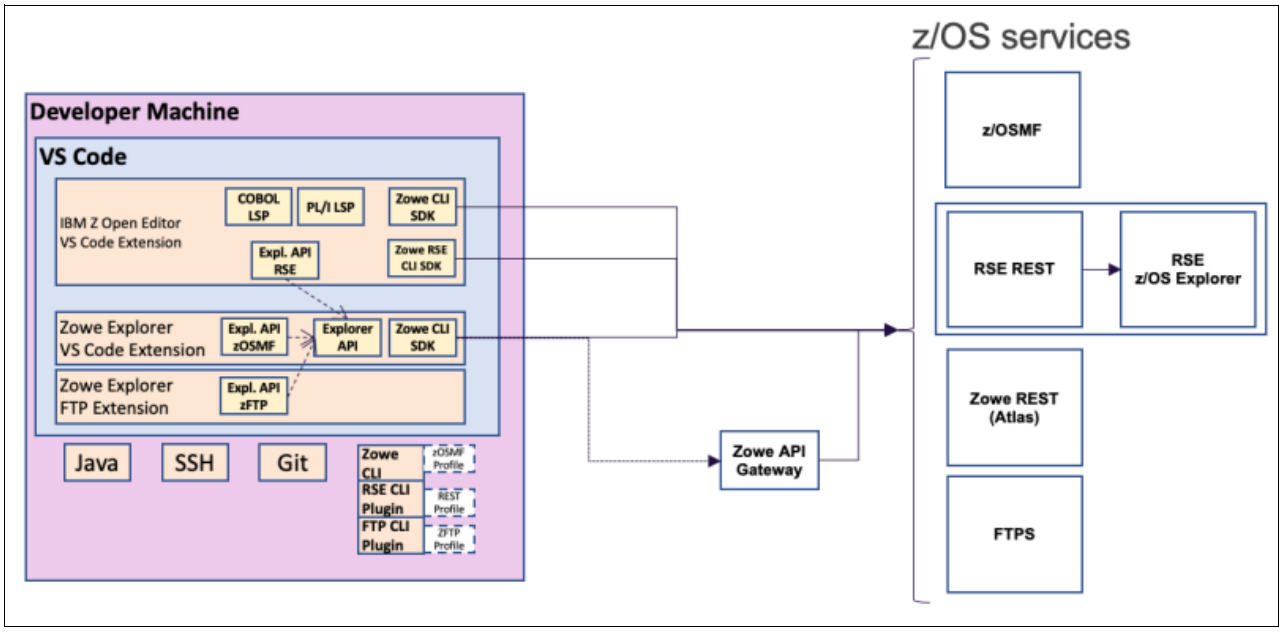


Figure 5-2 An overview of various extensibility use cases for Z Open Editor and Zowe Explorer.

On the left side of Figure 5-2, you can see three extensions for VS Code: Z Open Editor, Zowe Explorer, and Zowe Explorer FTP Extensions. In the middle row of these extensions, the Zowe Explorer extension provides an Explorer API that allows other VS Code extensions to register additional capabilities for the Zowe Explorer. IBM Z Open Editor does that by providing an implementation for the RSE API that then utilizes the RSE API CLI SDK. That RSE API CLI SDK is packaged with Z Open Editor and fully reused from the Zowe CLI plugin for RSE API. It allows using Zowe Explorer with RSE API capabilities independent from the Zowe CLI plugin being locally installed as well or not. Figure 5-2 illustrates that by showing various CLI plugins at the bottom that get directly installed on the development machine (as global npm packages). You see that the FTP Zowe Explorer extension does the same thing by implementing the Zowe Explorer API for the FTP protocol. When configured this way with these extending VS Code extensions, Zowe Explorer can then be used with any of the protocols that can interact with z/OS.

In addition to having VS Code extensions for alternative z/OS interaction protocols, a development team also wants to select extensions that provide specialized capabilities for other technologies used in the project. The following is a selection of extensions that we recommend for an IBM Wazi setup.

[1] Gamma, Helm, Johnson, Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley, 1994

- IBM Z Open Debug: provides a Debugger UI in VS Code for COBOL, PL/I, and High-Level Assembler. It is part of the IBM Z Cloud and Modernization Stack as well.
- Another open-source extension that utilizes the extensibility APIs of the Zowe Explorer is the Zowe Explorer for IBM CICS VS Code extension that adds CICS capabilities that allow interactions with CICS regions and programs and the ability to run commands against them.
- IBM Db2 for z/OS Developer Extension can be used for writing, analyzing, and debugging SQL queries and stored procedures.
- VS Code has only basic support for Git. The GitLens extension is used by millions and has many features, replacing the need for a command line almost entirely.
- IBM Z Open Editor uses YAML for configuration files. Red Hat has a YAML extension that reads the JSON/YAML schema and provides code completion and validation.
- If you are doing automation with Ansible and IBM's Red Hat Ansible Certified Content for IBM Z, which we use later in this chapter, then you will want the Red Hat Ansible VS Code extension, which provides code completion and validation.

5.3.5 Layer 5: Adopting containerization for deploying development tools with Red Hat OpenShift and Dev Spaces

In this chapter so far, we have showed examples of many technologies that can be utilized by a hybrid development team. The focus has been mainly on tools for Enterprise Applications, such as the COBOL and PL/I applications, that run on z/OS. For a hybrid project that mixes multiple technology stacks there will be many more tools that teams can choose from.

A key reason for the IBM Wazi tool set to be centered around VS Code is exactly to avoid that developers have to change tools for different technology combinations. We wanted to support the most commonly used environments, with the most flexible extensibility, as well as the largest portfolio of extensions available for hybrid development projects. Currently, these requirements are met by VS Code. Stack Overflow surveys show a more than 80% adoption rate of VS Code. Its extensibility API has been reimplemented even by other editors, including the many implementations of the language server protocol. The VS Code Marketplace provides tens of thousands of extensions that have been built with this API. The most popular extensions have tens of millions of users; for example, the Python VS Code extension has 65 million users. The IBM Z Open Editor has been installed at the time of this writing more than 65,000 times. In our own telemetry that we receive from our editor installations, in which users kept telemetry enabled, we can see more than 500 unique users every day.

So we believe for hybrid development teams staffed with personas like Deb, VS Code is the currently ideal choice of editor for Wazi as these developers probably used the tool and many of its extensions already for developing in many other programming languages for various technologies. However, what about the Kathleen persona? VS Code might be new to her as she worked on a traditional green screen emulator and perhaps a more integrated IDE such as Eclipse and IBM Developer for z/OS that come prepackaged with more capabilities.

In Figure 5-3, we list some of the steps that Kathleen would have to perform for setting up the prerequisites, tools, and extensions.

- install a package manager (e.g. MacOS: brew, Windows: choco)
- install Git
- install Java (at least 8; also install 11, but not as default)
- install node 12 (not newer)
- install Zowe CLI via npm
- install Zowe CLI plugins via npm
- install VS Code
- install Zowe Explorer extension
- install Z Open Editor extension
- install other extensions, such as GitLens, YAML, etc.
- set all kinds of VS Code settings for your accounts
- create Zowe CLI profiles for your z/OS hosts and accounts
- clone and open your repo

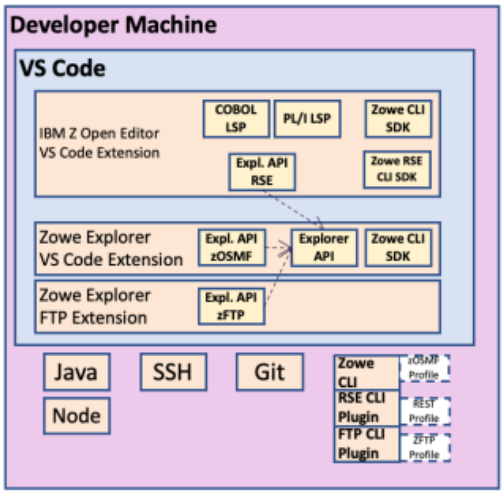


Figure 5-3 Steps a developer must perform for setting up Z Open Editor and Zowe Explorer.

Many development teams manage a wiki (or online web page) that documents all the setup steps required for a developer joining a project. For a developer experienced with these tools, such as Deb, these are straight-forward. However, even for her there are potential points of errors. For example, if a particular version of a language runtime such as Node or Java is required it could happen, particularly for developers that work on more than one project, that the wrong version is being used, leading to an accidental use of incompatible SDKs, build errors, etc.

As another example, a popular technology such as Ansible is built on many layers of other technologies such as Python and SSH and requires that you use the correct version combinations of these on the client as well as the host. In particular when it comes to using Ansible on z/OS, there are more potential incompatibilities to consider with specific versions of the IBM Z Open Automation Utilities and Python for z/OS. What makes it even more complicated is that the Red Hat Ansible VS Code extension also requires special versions of Ansible and the Ansible Linter to be installed on the client.

The extension depends on Python and the YAML VS Code extension and not every version of these extensions works with every other version. Especially if some team members choose to use a different VS Code "compatible" editor such as Eclipse Theia, they would not be able to use the latest of each version available as Theia's implementation of the VS Code API is not up to date. For Windows users, the setup is even more complicated as Ansible does not support Windows as a client (for example, the control node) and users need to install a Linux virtualization such as Windows Subsystem for Linux (WSL).

From these examples, you can see that maintaining a developer machine requires a lot experience with these technologies and very precise documentation by the team leads for which tools to install and maintain on their machines. A better solution would be to automate the installation and configuration of these development environments to enable all developers with exactly the same setup, or at least a setup with the mandatory prerequisites that can then be individually refined by each developer with the tools they can choose from and prefer to use. Such automation would have to be provided on several layers of the setup, from the recommended operating system, to the compilers and runtime environments, to build and automation tools.

For DevOps, this is a common problem with common solutions in the deployment space. Here containerization has been used for many years to ensure that applications get automatically deployed, potentially in many distributed locations, into the correctly configured environments with the exact same prerequisites. The sample principles can be applied for development environments as well, by containerizing the development setup prerequisites, mapping them to editor configurations, and deploying them into the Cloud, potentially hundreds of times, giving each developer a sample development baseline. There is an open-source project that provides exactly such a solution called Eclipse Che, shown in Figure 5-4.

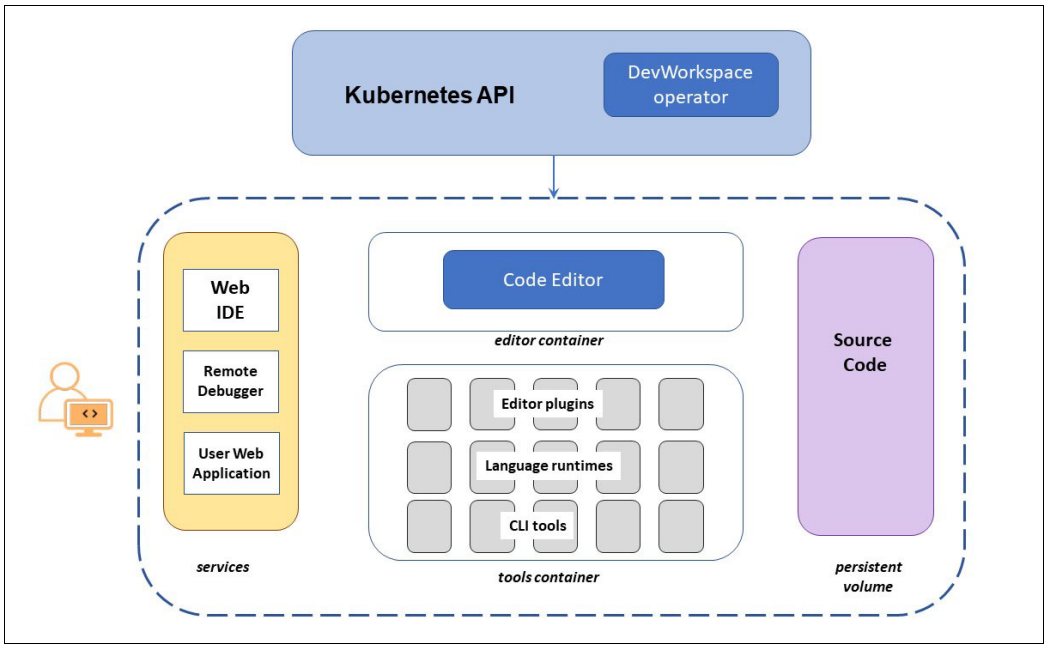


Figure 5-4 Overview of a developer's personal Eclipse Che workspace

In Eclipse Che, development teams can choose from a large stack of technology configurations provided as container images based on open-source Docker files and an open specification framework called *devfiles* that allow specifying development stacks as combinations of multiple images, VS Code extensions, storage volumes, settings, environment variables, command short cuts, Git repositories, and many more. A developer then uses a Web-based dashboard to simply select a predefined stack from Che to create a personal workspace, which runs on Kubernetes in a personal namespace as shown in Figure 5-4.

Based on the devfiles, images get instantiated into containers and VS Code extensions get loaded into a Web-based editor. The developer does not have to install anything themselves as everything can be predefined. The stacks that are provided by the Che open-source project are examples and starters that development teams can use to customize for their specific needs. In other words, a project's development lead can prepare the perfect development stack for the project and publish it in the Kubernetes-based Che system. Then every other developer can come in and simply instantiate the stack as their personal workspace with a single mouse-click. All the required prerequisites and VS Code extensions will be available with the right versions in the containers and editor as prepared by the lead. Developers can still customize their personal workspace by copying and modifying the devfile and even add their own images for their personal containers with special tools and VS Code extensions that they want to use in addition to the prepared stack. As devfiles can be maintained with the project's source code in Git, the team can continuously evolve the *devfile* and agree on new tools to be used by the team. Similarly the Dockerfiles defining the

container images can be managed by the team and republished to an image registry used by the team. Eclipse Che provides a Kubernetes operator to manage one or many installations with different configurations for different teams of an organization if needed.

As mentioned, Eclipse Che provides a large set of images and devfiles as open-source, donated by may different developers and organizations. Red Hat took a major subset of these and curated them in a similar fashion as they curate their Linux distributions. With an emphasis on stability, reliability, and security, they maintain each image by basing them all on the Red Hat Universal Base Image and certifying each image. Red Hat then publishes this variant of Eclipse Che as a branded distribution called Red Hat OpenShift Dev Spaces, formerly known as Red Hat CodeReady Workspaces that runs, as the name implies, exclusively of Red Hat OpenShift. All the images' dockerfiles and devfiles are still available as open-source under an EPL license so that customers can still take them as starting points for customization, but they need to license OpenShift and the Universal Base Images.

For Enterprise and Hybrid Application development support IBM then takes Red Hat's distribution of Red Hat OpenShift Dev Spaces and customizes it even further with its IBM Wazi for Dev Spaces offering. We add capabilities and integrations for z/OS development by providing an additional set of custom images and devfiles on top of what Red Hat already provides. The images are also fully IBM Cloud certified which adds additional requirements to the Red Hat certification.

Figure 5-5 shows an overview to the Wazi solution stack.

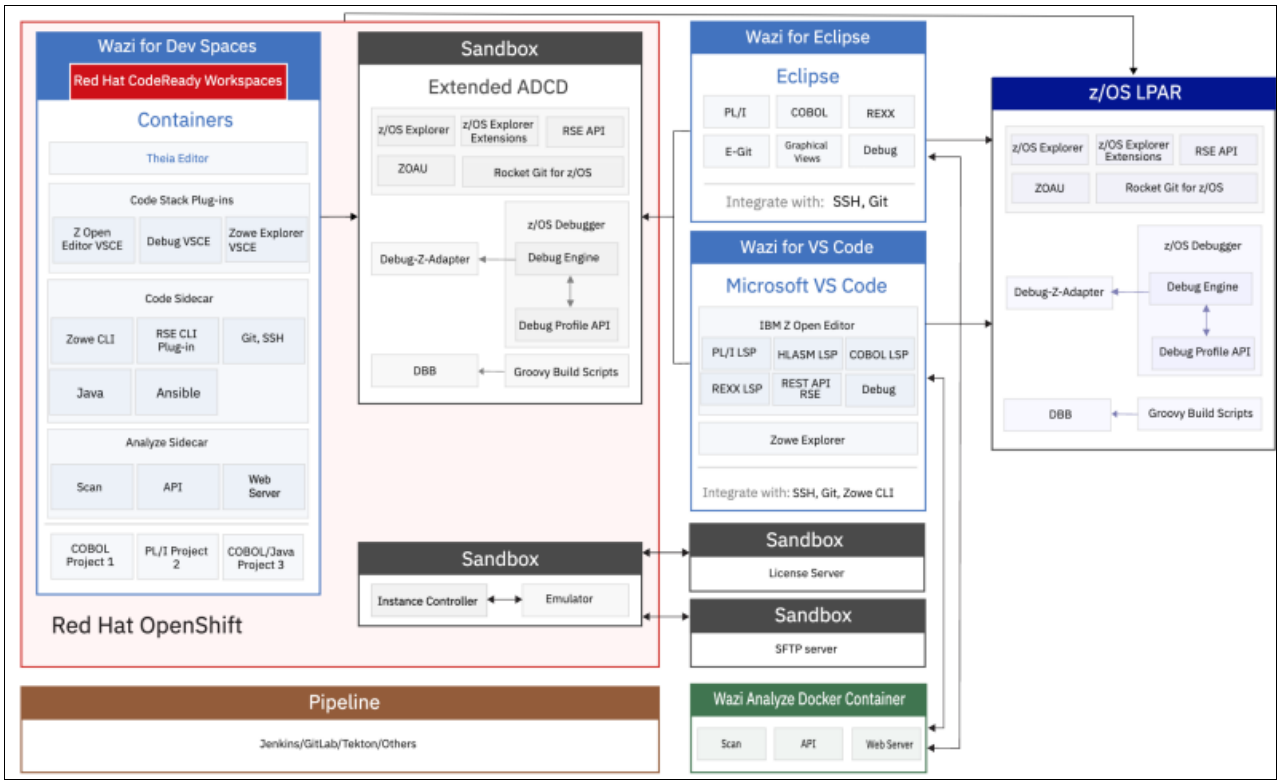


Figure 5-5 IBM Wazi for Dev Spaces and its integrations.

Wazi for Dev Spaces is shown on the left inside the Red Hat OpenShift box showing some of its runtime components such as two *sidecar* containers that provide special development configurations for z/OS developers. Code Sidecar provides essential tools for interacting with z/OS such as the Zowe CLI and the RSE API Plugin for Zowe CLI that were discussed in

section 5.3.2, “Layer 2: Building a foundational layer with client SDKs, open APIs, and command line interfaces” on page 73.

Together with the Zowe Explorer VS Code extension (Zowe Explorer VSCE) listed in the Plugins box in Figure 5-5 they provide tools to access z/OS systems from the editor. Another important set of productivity tools pre-configured in the Wazi stack are tools for Ansible and IBM's Ansible collections for z/OS, which are installed on that image together with the Ansible runtime and its prerequisites such as Python, as well as editing tools such as the Ansible Linter and the VS Code extension for Ansible. Developers can now immediately start working on Ansible playbooks in the editor getting language support through syntax checking, code completion, and documentation integration and run the playbooks against z/OS directly from the editor's integrated terminal window. This chapter contains a tutorial for this.

Another sidecar that is available for Wazi for Dev Spaces is the Analyze Sidecar, which provides static code analysis of your applications written in COBOL, PL/I, Assembler, JCL, or Java. You can run an analysis scan of the code that you are currently editing and review the results in a Web-based graph-browser presentation directly from the Wazi for Dev Spaces editor. We will explore scenarios for Analyze also further below.

As shown in Figure 5-5, Wazi also provides alternative choices. Although the Wazi for Dev Spaces solution provides all these preinstalled configurations some developers might prefer to be more in control and work locally. They can use the alternative VS Code or Eclipse-based editors of the Wazi package as well. Finally, Wazi comes with the Wazi Sandbox that provides developers with their own personalized z/OS environment that is emulated on x86 hardware and deployed and managed from Red Hat OpenShift, which we explore further in this chapter.

Note: To try the IBM Z Open Editor running in Red Hat Dev Spaces running VS Code, Red Hat offers a 30 day trial named the Developer Sandbox and can be found at the following website:

`https://developers.redhat.com/developer-sandbox`

After signing up for the trial, log on, and then use the following URL to load a workspace in the Cloud, running Z Open Editor and Zowe Explorer in VS Code in your browser:

`https://workspaces.openshift.com/#https://github.com/IBM/zopeneditor-sample/tree/devfile?che-editor=che-incubator/che-code/insiders`

The trial loads a Git repository with some sample COBOL, PL/I, HLASM, and REXX programs that you can test the editor experience with. For more detailed instructions, see the following website:

`https://github.com/IBM/zopeneditor-about/tree/main/che`

5.3.6 Layer 6: Moving z/OS development into the cloud

In the section outlining our vision for cloud-native hybrid application development above, we discussed the need to unify the development processes and tools of the hybrid team. We talked about developers such as Deb and Kathleen who are looking to utilize the same methods, editors, and tools for developing enterprise applications as well as cloud applications. In previous sections we talked about connectivity tools and editors such as Wazi for Dev Spaces that can run in the Cloud, deployed into an OpenShift namespace.

We now want to explore in this and the following sub-sections what else we can move to the Cloud to empower Deb and Kathleen to fully work in virtual environments that can be rapidly created and recreated and run close or even in the same development and test infrastructure

as the Cloud-based applications that they are building. We want to look at z/OS systems, infrastructure management tools, build tools and pipelines, as well as other automation.

We start in this section by looking at the various ways development teams can get access to z/OS systems that can run in the private or public Cloud. The basic usage model is the same for all of these offerings. They come with a preconfigured z/OS image that has been curated for IBM Z DevOps scenarios as described in this chapter providing many of the IBM software components needed such as compilers for all Enterprise languages as well as modern languages such as Java or Python, connectivity servers such as RSE API and z/OSMF, Debug, IBM Dependency Based Build and many more. These images are a great starting point for development teams, especially if they work with the IBM Z DevOps solution components. However, it is possible to customize the image or build your own based on your on-premises z/OS systems using the Wazi Image Builder. Coming back to our personas from Figure 5-1, the system programmer Zach would assemble the image for the team and then work with Todd, the Cloud operations administrator, to publish it in the right location for the following three technologies to pick up:

- IBM Wazi Sandbox is part of the IBM Z Cloud Modernization Stack. As with many of the modernization components it comes with an operator that allows easy deployment on Red Hat OpenShift clusters. It is targeted for on-premises deployment using x86 hardware, but it can be deployed in the Cloud as well in what is known as Virtual Private Cloud (VPC) environments. Todd will prepare the environment on Red Hat OpenShift so that Deb and Kathleen can easily create personal instances via the Red Hat OpenShift developer user interface by pasting and editing YAML files. The Sandbox instances can then be used as the development platform with other Modernization Stack components such as Wazi for Dev Spaces, Cloud Broker, as well as z/OS Connect.
- IBM Virtual Dev and Test for z/OS (ZVDT) is very similar to the Wazi Sandbox with the difference that it is targeted to all-Z development shops. The virtual server instances run on s390x hardware running Linux on IBM Z. The z/OS system gets emulated on the real target hardware offering big performance advantages. ZVDT can eliminate typical development bottlenecks by enabling flexible, horizontal scaling of early development, test, and education activities.
- IBM Wazi as a Service is a pure Cloud-based offering that also runs on real IBM Z hardware in IBM's Cloud data centers in many regions throughout the world. Developers such as Deb and Kathleen can self-provision z/OS virtual server instances into a secure virtual network managed by Todd in a VPC. As with the other two solutions they can create instances using the IBM default development and test image or use a custom image created by Zach and uploaded by Todd into secure Cloud Object Storage. By using the regular Cloud interfaces and automation utilities they can add additional volumes for test databases and other requirements.

With their own z/OS system for development, Deb and Kathleen are empowered to configure this system, using it for building and debugging, deploying their applications and running local tests before committing code and submitting changes to the larger team and Z DevOps pipeline. With a Wazi Code editor such as Wazi for Dev Spaces, they can connect to these systems in the Cloud or on-premises Red Hat OpenShift clusters without the need for extra network configurations if everything is deployed into the same subnet, simplifying how they can use these systems for their development activities.

5.3.7 Layer 7: Establishing a common control platform on Red Hat OpenShift

Describing the usage scenarios for IBM Sandbox in section 5.3.6, “Layer 6: Moving z/OS development into the cloud” on page 81, we briefly mentioned the role of operators in Red Hat OpenShift playing a role for deploying the Wazi Sandbox and developers using the user interfaces in Red Hat OpenShift provided by this operator to create Sandbox instances. To

define the term, Operators are pieces of software that ease the operational complexity of running and maintaining applications and services in a Kubernetes cluster. They are defined as an extensibility pattern as part of the Kubernetes specification and the preferred method for deploying and managing applications on Red Hat OpenShift.

This approach of utilizing operators in Red Hat OpenShift is also part of a larger strategy for IBM's approach to hybrid cloud-native z/OS development. Assuming that hybrid development teams will want to create their hybrid applications for and with Red Hat OpenShift, then we want to utilize it as a general control platform for managing the development tool infrastructure as well as the development and test deployments of the entire application under development, even the z/OS parts, all from one central location: the Red Hat OpenShift dashboard and command line interface.

With our Red Hat OpenShift operators deploying the editor and development images for the editor as well as the emulated z/OS Sandbox systems, another solution component of the IBM Z and Cloud Modernization Stack - the z/OS Cloud Broker - can now be used to further configure and refine these development platforms. It could be used to, for example, install and configure additional development resources like a C++ compiler, through an Red Hat OpenShift user experience. In this way, z/OS Cloud Broker allows developers to interact directly with z/OS resources without deep mainframe expertise and to use z/OS in a cloud-native way. The user experience is designed in a Red Hat OpenShift native way: A developer can log on to the Developer Perspective, browse a catalog of resources available to deploy or review the resources already available. She can select resources and actions such as deploy, install, or uninstall.

Moreover, the z/OS Cloud Broker allows development teams to create their own custom operators using the popular Ansible framework with IBM's Ansible collections for z/OS to support direct z/OS interactions. We have already described in “Layer 5: Adopting containerization for deploying development tools with Red Hat OpenShift and Dev Spaces” on page 77 how VS Code and Wazi for Dev Spaces can provide a highly productive development environment for Red Hat Ansible and such operators empower the hybrid development teams to create and then standardize on common z/OS and other systems automation tasks without the need to involve system programmers.

5.3.8 Layer 8: Creating end-to-end automation with DBB/Groovy and Ansible collections for z/OS

So far, we have focused on automation for developers to provision and prepare their personal z/OS systems in the Cloud. The advantage of having these systems are that development teams can quickly spin up and configure such systems themselves and use them for development and test in any way they need. However, the potential disadvantage is that these systems, unless they were created with the Wazi Image Builder specifically for a particular project, are generic and in many cases need to be configured for the specific application under development. They require automation to configure the subsystems used, such as CICS, to build and to deploy the application.

Automation for configuring z/OS

We have covered several technologies, starting with connectivity APIs, that development teams can choose from to configure their freshly deployed Wazi Sandbox or Wazi as a Service virtual server instance (VSI). They can pick any of these based on the skill set available in the team or other preferences. The following is a summary of the most important ones.

- Zowe CLI and its many plugins: In addition to the basic z/OS interactions that Zowe CLI provides there is a large list of plugins that provide extensions for many more capabilities, including z/OS subsystems such as CICS, IMS, MQ, and IBM Db2, but also many third party technologies. The Zowe web site currently lists 18 fully Zowe project certified plugins available: https://www.openmainframeproject.org/all-projects/zowe/conformance
- Z Open Automation Utilities (ZOAU) is part of the IBM Z and Cloud Modernization Stack and was developed to serve partially as an alternative to JCL, but also augmenting the use of "legacy" JCL script with modern scripting techniques to ease the adoption for hybrid development teams. For example, there is an Open Source Node.js package available for developers to use ZOAU services in Node.js .

  ZOAU is designed as a natural way for programmers familiar with Linux and Unix to use the Unix System Services environment directly to access traditional MVS resources such as data sets directly without the need for JCL. The utilities have a name and syntax that is familiar to Unix developers. For example, you can use the `dls` command to list data sets, which has a similar syntax and output to the `ls` command that is available on Unix environments. ZOAU provides easy-to-use class libraries for accessing MVS resources such as data sets directly from other languages such as Python, without requiring installation or configuration of other software. And where specific language libraries do not exist, ZOAU provides a shared library interface written in C that can be invoked by any programming language using the appropriate C language bindings[2].
- Red Hat Ansible and Red Hat Ansible z/OS collections have been partially built on top of the ZOAU utility and requires it as a prerequisite, but it targets Red Hat Ansible developers familiar with that framework. The z/OS collections also cover modules for many subsystems such as CICS and IMS, but also z/OSMF workflows and IBM Z System Automation. More generally, Red Hat Ansible can also integrate with and orchestrate existing tooling and automation provided by the various technologies covered in this chapter. Red Hat Ansible provides a rich open framework to accelerate DevOps integration and practices[3].
- IBM z/OS Cloud Broker, with its extensibility via custom operators, can provide a higher level of automation than the script-based frameworks listed in the previous bullet points.
- Dependency Based Build and Groovy are intended as a pure build automation framework that will be discussed in “User Build Automation and Debug” on page 84. Dependency Based Build provides a JVM-based API that can be used for configuration tasks as well. A user-built script that gets executed from the editors to run on z/OS can perform system preparation tasks for the developer without requiring them to use another framework.
- JCL is also a valid alternative as it can be used in combination with any of the frameworks as well as offer the ability to upload and execute JCL directly from a remote system. Additionally, the Zowe Explorer can be used by developers to quickly find and run JCL.

User Build Automation and Debug

As enterprise application editors such as Z Open Editor or Wazi for Dev Spaces run on remote clients such as developer laptops and OpenShift clusters running on x86 Linux or Linux on Z, developers not only need the ability of real-time syntax checking as provided by our editor, but they want to actually build, run and debug their applications on z/OS.

The IBM Z and Cloud Modernization Stack includes the Dependency Based Build (DBB) that provides build automation capabilities not only for build pipelines as outlined further below, but also build capabilities for developers. This includes the ability to run builds via command line, using SSH shells to USS, as well as integrations to run builds for individual programs

[2] https://www.ibm.com/docs/en/zoau/1.2.x?topic=extending-zoau-other-languages
[3] https://www.ansible.com/blog/devops-and-ci/cd-with-automation-controller

right out of the IBM editors; if needed, after a save with a simple right-click command. For that, developers manage build configuration files (defining things such as which compiler to use and where to search for include files) that they can store right with their source code in Git, as well as build scripts that have been written in Groovy, a language made popular with the Gradle build framework, that makes it easy for hybrid development teams to understand and maintain.

Figure 5-6 shows a simple workflow diagram for Z Open Editor's user build that is also available with Wazi for Dev Spaces, and Wazi for Eclipse.

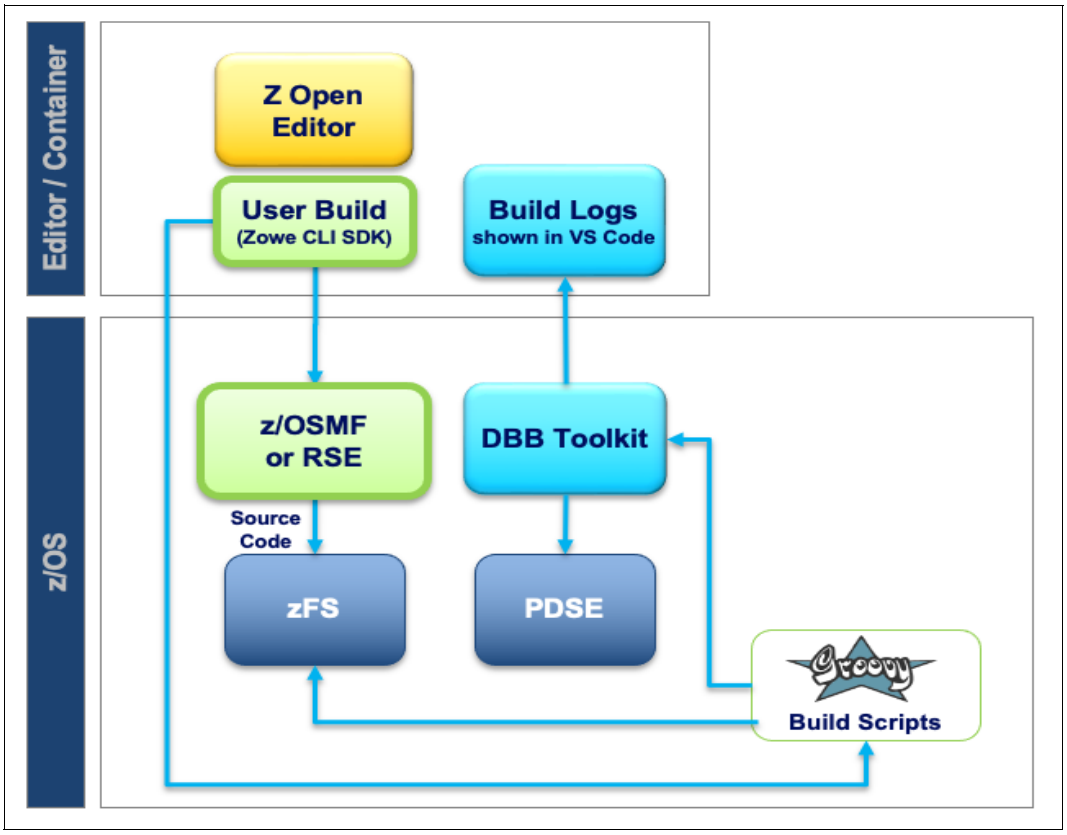


Figure 5-6 Z Open Editor's User Build with IBM Dependency Based Build.

As a Dependency Based Build requires installation of components on z/OS, the development and test images available for Wazi Sandbox, ZVDT and Wazi as a Service have everything installed and configured, ready for Deb and Kathleen to use directly from the Wazi editors as soon as they have their personal systems running. It is available for COBOL, PL/I and HLASM programs.

Deb or Kathleen can build the program that they are currently editing via a right-click command from the editor. This command then utilizes the editor's language server capabilities to compute the list of local dependencies, copybooks in the COBOL cases, and uploads these together with the program to zFS USS via a Zowe Explorer profile, utilizing the SDK capabilities of Zowe CLI described for Layer 3 in section 5.3.3, “Layer 3: Standardizing on next generation editors and modern languages capabilities” on page 74. Whether the user has created profiles for z/OSMF or RSE API does not matter; user build will automatically pick the one available. On USS, it will then remotely execute a Dependency Based Build via a Groovy build script. This build script performs all the operations required to build the application, creating PDS data sets, copy files, running the compile and link commands, etc.

The built application can then be deployed and started depending on the application. The developer could submit a JCL from Zowe Explorer or use any of the automation capabilities listed above. She can also start a Debug session for the IBM Z Open Debug VS Code extension. This Debugger provides all the capabilities that VS Code developers are used to when debugging in any other language such as setting break points, watching variable values, and stepping over and into calls.

Deployment Automation

Dependency Based Build (DBB) is not only used for user build from the editor, but the same build scripts and configuration settings can be reused to build many programs and applications as part of an automated pipeline. The goal of that pipeline would then also be to deploy the application it is building and run integration tests against it. Deployment automation frameworks such as Urban Code Deploy (UCD) can be used together with DBB.

UCD provides a web-based user experience for defining applications through conceptual models comprised of components, resources, processes, and environments. DBB performing a continuous integration (CI) build, such as for changes pushed to a Git server, can publish its build results to an artifact repository, such as Artifactory, and trigger a continuous deployment (CD) process in the UCD server. UCD will then orchestrate the deployment of the Artifactory assets to a dedicated z/OS test system via an agent installed there.

For a detailed tutorial showing how to install the z/OS agent with the UCD server to define an application process, set up a build pipeline for DBB in Jenkins, and connect UCD to the Artifactory, see the following website:

`https://www.ibm.com/docs/en/dbb/1.0.0?topic=api-reference`

This website also explores configuring UCD deployment steps for CICS and Db2.

5.3.9 Layer 9: Adopting a pipeline technology that matches the application platform

In the previous sub-section about automation we already mentioned that the various automation components can now all be utilized in fully automated CI/CD pipelines. As we enumerated a large list of automation technologies that developers can choose from, there is a similar long list of pipeline technologies that are available to development teams that they can then use in many different combinations with these automation tools. All the pipeline technologies listed below we consider (a) as Cloud-native, i.e. they can either run directly in the Cloud or access resources and nodes in Cloud, as well as (b) can integrate z/OS as a compute node into the pipeline workflow. For hybrid development teams we envision pipelines to provide fully automated processes to build, deploy, and test the entire hybrid application. These automated processes can be triggered by any developer's push operation of code changes in a versioning branch to a central Git repository management system. We will discuss representative example pipeline technologies here, but there are many more that could be utilized as well.

GitLab CI/CD

GitLab Ultimate is tailored for Hybrid Z DevOps scenarios. Together with Dependency Based Build and a GitLab Runner (an Agent, for example) that can run on Linux for IBM Z, it is a package tailored for hybrid teams that need to build z/OS components. GitLab servers and agents can be deployed in the Cloud as well as on-site and used in a fully distributed manner using the GitLab CI/CD automation framework for defining pipelines that run on many

different platforms and integrate all DevOps steps from source code management to deployment.

Just like all of its competitors such as GitHub or Azure, GitLab offers a proprietary pipeline description language in the form of YAML files that can be managed with the application source code in Git repositories stored on a GitLab server. Pipelines can then be triggered based on a branch push, run on a schedule, or be manually started. The agent that executes the pipeline scripts can be running on a regular virtual, Cloud-based, or hardware Linux or Linux for IBM Z nodes and execute build operations via SSH against a z/OS build machine running Dependency Based Build. Build results can then be pulled by the agents to populate build reports and artifact stores provided by GitLab or third parties such as Artifactory.

IBM has published a large number of tutorials and documentation for combining GitLab with many of the technologies discussed above. Find the whole list at the following website:

https://ibm.github.io/mainframe-downloads/DevOps_Acceleration_Program/resources.html

Some important examples are:

- Build a pipeline with GitLab CI, IBM Dependency Based Build, and IBM UrbanCode Deploy, which perfectly completes the automation story we started in the previous two sub-sections about using DBB with UCD as well as now orchestrated by GitLab CI/CD.
- Integrating IBM z/OS platform in CI/CD pipelines with GitLab, which is a deep-dive into configuring GitLab for hybrid applications building on multiple platforms as well as integrations with other Z DevOps capabilities such as Wazi Analyze.
- Implementing a Release-based Development Process for Mainframe Applications, which explores in detail how hybrid teams can manage Git branches and GitLab pipelines for parallel development, including user build, test automation, collaborative code review workflows, as well as frequent release deliveries.

IBM Cloud Toolchain

Although you can deploy GitLab securely in a Virtual Private Cloud in Virtual Server instances or Kubernetes clusters, the most common deployment scenarios for GitLab will be on-premises with perhaps a hybrid mix of local and Cloud-based nodes. If you want to run a purely Cloud-native pipeline, you can utilize IBM Cloud's Continuous Delivery offering that offers a Toolchain feature or graphical modelling of your entire DevOps configuration from Git repositories to links to Cloud-based editors such as Wazi for Dev Spaces, to build pipelines and artifact and build report management and insights tools.

When creating your toolchain by using the graphical editors in IBM Cloud, there is a toolchain template you can pick that is tailored for building z/OS applications. For a detailed walkthrough of the wizard utilized by the template and all the options available, see the following website:

https://mediacenter.ibm.com/media/Create+a+toolchain+to+secure+z+OS+application+development+in+IBM+Cloud/1_ax4aw1hr

The resulting pipeline implements many of the practices that we explored in section 5.1, “Core practices of Z DevOps for hybrid enterprise application development” on page 62, such as for baselining build artifacts that use Compliance Evidence Lockers, which are critical to achieve various compliance standards. The goal with the toolchain is to provide a full trace of any change to all build and application assets for an audit of all the activities such as scans and tests that have been performed for the change. These evidence lockers are often

implemented as a set of versioned reports that are stored as baselined Git artifacts as well as databases or general storage that maintains binary artifacts.

Under the covers, the pipeline is executed as a Tekton pipeline that can then execute build steps against the target system. Developers manage their Tekton build scripts in a dedicated Git repository configured for the pipeline. They can define custom container images to be used for these builds that provide all the tools required to run a remote build against z/OS and then use any of the connectivity solutions that we described earlier, such as SSH or Zowe CLI to run the build steps on the target z/OS system.

Tekton

The IBM Cloud Toolchain provides an abstraction layer and graphical front-end for creating and running Tekton pipelines. Tekton is a very popular open-source framework for creating Kubernetes-native CI/CD systems and was created for the Kubernetes project. For a hybrid development team that works on a Kubernetes front-end application as well as z/OS backend applications, running the integrated pipeline on Kubernetes with Tekton can be an attractive alternative. Particularly, if you run on a different Kubernetes platform such as Red Hat OpenShift, AWS, or Azure, as well as develop with on-premises Kubernetes clusters, the use of Tekton directly can be a valuable alternative.

Red Hat OpenShift, that we positioned throughout this chapter as a potential central control center for the entire hybrid development project's development infrastructure offers with its Red Hat OpenShift Pipelines a great, almost platform independent, complement to the tools of the IBM Z and Cloud Modernization Stack. This means that you can define the pipeline with the defacto-standard Tekton and then potentially run it almost unmodified on the many different Red Hat OpenShift provider platforms, or slightly modified on any other Kubernetes platform. As with the other IBM Z and Cloud Modernization Stack features, the Red Hat OpenShift Pipelines can be written as code, but also integrate with the graphical Red Hat OpenShift user experience in the same Developer Perspective as the z/OS Cloud Broker, Wazi Sandbox, and Wazi for Dev Spaces operators, providing a one stop shop for developers of the hybrid project.

Although Tekton was designed to build and deploy Kubernetes applications it can also be used as an orchestrator for kicking off external sub-pipelines that run on other network nodes and wait for their completion. This can be utilized to integrate other non-Tekton execution models such as the ones required for a Dependency Based Build running on a z/OS node building and deploying z/OS applications. Tekton introduced in 2020 the concept of Custom Tasks to support these kinds of use cases. Andrews Smithson provides an example with some code snippets for how he integrated access to z/OS into his hybrid project's Tekton pipeline using the Zowe CLI on the following website:

https://medium.com/zowe/accessing-z-os-from-your-tekton-pipeline-4abaffd7110c

Jenkins

Jenkins represents the classic way of running an automation pipeline. There are many advantages to using Jenkins. Jenkins is a widely used platform with a very large and active community. It is easy to find developers with Jenkins skills, documentation and reusable solutions for most use cases, as well as a very large portfolio of extending plugins. Additionally, Jenkins can be deployed almost everywhere, including VPCs in the Cloud, Kubernetes, Docker and almost every operating system, and it can connect to almost everything, including z/OS. A detailed tutorial on how to build a pipeline with Jenkins, Dependency Based Build, and UrbanCode Deploy can be found on the following website:

https://developer.ibm.com/tutorials/build-a-pipeline-with-jenkins-Dependency Based-build-and-urbancode-deploy/

This tutorial describes in detail how to set up a pipeline with Jenkins, Dependency Based Build, and UrbanCode Deploy. You can compare it to building a pipeline with GitLab CI, IBM Dependency Based Build, and IBM UrbanCode Deploy, comparing a Jenkins setup with GitLab to understand the differences in the user experience and decide which fits better in your hybrid development project.

5.4 A next-generation developer end-to-end development example

In the final section of this chapter we will walk you through some examples for utilizing many of these technologies in a development project.

To illustrate how a next-generation developer can leverage the IBM Z and Cloud Modernization stack, we can take a closer look at Deb, our user persona representing an archetypical new z/OS developer, and explore how she is accustomed to working based on her background.

5.4.1 Deb's story

The following list provides a generic, high-level overview of the development workflow that Deb would typically follow to implement a change by using the practices and technologies she is familiar with. Section 5.4.3, “Applying next-generation development strategies and tools to mainframe application development” on page 92 maps these steps to a detailed example of their application in mainframe DevOps practices.

1. Receive the assignment: Deb receives a development task such as implementing a new feature in an application.
2. Get the latest code: She begins her work by cloning or pulling a copy of the application code from a central Git repository down to her local workstation.
3. Switch to the feature branch: On her local workstation, Deb creates a "feature branch", which is a new branch of the application code that she will dedicate for her specific task. Switching to this new feature branch to make the code changes for her task allows Deb to work on the task in isolation, in parallel with her team, without having to worry about their other development activities disturbing her work.
4. Make the code changes: On her feature branch, Deb uses her preferred IDE to make the code changes for her task.
5. Run a personal build and test: Once she has made her code changes, Deb runs a personal build of the application so that she can test it and verify that her new feature works and does not cause regressions. Automated unit testing is integrated into this build process.
6. Commit and push code changes: After determining that her code changes look good, Deb commits her code and then pushes her feature branch to the central Git repository.
7. Merge request and approval process: Once Deb's feature branch is on the central Git repository, she creates a merge request (also referred to as a "pull request" by some Git providers) to integrate her code changes into the common development branch for her team. Her team has set up an automated pipeline to run builds of the code in the merge request at this point, which also include tests and code scans.

This is also the point at which her team has implemented an approvals process where she can add teammates to review her changes and approve them before merging them into their common branch of code.

8. Integrate code changes: Once Deb's merge request is approved, her code changes are merged into her team's common development branch of code. The feature branch on which she did her development work can subsequently be deleted. A build of her team's common development branch, which now contains Deb's code changes, can be run to move the changes forward or associate them with a release.

As Deb enters the workforce, whether it is in distributed application development or mainframe application development, she wants to be able to continue working in this way with tools, technologies, and workflows that support the best practices she was taught.

5.4.2 Deb's tools

Having examined Deb's modern development background, we can now discuss the tools and technologies she is accustomed to using to achieve her desired development workflow from beginning to end. Within the CI/CD pipeline, these can be broken into the following six main components:

1. SCM
2. Integrated Development Environment (IDE)
3. Build component
4. Artifact repository
5. Deployment manager
6. Pipeline orchestrator

SCM

The software configuration management (SCM) tool is what Deb and her team use to store and manage different versions of their source code files, as well as application configuration files, test cases, and more. This is what enables Deb and her team to do parallel development.

Git has essentially become a de-facto industry standard in source code version control. Popular Git providers such as GitLab and GitHub have enhanced Git with graphical web interfaces and features such as merge requests or pull requests that help teams coordinate planning, development, and code review activities. They also provide webhooks and even pipeline orchestrators (for example, GitLab CI/CD and GitHub Actions) to help integrate the coding step of the DevOps cycle with other CI/CD steps.

Note: As the source code for mainframe applications might be currently managed on z/OS, the following methods exist to facilitate the migration of a monolithic mainframe application into logically decoupled Git repositories:

- The Dependency Based Build migration tool
- The SCLM migration tool
- Manual migration of source files in MVS to Git repositories is also a possibility. The files would first be copied into USS using ISPF/TSO, the USS command line, or IDz, and a Git repository could then be initiated from the destination folder in USS

IBM provides a free, self-paced, online course where you can learn more about the source code migration process and other DBB fundamentals on the following website:

https://learn.ibm.com/course/view.php?id=5146

Integrated Development Environment

Deb uses the integrated development environment (IDE) to check out and edit her code, as well check it back into the SCM. Many modern IDEs have features that enhance development capabilities, such as real time syntax checking, code completion, outline views of the code structure, variable declaration lookups, and variable reference search.

Although the IDE component is often installed on the developer's local workstation, newer containerized options on the cloud, such as IBM Wazi for Dev Spaces, allow developers to access and use a shared image of a development environment provided by their team via a web URL. The image contains the configured IDE and other development tools used by the team, which reduces the time and effort required to generate new instances of the development environment and onboard new teammates.

Build component

The build component takes care of understanding and resolving dependencies, and then converting the source code to produce the executable software artifacts. In this component, Deb and her team can also integrate automated steps for unit testing and code quality inspection.

For z/OS application development, IBM Dependency Based Build (DBB) provides an intelligent build capability that discovers and resolves dependencies between objects before compiling and link-editing the application. This build tool is optimized to be able to build only the changed programs and their impacts (similar to how mainframe application teams often manage builds), but DBB enables automation of these traditionally manual z/OS development processes so that they are more efficient and can be integrated into modern CI/CD pipelines. DBB is often used with zAppBuild, a sample z/OS application build framework that can be customized to your enterprise's needs.

Artifact repository

After the build process is complete, the resulting build outputs are packaged together along with anything else that the team would like to install or run during deployment. The package is then uploaded and stored in the artifact repository, which also stores metadata to help trace the software artifacts back to the source code that they originated from. This helps decouple the SCM from the runtime environments, which is a fundamental DevOps practice.

Deployment manager

The deployment manager component rolls out application packages to various environments for purposes such as acceptance testing, system integration testing, and even production. The deployment manager also keeps track of the inventory of execution environments so the team can know what each one is running.

IBM UrbanCode Deploy (UCD) is a popular deployment manager option, but it is also possible for the development team to script their deployment. Sample CI/CD pipeline scripts that can be customized for your use case can be found in the Pipeline section of DBB's public GitHub repository:

https://github.com/IBM/dbb/tree/main/Pipeline

Pipeline orchestrator

Overseeing all the automated processes in the pipeline is the pipeline orchestrator. This component integrates the steps from the different tools (in particular, the SCM, build component, artifact repository, and deployment manager) together and ensures that they all run in the correct order.

A centrally-controlled pipeline is important for implementing a safe development process in which changes can only be delivered through the pipeline process. The pipeline itself is mostly automated, although development teams can also integrate manual approval steps where necessary.

5.4.3 Applying next-generation development strategies and tools to mainframe application development

With the next-generation tools available today, we can adapt Deb's workflow to mainframe application development. For some components such as the SCM, artifact repository, and pipeline orchestrator, the z/OS application development process can use the same technologies favored by distributed development teams. This enables standardization of development tools across the enterprise, which can allow z/OS and distributed application teams to collaborate more easily, breaking down the silos between them and paving the way for enabling a multi-technology application architecture. There are many choices for integrating modern development tooling and strategies into mainframe development processes. The following scenario with Deb's team illustrates an example implementation.

CI/CD pipeline setup

Sharing the technologies used by their enterprise's distributed development teams, Deb's team is using GitLab as their SCM and GitLab CI/CD as their pipeline orchestrator. Thus, their z/OS application's source code, tests, and configurations are stored on a central Git repository hosted by GitLab. GitLab CI/CD will integrate the SCM with other pipeline tools such as DBB to perform builds and other CI/CD actions at appropriate points in the development process.

To streamline the onboarding process for developers, Deb's team has set up a preconfigured developer workspace in IBM Wazi for Dev Spaces that contains all the necessary source code, z/OS access, and tools to be productive on the team. Deb and her teammates can access instances of this development environment with a single click after logging onto the team's Wazi for Dev Spaces website. Alternatively, if she prefers, Deb can use Git to create a local copy, or clone, of her team's application repository on her local workstation, allowing her to work on her development tasks there using IBM Wazi for VS Code or IBM Wazi for Eclipse.

In this case, Deb is picking IBM Wazi for Dev Spaces as the most convenient option, as it already set up for her and does not require any special configurations or tools on her local workstation.

Preparation of the z/OS environment

In order to easily create the z/OS development and test environments for Deb's team to work on, their system programmer, Zach, uses the Wazi Image Builder to extract a custom image from their organization's on-premises IBM Z platform. This image contains the setup and configuration information for the z/OS system, and the organization's cloud administrator, Todd, uses it as the basis to create additional development and test environments. Todd has the option to use IBM Cloud's graphical web interface to create the new Wazi as a Service (WaaS) target environments from the image. Although this graphical approach only takes a few clicks per environment instance, Todd has implemented a more streamlined process by using an "infrastructure as code" (IAC) approach. The IAC approach leverages Terraform and Ansible scripts to automate the creation of the target environments, making the process more efficient and scalable. This means that when Deb is ready to connect to their z/OS environment, she can self-provision her z/OS virtual server instance on the VPC managed by Todd. Section 5.5, “IBM Wazi as a Service and IBM Z and Cloud Modernization Stack tutorial” on page 97 provides a more detailed introduction to the concepts and graphical interfaces of WaaS.

Code

Deb follows her team's Git-based CI/CD workflow for each development task she takes on and uses the following steps.

1. Receive the assignment: After completing a planning session with other teams in their business unit, Deb's team follows up by creating issues within GitLab according to features, tasks, and bugfixes identified as targets for the next milestone. They then organize these issues on a GitLab issue board for the milestone, and move the issues between the issue board's columns to represent different statuses as they progress.

   For this particular example, Deb has been assigned an issue to fix a display message in one of her team's COBOL programs, similar to that shown in Figure 5-7. She can view the details of her assigned issue on its dedicated GitLab issue page, as well as update her issue's description, status, comments, and other metadata there.

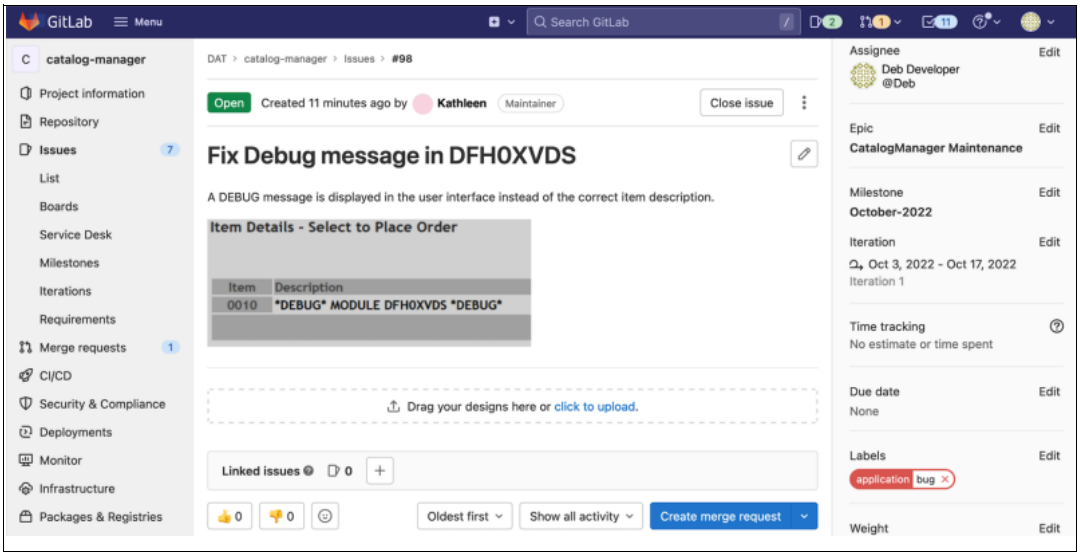


Figure 5-7 GitLab issue interface with issue description and metadata.

2. Get the latest code: Deb logs into her team's Wazi for Dev Spaces website, and accesses her instance of the team's development workspace. This environment already contains a copy of the application code along with the necessary development tools. Deb can use the Git integration in the IDE to pull any latest changes from the team's central Git repository.
3. Switch to the feature branch: To begin working on her assigned issue, Deb uses the Git integration on her IDE to create a new branch of the application code. This new branch is a copy of her team's shared development code, and she ensures that she switches to this new branch before beginning her coding work so that the branch is completely dedicated to her work on her assigned task.
4. Make the code changes: Using the IBM Z Open Editor and Z Open Debug extensions to support her coding work, Deb makes the code changes to fix her assigned issue. She can also run unit tests related to the code she is modifying in order to make her changes more robust and easier to maintain. She uses the Zowe Explorer extension to connect remotely with her team's mainframe system to manage her data sets, USS files, and jobs. The IDE she is using provides her with the following capabilities, among others:
   - Editing: In Z Open Editor, Deb uses the Outline View to explore and navigate the code for the program she is fixing. The copybook resolution feature allows her to preview contents of copybooks referenced in the COBOL program by hovering her mouse curser over the copybook name, and she can also open the copybook itself in a separate editor by simply pressing Ctrl+Click (Windows).

     While Deb is coding her fixes, Z Open Editor provides a code completion feature, which lets her easily pick from a selection list of commands, defined variables names, and code snippets as she types in commands, variables, or paragraph names. Deb also takes advantage of the hover declaration lookup feature that allows her to hover over a variable or paragraph name and view its definition (see Figure 5-8). She uses the variable reference search feature to search for and find all references to the variables and paragraphs she is interested in.

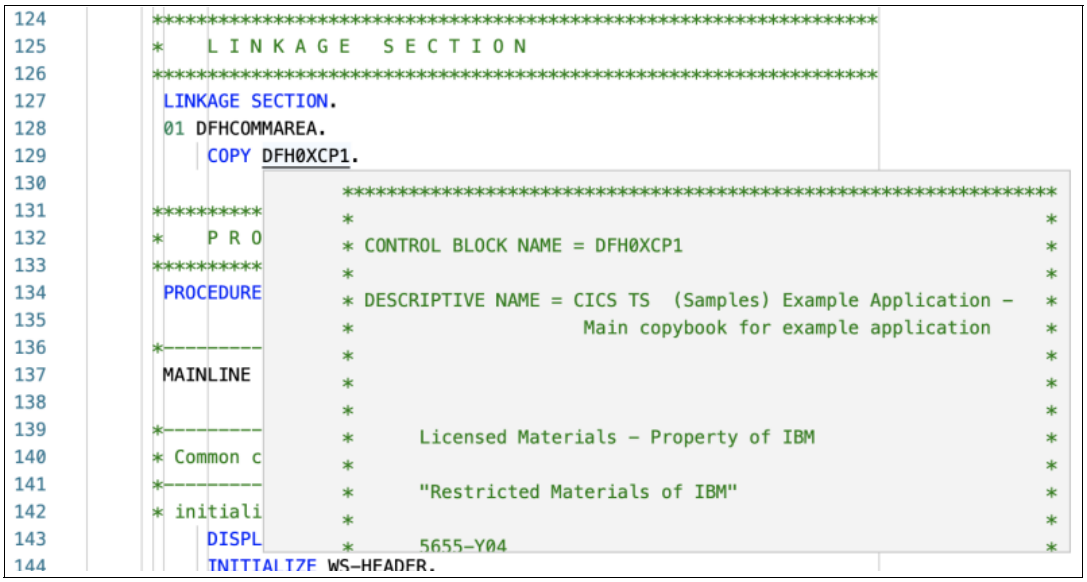


Figure 5-8 Hovering over the underlined copybook reference reveals a preview of the copybook

- Debugging: Within the IDE, the IBM Z Open Debug extension helps Deb troubleshoot her issue by allowing her to set breakpoints in her COBOL code and view the state of different variables at different points in the program or application. This lets Deb monitor the code's execution path to analyze the root cause of the issue.

5. Run a personal build and test: Once Deb has made her code changes that she is ready to test, she uses DBB's User Build tool to create a personal build of the program so that she can test it and verify that her fix works and does not cause regressions. Figure 5-9 shows the build log. Deb can install the generated build artifacts (such as load modules) into her self-provisioned z/OS instance and perform manual testing without having to worry about her team's other development activities affecting the results of her testing. The User Build process also includes automated testing, which provides her with an additional efficient way to check for regressions.

   - Unit testing: Deb and her teammates can use the IBM z/OS Automated Unit Testing Framework (zUnit) within IBM Developer for z/OS (IDz) to create unit tests, which can then be stored as part of the same application repository containing the source code. Deb's team can then script the execution of these zUnit tests to automatically occur after a build is complete.

```
** Build start at 20221010.044724.047
** Build output located at /u/ibmuser/projects/logs
** Adding /u/ibmuser/projects/catalog-manager/cobol/DFH0XVDS.cbl to Building build list
** Writing build list file to /u/ibmuser/projects/logs/buildList.txt
** Invoking build scripts according to build order: Assembler.groovy,BMS.groovy,Cobol.groovy,LinkEdit.groovy
** Invoking test scripts according to test order: ZunitConfig.groovy
** Building files mapped to Cobol.groovy script
*** Building file catalog-manager/cobol/DFH0XVDS.cbl
** Writing build report data to /u/ibmuser/projects/logs/BuildReport.json
** Writing build report to /u/ibmuser/projects/logs/BuildReport.html
** Build ended at Mon Oct 10 04:47:28 EDT 2022
** Build State : CLEAN
** Total files processed : 1
** Total build time  : 3.819 seconds
```

Figure 5-9 Sample DBB User Build log with build results outlined in green.

6. Commit and push code changes: Having coded, built, and tested her fix for the assigned issue, Deb is happy with her code changes, and uses the IDE's Git integration to commit and push her feature branch with her code changes to her team's central Git repository.
7. Merge request and approval process: Now that her feature branch to fix her assigned issue is on the central Git repository, Deb opens a merge request in GitLab to have her code changes merged into the common development branch for her team. Deb's team has set up an automated GitLab CI/CD pipeline to run builds of the code for any changes to merge request code, which also include tests and code scans (see Figure 5-10). Following her team's approval process, Deb also adds some teammates to review her changes in the merge request.

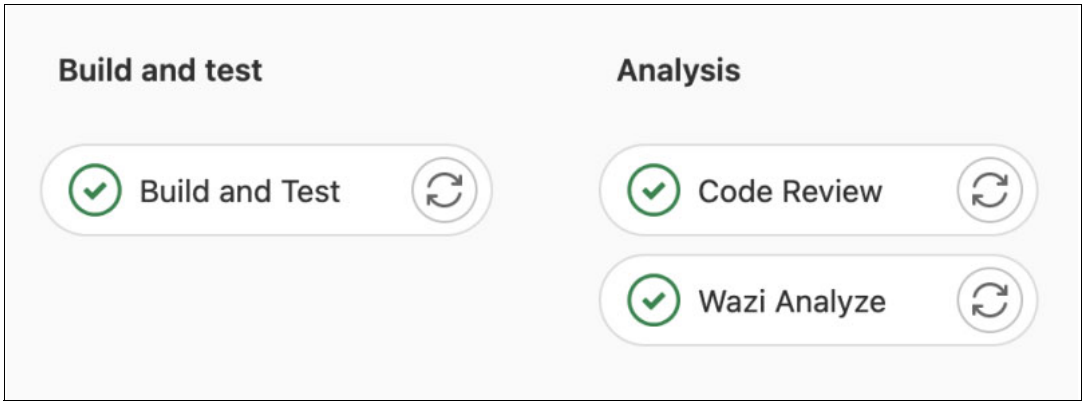


Figure 5-10 The primary steps in a CI/CD pipeline for a merge request's feature branch.

8. Integrate code changes: Once her request is approved and her fix is merged in, the feature branch she used for her development work can be deleted, and a full or impact pipeline build can run on the team's common development branch (Figure 5-11)- now with Deb's fix - to move the changes forward or associate them with a release.

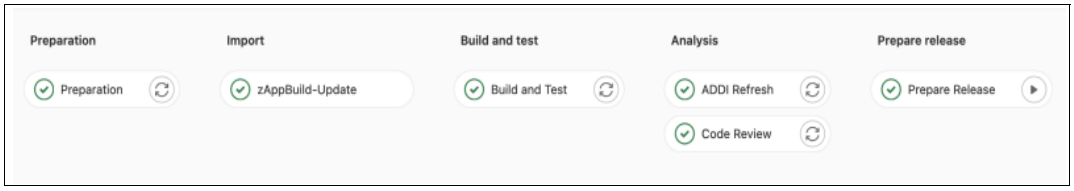


Figure 5-11 An example of a CI/CD pipeline that can run when a merge request is merged into the team's shared development branch.

After Deb’s code changes are merged in, the artifacts and metadata generated by the pipeline build on the team’s common development branch in step 8 can be uploaded and stored in the team's artifact repository, from where they can subsequently be downloaded and deployed to the various test environments such as system integration or acceptance testing environments. The artifact repository serves as a way to enable the deployment of the same package into multiple environments, as well as a way to trace the generated build artifacts within that package back to their source code origins. The deployment itself can either be handled by a deployment manager such as IBM UrbanCode Deploy or, as Deb's team has set up, it can be handled by a deployment script.

Once Deb's team has achieved their development milestone and is ready for release, they run predefined steps in their GitLab CI/CD pipeline orchestrator to create a release package from the generated build artifacts, as well as create a release on their GitLab SCM. The release package is uploaded to the team's artifact repository, from where it can then be downloaded

and deployed to the various testing environments and, once ready, also deployed to the production environment.

For more guidance on CI/CD pipelines designs that can be implemented at different points in the release development process, see the following website:

https://www.ibm.com/support/pages/node/6619083

5.5 IBM Wazi as a Service and IBM Z and Cloud Modernization Stack tutorial

This section will give you the opportunity to try some of the tools and methods discussed throughout this chapter yourself, hands-on. The following tutorial will provide the steps that you need to perform for deploying a complete Cloud-based development environment. For that all you need is a Web browser (Firefox or Google Chrome) and nothing else. The system that we are deploying is shown in Figure 5-12, which is a variant of Figure 5-5 replacing Wazi Sandbox running in Red Hat OpenShift now with Wazi as a Service running in the IBM Cloud on real IBM Z hardware as a virtual server instance.

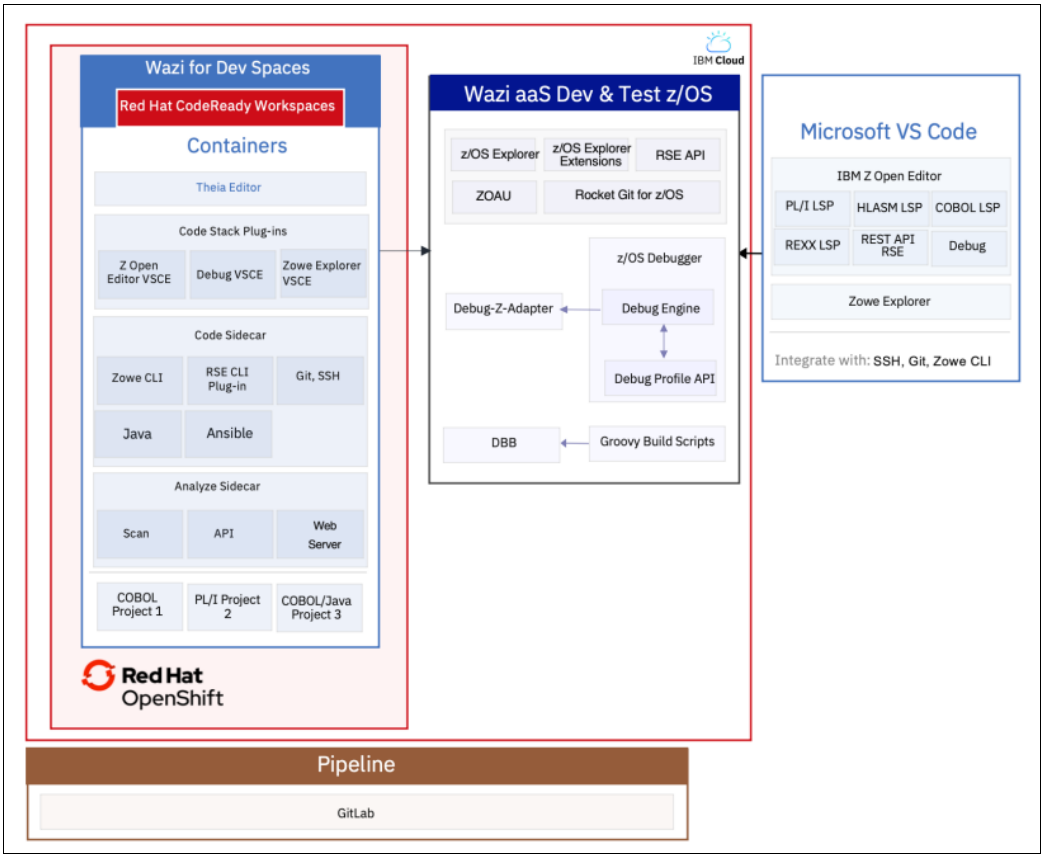


Figure 5-12 Cloud Deployment Diagram of the tutorial system.

Once set-up, you will then be able to use the Cloud-based editor in Wazi for Dev Spaces to edit, build, and debug a sample COBOL program in that environment. There will be some optional steps that we will not cover in this tutorial to keep it more brief, such as setting up a virtual private network (VPN) or deploying a local GitLab server and GitLab runner to run a

pipeline. For these pieces we will provide links to the documentation and recorded demo videos that show you how those parts work. With this system you would also be able to try other components from the IBM Z and Cloud Modernization Stack that we cannot cover here.

Note, that there will be a fee associated with creating these systems. The costs will be shown when you create these in the IBM Cloud creation dialogs. In most cases, the charges will be per hour. So, if you keep your tutorial walk-through brief and delete your resources afterwards the charges should be reasonable.

5.5.1 Creating a Virtual Private Cloud and z/OS Virtual Server instance

Let's assume the role of Todd, who is tasked with provisioning a fully Cloud-based development infrastructure for the team. His plan is to provision a Virtual Private Cloud (VPC) for the team and deploy everything they need there. To sketch out his plan and to discuss it with the team he creates the diagram in Figure 5-13 that shows all the components he plans to deploy and configure.

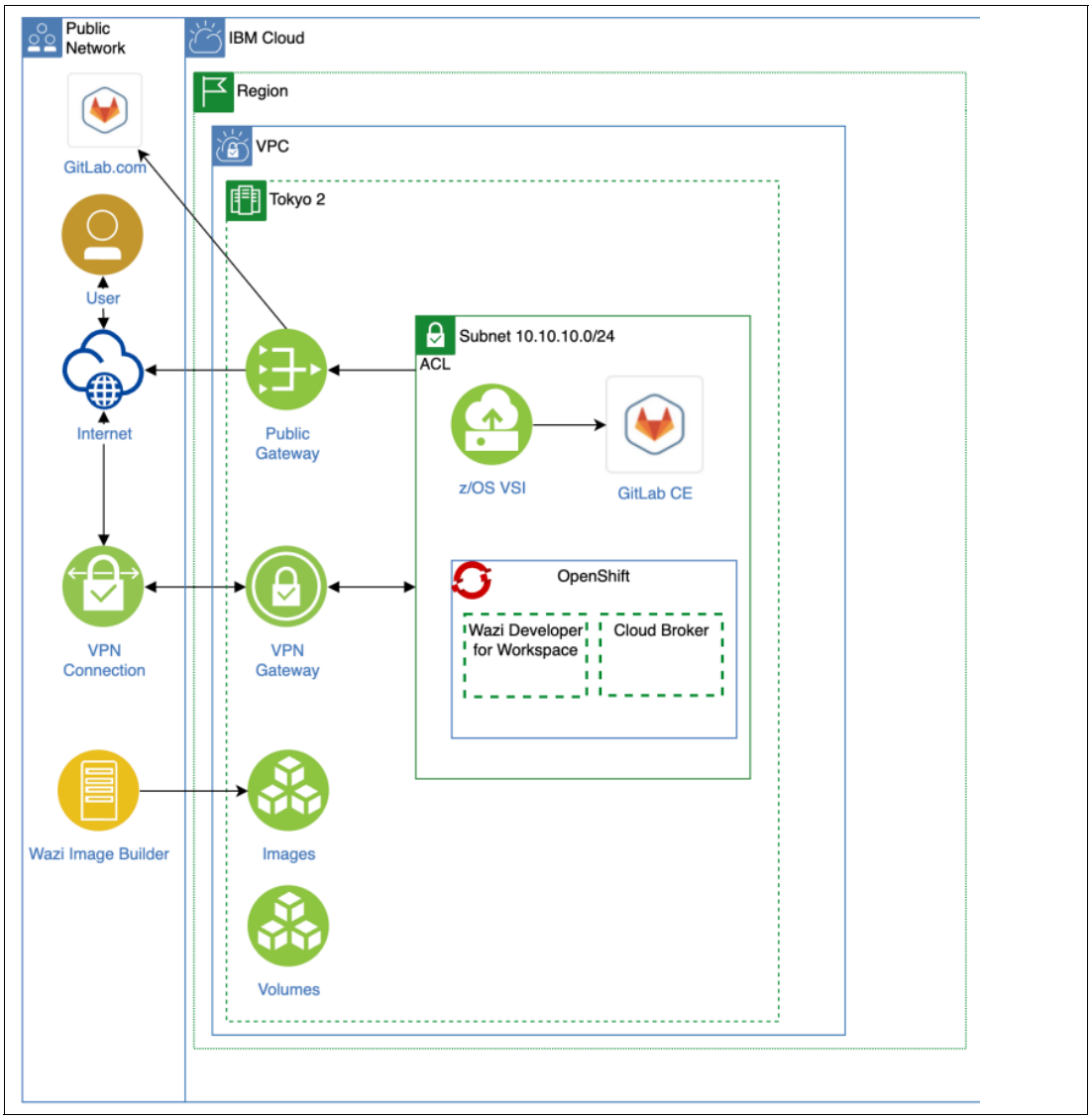


Figure 5-13 Cloud Deployment Diagram of the tutorial system.

As an experienced Cloud administrator he would deploy this infrastructure entirely with automation scripts written in Terraform and Ansible, based on reference scripts provided by IBM. However, this time he wants to teach his colleagues Deb and Kathleen the principles of this technology and is using the IBM Cloud graphical user interface to deploy each component one by one to explain each piece more visually.

The first step is to create an account in IBM Cloud. He upgrades his new account to a Pay-As-You-Go account. For more information, see the following website:

https://www.ibm.com/docs/en/wazi-aas/1.0.0?topic=vpc-setting-up-your-cloud-account

He would also create access groups and add other Cloud administrators to this account and give them permissions. Deb and Kathleen do not need an account.

Once he has access he is ready to create a Virtual Private Cloud and a virtual z/OS server instance (VSI). For more details on this, see the following website:

https://www.ibm.com/docs/en/wazi-aas/1.0.0?topic=vpc-getting-started-virtual-private-cloud

He starts by showing Deb and Kathleen the IBM Cloud VPC overview page - in our example, https://cloud.ibm.com/vpc-ext/overview and shown in Figure 5-14, which is the central hub for creating and configuring many of the components that he needs.

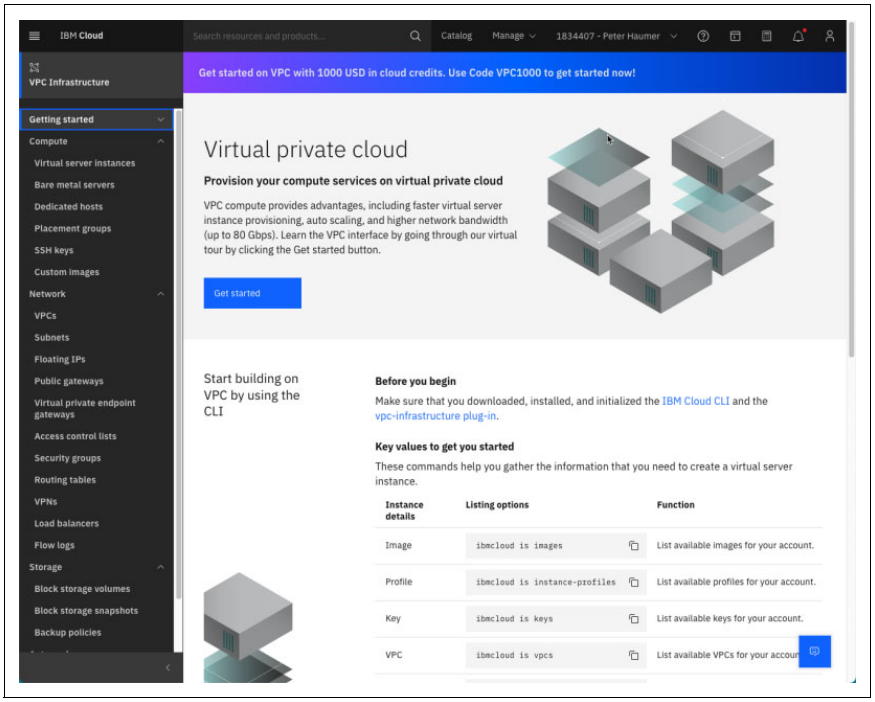


Figure 5-14 IBM Cloud Virtual Private Cloud landing page

From the table of contents on the left, he navigates to various pages to create the resources needed. The pages used include the following tasks:

1. The SSH keys page provides a public key to be used later for connecting to the z/OS VSIs.
2. He creates a VPC in his preferred region that also hosts z/OS VSIs.
3. He creates subnets in three zones as he will use the same VPCs and subnets to also run his OpenShift cluster later.

4. He attaches a public gateway as he wants access to the public internet to be able to download IBM OpenShift Operators and images such as Wazi for Dev Spaces.
5. He configures an access control list (ACL) to limit the subnet's inbound and outbound traffic.

Todd would also create a VPN for clients to connect securely to the VPC from his organization. For more details on this, see the following website:

https://www.ibm.com/docs/en/wazi-aas/1.0.0?topic=vpc-creating-client-site-vpn-server

With the VPC ready and configured he can now deploy his first z/OS Virtual Server Instance (Figure 5-15).

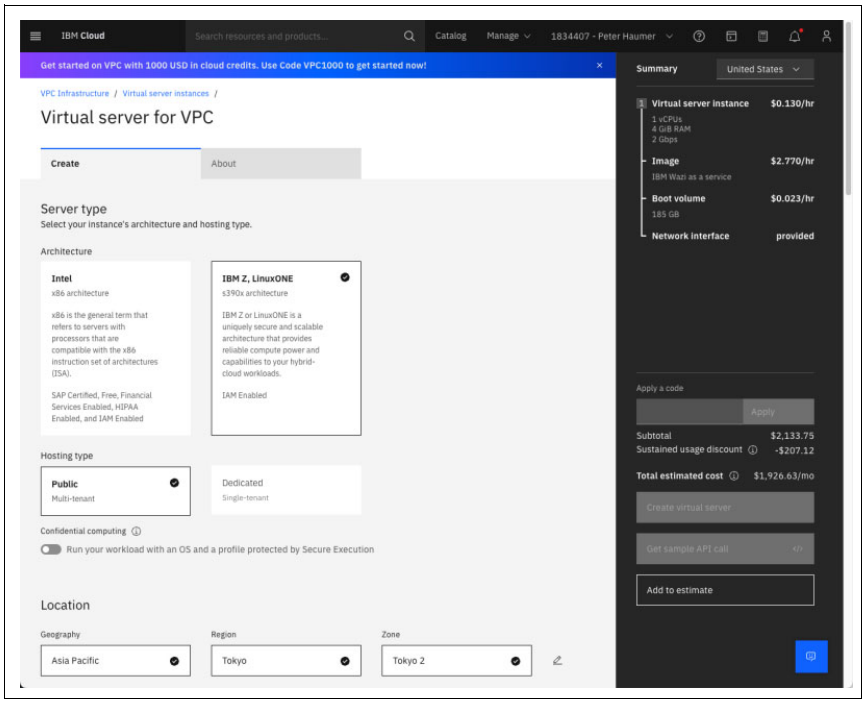


Figure 5-15 Creating a Virtual Server instance for the IBM Z architecture

He performs the following tasks. The screen he is using is shown in Figure 5-16.

1. On IBM Cloud's left side menu, he navigates to "VPC Infrastructure" -> "Virtual server instances".
2. After clicking the "Create" button on the Virtual server instances page, he is brought to the screen shown in Figure 5-15, and selects the "IBM Z, LinuxONE" architecture template.
3. He selects his VPC Geography, which automatically also selects the same Region and Zone (subnets) of the VPC.
4. Moving down to the "Details" section, he enters a unique name for his VSI, and then selects the appropriate resource group.
5. He scrolls through to the screen shown in Figure 5-16, and then selects "IBM z/OS" as the image's operating system, with the image version "ibm-zos-2-4-s390x-dev-test-wazi-3". (Note: The actual number at the end of the image version might vary.)
6. He selects a profile with his vCPU and RAM requirements, mz2-2x16. He selected this as ?it is a good choice for evaluation.

7. Under "SSH keys", he selects his SSH key created earlier to be used for connecting to the VSI.

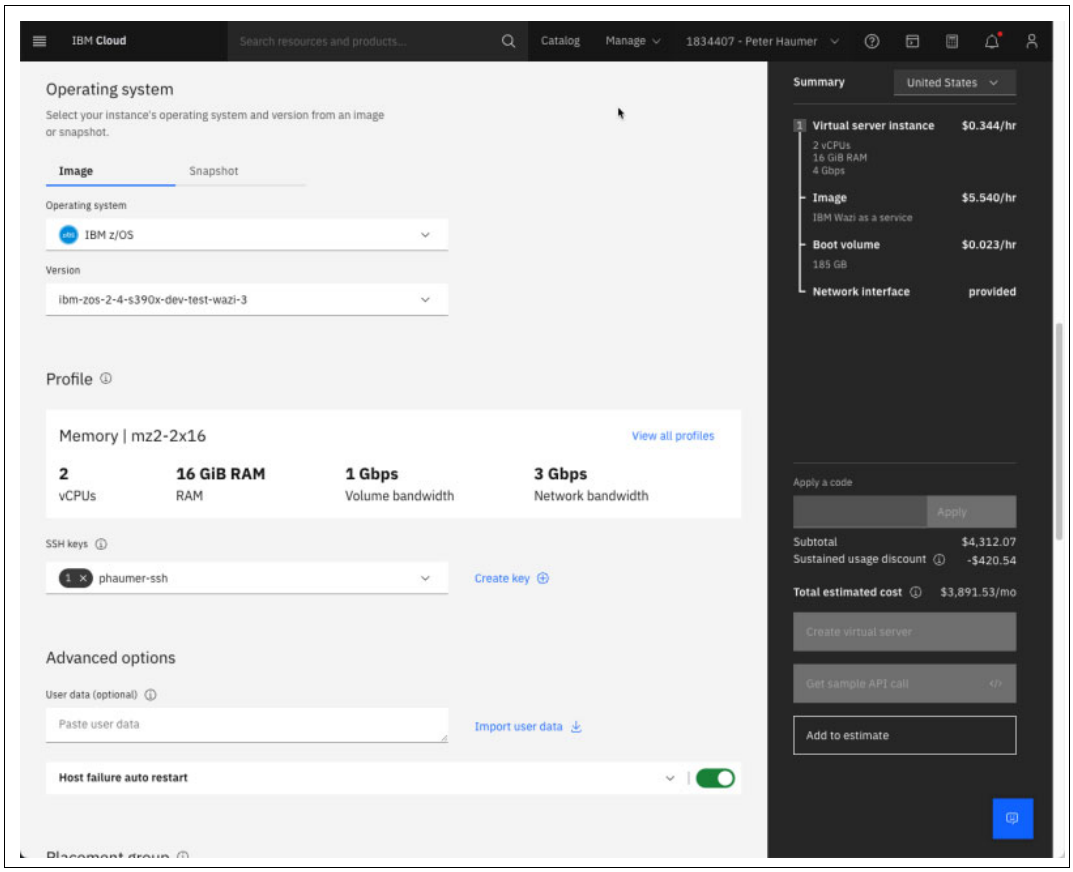


Figure 5-16 Creating a Virtual Server instance with the z/OS Development and Test image

Todd reviews the cost estimates for his configuration choices on the right and clicks the "Create virtual server" button to deploy his new VSI. The new VSI will appear in his list of VSIs in the VPC management page.

To be able to access the VSI now without setting up a VPN he temporarily creates a Floating IP by selecting his VSI from the list, drilling into the details screen and using the Network Interfaces panel to reserve and assign one. He can now try connecting to the VSI via SSH by using the following command:

```
ssh ibmuser@<ip-address>
```

Once logged on he can enable a default user with a secure password that can be used for other setup work by using the following command:

```
tsocmd "ALTUSER IBMUSER PASSWORD("password") NOEXPIRE RESUME"
```

5.5.2 Deploying Red Hat OpenShift and IBM for Wazi Dev Spaces in a VPC

After having a VPC and VSI created in the IBM Cloud, Todd wants to deploy Red Hat OpenShift into the same VPC for Deb and Kathleen's development work.

The images that are required for running an IBM Z and Cloud Modernization Stack with Wazi for Dev Spaces on Red Hat OpenShift are in public IBM and Red Hat catalogs that need to be

pulled from the internet. To prepare that, Todd needs to configure a public gateway that makes the internet reachable for his subnets for outgoing traffic.

1. He uses the VPC management pages in IBM to create and assign a public gateway by logging into the following URL: https://cloud.ibm.com/vpc-ext/network/publicGateways
2. He clicks the Create button.
3. Once created, he uses the Subnets management page to drill into each subnet and attaches the public gateway with the slider.
4. Todd then also needs to order encrypted Cloud Storage required by Wazi for Dev Spaces to persist the developer's virtual workspaces. The quickest way to order this is to use the Search widget at the top of each IBM Cloud page to search for Object Storage, select IBM Cloud as the infrastructure and select the free Lite plan for now.
5. Todd then provides a name for his Cloud Object Store and creates it.

Now he is ready to deploy a Red Hat OpenShift cluster into his VPC and performs the following steps.

1. Todd again uses the Search widget at the top of each IBM Cloud page to search for Red Hat OpenShift.
2. On the Red Hat OpenShift creation page, he selects VPC as the infrastructure for deployment.
3. He selects his VPC and his Cloud Object Storage to be used for this cluster.
4. He selects Red Hat OpenShift 4.9 or newer. He uses the defaults for all the other values.
5. On the right side of the screen he reviews the cost estimates and clicks Create.

After the deployment has finished, he clicks the OpenShift web console button on the cluster screen to navigate to the OpenShift console.

In OpenShift he can now configure the IBM Operator Catalog and deploy IBM Z and Cloud Modernization Stack components through it. In the following, Todd deploys Wazi for Dev Spaces by performing the following tasks.

1. In the OpenShift web console, Todd uses the Administrator perspective and clicks the plus (+) icon at the top-right to open a YAML editor and submits the YAML shown in Example 5-1 to configure the IBM Catalog.

Example 5-1 YAML to configure the IBM Catalog

```
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:latest
  updateStrategy:
    registryPoll:
      interval: 45m
```

2. Once the catalog is added he navigates to Operator → OperatorHub and searches for "wazi" in the *Filter by keyword* field.

3. He clicks the IBM Wazi for Dev Spaces operator. A summary dialog is displayed.
4. He clicks Install and accepts all the default settings and clicks the next Install button.
5. The operator installation prompts him to create a license. He clicks on the link and accepts the license terms and installs the license.

He can now create instances of Wazi for Dev Spaces by using the "Create Instance" link in the installed operator's Details page by using the following steps.

1. In the Create CheCluster page, he accepts all the defaults and clicks the Create button.
2. Once the creation has finished and Wazi for Dev Spaces is deployed, he will be able to drill into the instance from the IBM Wazi for Dev Spaces list and find all the details needed as shown in Figure 5-17.

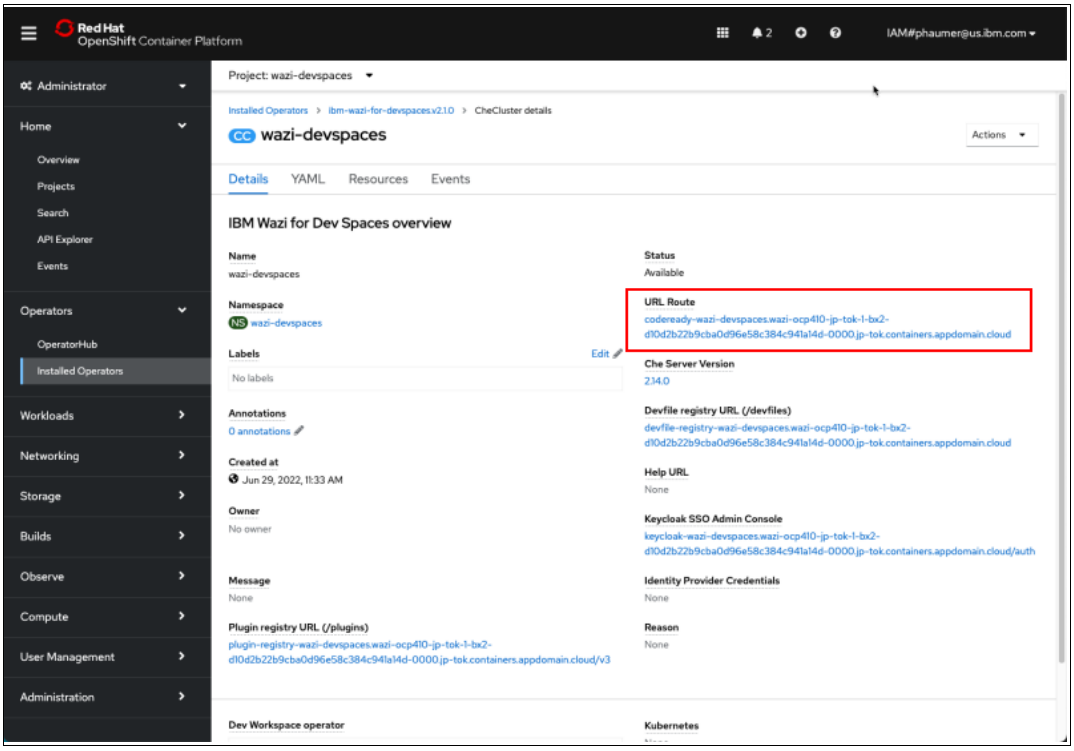


Figure 5-17 Deployed Wazi for Dev Spaces instances

Todd is using the Administrator menu on the left of the Red Hat OpenShift console to navigate to Workloads → Secrets and drills into the "che-identity-secret". Here he finds the password generated that is used for several administrative purposes by clicking "Reveal values".

Back in the Dev Spaces instance page shown in Figure 5-17, he uses the link to the "Keycloak SSO Admin Console" with the password he just looked up to create accounts for Deb, Kathleen, and himself. For more details on managing identities and authorizations, see the following website:

https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.14/html/administration_guide/managing-identities-and-authorizations_crw

Finally, Wazi Dev Spaces is ready for Deb and Kathleen to use. He copies the links shown under URL Route (Figure 5-17) and sends it to them as this will be the URL for them to log on.

5.5.3 Creating and configuring a development workspace in Wazi Dev Spaces

Now that Todd has created a Wazi for Dev Spaces deployment in IBM Cloud running in a VPC and created user accounts, Kathleen can use the URL provided by Todd to log on to the Dev Spaces Dashboard and create her personal development workspace. The dashboard shows the Create Workspace page that presents several prepared templates for different technology stacks. As Kathleen is trying the editor for the first time, she selects the "Wazi Code with sample apps" template that provides a complete technology stack for COBOL, PL/I, HLASM, REXX, Zowe CLI/Zowe Explorer as well as Ansible automation clients and sample applications for all the languages that she can use for experimentation.

After the workspace has started up, she is in the Cloud-based editor running in her browser with the sample application cloned from a Git repository as shown in Figure 5-18. For security, she was asked before cloning the repo if she trusts the repository source, which she answered with Yes.

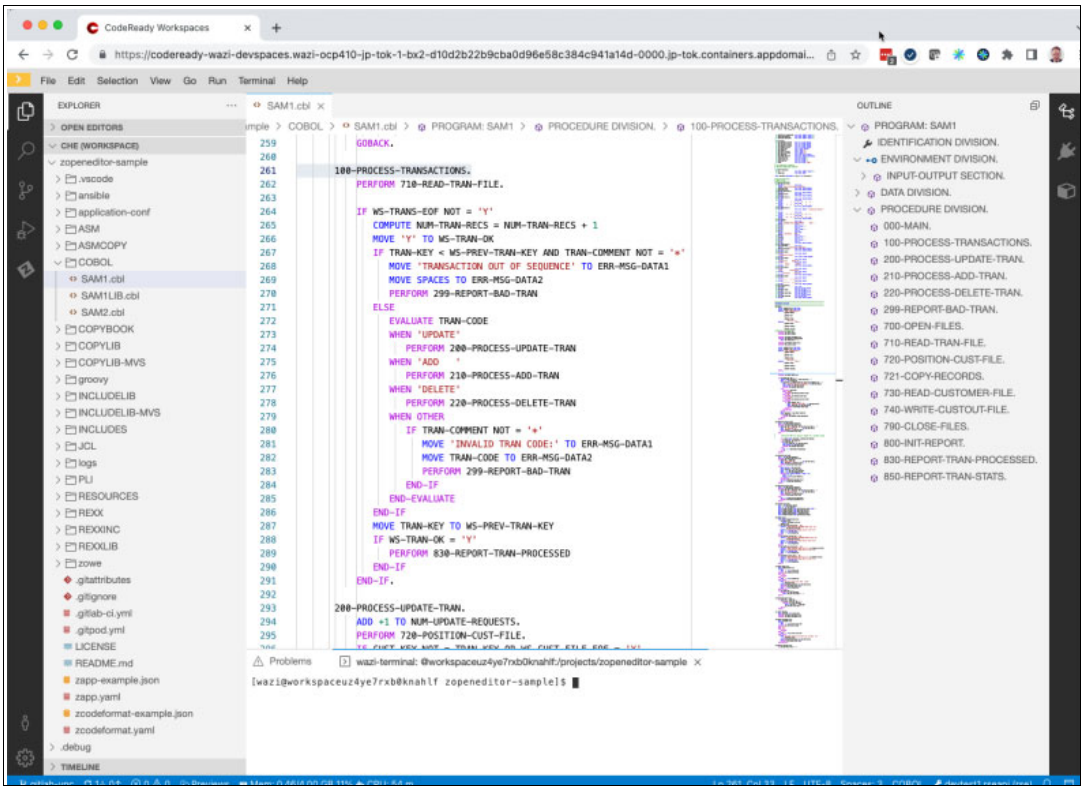


Figure 5-18 The Wazi for Dev Spaces COBOL editor.

Kathleen opens her first COBOL program from the sample repo and explores the editing features such the outline view on the right that she can use to quickly navigate the program. When she hovers the mouse over a variable, she sees its definition which she can right-click on the variable and navigate to that definition in the code or review other locations in the program where it is used. She can also write new lines using code completion by pressing Ctrl-Space to bring up choices for completing a command for the language she is working in. She also discovers that the code completion menu provides a large number of code templates that she can choose from. These templates offer standard development code patterns such as opening and writing to VSAM files or a CICS Handle Abends code snippet.

Next, she wants to be able make and build code changes on her team's Wazi as a Service VSI and then start a Debug session from there as well. For that the repository comes with

sample automation scripts written in Ansible (for configuring the new z/OS VSI) and Groovy (for running builds on z/OS with IBM Dependency Based Build).

To use the Ansible script, she needs to edit an Ansible configuration file that contains all the specific host variables needed for the execution. The Wazi for Dev Spaces sample repository already ships with configuration files prepared for the Wazi as a Service Development and Test image (as well as the Wazi Sandbox image). So all she has to do is provide the IP for the VSI and the user name that Todd enabled for her. Remember, that for the tutorial we did not set up a VPN, but a public IP address, so Todd needs to provide that address to the developers. He finds it in the VSI list of the VPC homepage in IBM Cloud.

Kathleen opens the file `ansible/inventories/inventory.yml` in the editor and edits the IP address for the `devtest` entry that is already listed there. She also replaces the user name with `ibmuser`. To her surprise, she sees that editing the file also comes with advanced features as the development stack she selected for her Wazi for Dev Spaces workspace comes with tools for editing Ansible and YAML.

Kathleen also opens the file `ansible/initialize-local-files.yaml` to review the first playbook Todd asked her to execute. She uses the languages selector in the status bar of the editor to switch the language to Ansible, which starts the Ansible language support. She now can use similar features that she observed for the COBOL editor, such as rich hovers that show the documentation of most Ansible commands, syntax highlighting and errors in the Problems view, as well as code completion.

She now opens the `wazi-terminal` from the Terminal menu and runs that playbook by entering the following commands:

```
cd ansible

ansible-playbook -i inventories --extra-vars "host=devtest1"
initialize-local-files.yaml
```

The command prompts for the following:

1. For the password of the user she specified in the configuration file and then generates several files.
2. For the version of Zowe CLI profiles that she wants to create. She confirm the default: v7
3. For the editor she is using. She enters `che` as she is in Wazi for Dev Spaces.
4. Asks if she wants to overwrite previously created files. She enters `yes`.

When the playbook has finished executing, she scrolls back up and carefully reviews the output. It contains instructions for the following tasks to complete the setup:

1. To finish the Zowe CLI setup, depending on how she answered the prompts. It tells her that it created a Zowe team configuration file for her host `devtest1` and placed it in `~/.zowe/zowe.config.json`.
2. To complete the setup, she runs the following command to complete the Zowe configuration:

   ```
   zowe config update-schemas
   ```

3. The script output also tells her that it created a Z Open Debug launch for her that she must copy into her workspace's *.vscode* folder.
4. Finally, the script printed JSON values that she needs to add to her IBM Wazi for Dev Spaces user settings to be able to run the Debugger. She uses the File > Settings > Open Preferences to open the Preferences panel, which she switches to the JSON view and pastes the provided JSON at the end (before the final closing brace).

As Todd has not yet gotten hold of Zach to configure the Debug with his company's signed SSL certificates, he decided to use the Debugger with self-signed certificates for the moment. He provided Kathleen and Deb with the instructions to execute these manually. He instructed them to open a new browser tab and go to the URL: `<https://vsi-ip-addres:8192/api/v1/remote-debug/config>` to accept the self-signed certificate into the browser so that Wazi for Dev Spaces can use it for connecting to the debugger.

Now she has configured a connection to her Wazi as a Service VSI using Zowe and configured the Debugger to work with their VSI. She tests her configuration by switching to the Zowe Explorer view in the editor. She then uses the USS view to connect to the VSI and look at ibmuser's home directory as `/u/ibmuser`. For more details on using the Zowe Explorer, see the following website:

https://www.ibm.com/docs/en/cloud-paks/z-modernization-stack/2022.3?topic=zowe-using-explorer-views

In the `/u/ibmuser` home directory Kathleen opens the file `.ssh/authorized_hosts` and pastes and saves her Wazi for Dev Spaces Workspace's SSH key, which she retrieved from the command menu via Ctrl-Shift-P (Windows) or Cmd-Shift-P (Mac) -> SSH: View public keys -> Default.

See the following web page for more details:
https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.14/html/end-user_guide/che-theia-ide-basics_crw#version-control_crw

Next, Kathleen wants to build and run a COBOL application. For that she needs to prepare the IBM Dependency Based Build (DBB) setup on the z/OS VSI. Its development and test image already comes preinstalled and preconfigured for DBB, so she really only has to configure her user account. The sample repository has another Ansible playbook for that.

She runs the playbooks with the following command:

```
ansible-playbook -i inventories --extra-vars "host=devtest1"
dbb-prepare-userbuild.yml
```

The script runs through fully automated. When it finishes, she reviews the output as well as the playbook's source code to understand what it did.

The output of the playbook provides again some JSON that must be copied into Kathleen's user settings. It contains user- and system-specific values that the editor will use to start a user build remotely on the VSI.

Kathleen also sees that the script cloned a Git repository remotely on USS into the folder `/u/ibmuser/projects` that contains the latest version of IBM's dbb-zappbuild with the build scripts needed for remote building from the editor. It also copied a configuration file into that repository. This configuration is specific to the development and test z/OS VSI system, listing the data sets for the compiler, libraries, and ports. As mentioned above, all the required values are provided for the Development and Test Stock Image in the Ansible inventory by the file ansible/inventories/host_vars/devtest1.yml.

After just running two Ansible playbooks for setup, Kathleen is now ready to start development.

5.5.4 Building, running, and debugging your application

After this quick setup, Kathleen can now immediately focus on working on an application. She needs to build it, run debug sessions to trace through its functions and her changes, and run it with test data to ensure any changes she applies do not break the application or introduce regressions.

To simplify development for enterprise applications and make it easy for Deb and Kathleen to perform these same operations, the team uses another set of Ansible playbooks that provide a very similar user experience as if Deb were to build, run, and debug a Java or JavaScript application.

So the team wants simple operations for the following:

1. Installing all the prerequisites and building all the programs of the application.
2. Uploading test data to the z/OS system and running the application to test its functionality.
3. Quickly building just the program currently being edited to ensure it compiles.
4. Launching a debug session of the program that was built to step through its functions.

Wazi for Dev Spaces and the sample repository have Ansible playbooks and editor launches for exactly these operations. In our example, the following steps occur.

1. Kathleen runs the playbook by using the following command to build the entire application. This playbook will also generate and upload JCL files to the VSI for running and debugging.

```
ansible-playbook -i inventories --extra-vars "host=devtest1" dbb-sam-build.yml
```

2. Kathleen runs another playbook by using the following command to upload test data and run the application against that test data. The output of the playbook shows the downloaded application's output to check the results.

```
ansible-playbook -i inventories --extra-vars "host=devtest1" dbb-sam-run.yml
```

3. Kathleen edits the program `COBOL/SAM1.cbl` that she had opened earlier. She then performs a right-click operation from within the source code and selects "Run IBM User Build" (the first time she runs it, she needs to select "Run setup for IBM User Build"). This starts a user build session. The editor opens a new output view and shows the log for uploading the program and its local copybook dependencies and starting a DBB build using the dbb-zappbuild repository's scripts.
4. After the build succeeds she can start a Debug session. For that she uses Zowe Explorer to find the DEBUG JCL file that was uploaded by the build playbook to `IBMUSER.SAMPLE.JCL`. She right-clicks the file to start the session via the Submit command. She then switches to the editor's Debug view and selects the launch that was created by the initialize playbook that she ran at the beginning. She presses the Play button and the Debug session starts in the editor. She can set breakpoints and examine the value of variables. The user experience is the same as debugging Java or TypeScript.

5.6 Summary

This concludes our review of the Enterprise DevOps Enabler Patterns and the IBM technology portfolio around them. We walked you through many important best practices for DevOps and how they apply to hybrid development projects. We then explored key tools and technologies that help enable development teams to realize these best practices. We presented them as a set of layers that build on top of each other while also recounting the

history of IBM Z DevOps, Wazi, and IBM Z and Cloud Modernization Stack along the way. We then gave you two end-to-end examples that use these practices and technologies. One from the development team's process point of view and one technical hands-on tutorial that you can try out yourself.

For additional information that summarizes many of the core concepts of this chapter, as well as to see the end-to-end demo from the tutorial section in action, see the following video:

https://www.youtube.com/watch?v=8ZOGPN7Ld2w

6

Managing your applications

This chapter discusses the implications related to application monitoring and management aspects when running in a Hybrid Cloud architecture. Additionally, we discuss Red Hat OpenShift implementations on IBM zSystems.

6.1 Monitoring, logging and metering introduction

Monitoring systems, applications and IT infrastructure components is an essential step to achieve high standard service levels. Being able to predict and quickly respond to failure, fix issues rapidly, and understand resource utilization is crucial.

This can be achieved by applying the following different techniques:

- Monitoring: comprises of strategies and practices for analyzing, tracking and managing services and applications, allowing systems administrators to maintain visibility of the performance of their IT assets.
- Logging: In a Hybrid Cloud environment, logging refers to the ability of capture and aggregate logs from various applications and services. Ideally it comes coupled with Analytics tools that allow administrators and operations teams to have insights of the overall systems health.
- Metering: refers to the ability of collecting metrics that allow administrators, operations teams and users to understand usage of applications, services and IT resources.

6.2 Components of the Red Hat OpenShift monitoring stack

Red Hat OpenShift Container Platform includes a preconfigured, preinstalled, and self-updating monitoring stack that provides monitoring for core platform components. The Red Hat OpenShift Container Platform monitoring stack is based on the Prometheus open source project and its wider ecosystem. The monitoring stack includes the following:

- **Core platform monitoring components:** A set of platform monitoring components are installed in the openshift-monitoring project by default during an Red Hat OpenShift Container Platform installation. This provides monitoring for core Red Hat OpenShift Container Platform components including Kubernetes services. The default monitoring stack also enables remote health monitoring for clusters. These components are illustrated in the “Installed by default” section of Figure 6-1.
- **Components for monitoring user-defined projects:** After optionally enabling monitoring for user-defined projects, additional monitoring components are installed in the openshift-user-workload-monitoring project. This provides monitoring for user-defined projects. These components are illustrated in the “User” section of Figure 6-1.

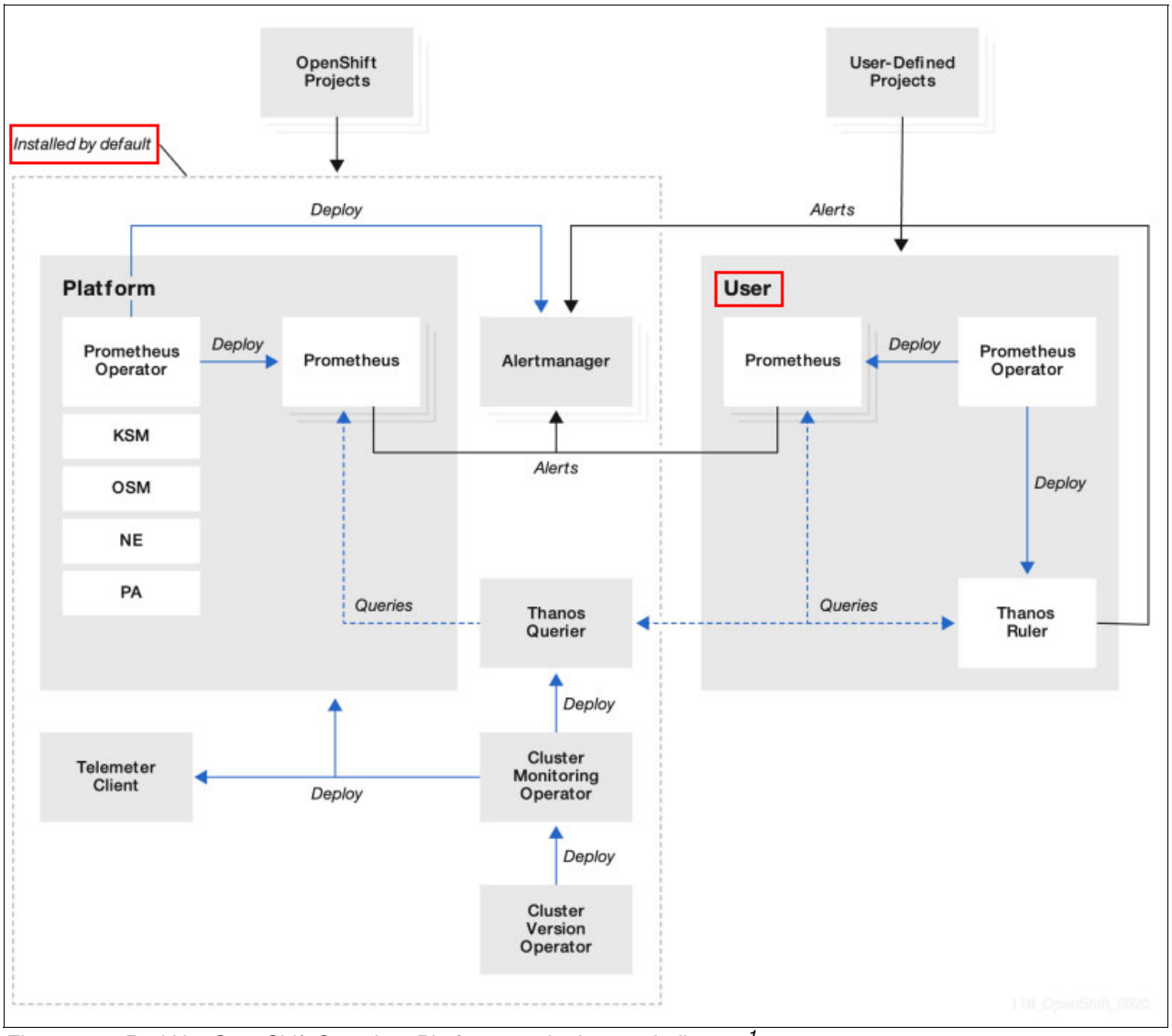


Figure 6-1 Red Hat OpenShift Container Platform monitoring stack diagram[1]

Ideally, you should configure persistent storage for your running Red Hat OpenShift cluster to store that data into a persistent volume(PV) so it can survive a pod being restarted or recreated. Because Prometheus has two replicas and Alertmanager has three replicas, you need five PVs to support the entire monitoring stack. Depending on the persistent storage solution in place, all these PVs will be dynamically provisioned, for example: Using Red Hat Data Foundation will provide that capability.

Monitoring user defined workloads is beyond of the scope of this introduction, but to contrast user defined projects versus standard monitoring of components of a particular project/namespace, consider the following example: A PostrgeSQL database is deployed in a project named Database, and uses the standard monitoring capabilities of the built-in monitoring stack, one can check the amount of cpu, memory, network usage, etc.

The user defined monitoring capability described by the Red Hat documentation means that extended monitoring capabilities can be added, such as the number of connections to the database. This extends the monitoring capabilities beyond the out-of-the-box capabilities. To

[1] https://docs.openshift.com/container-platform/4.11/monitoring/monitoring-overview.html

enable monitoring of user defined projects see the following website:
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.11/html-single/monitoring/index#preparing-to-configure-the-monitoring-stack

6.2.1 Monitoring Red Hat OpenShift Container Platform infrastructure with Prometheus

This section covers the built-in monitoring stack that comes with Red Hat OpenShift Container Platform.

Red Hat OpenShift Container Platform provides a preconfigured and self-updating monitoring out of the box. You can monitor the platform's core components as well as user-defined projects. By using the user-defined monitoring, cluster administrators, developers and other types of users can specify how services and pods are monitored for each project (also known as namespace).

Red Hat defines monitoring of user-defined workloads as monitoring the number of connections for a database, or any other specific characteristic of that workload that does not meet the default items covered by the out of the box monitoring stack.

This section does not replace the official Red Hat OpenShift Platform documentation, for more information, see the following website:
https://docs.openshift.com/container-platform/4.11/monitoring/monitoring-overview.html

As a summary, the Red Hat OpenShift Container Platform web console provides a way to view and manage metrics, alerts and access the monitoring dashboards.

Besides the built-in monitoring dashboards, Red Hat OpenShift Container Platform also offers third-party interfaces such as Prometheus, Alertmanager and Grafana.

6.2.2 Using the Red Hat OpenShift Container Platform web console's dashboard to monitor your cluster and customer workloads

In this section, we explore the Red Hat OpenShift Container Platform web console's dashboard that is used to monitor your cluster and customer workloads.

Exploring the Overview dashboards as the cluster administrator

The Overview Dashboard (shown in Figure 6-2) provides information such as Alerts, as well as the status of the cluster, control plane nodes, the status of operators and if your cluster is deployed as a connected cluster. You can take advantage of the Insights capability that helps prevent issues on your environment by tapping into a huge database of information from Red Hat and leveraging the AI capability to create alerts of potential issues in your environment.

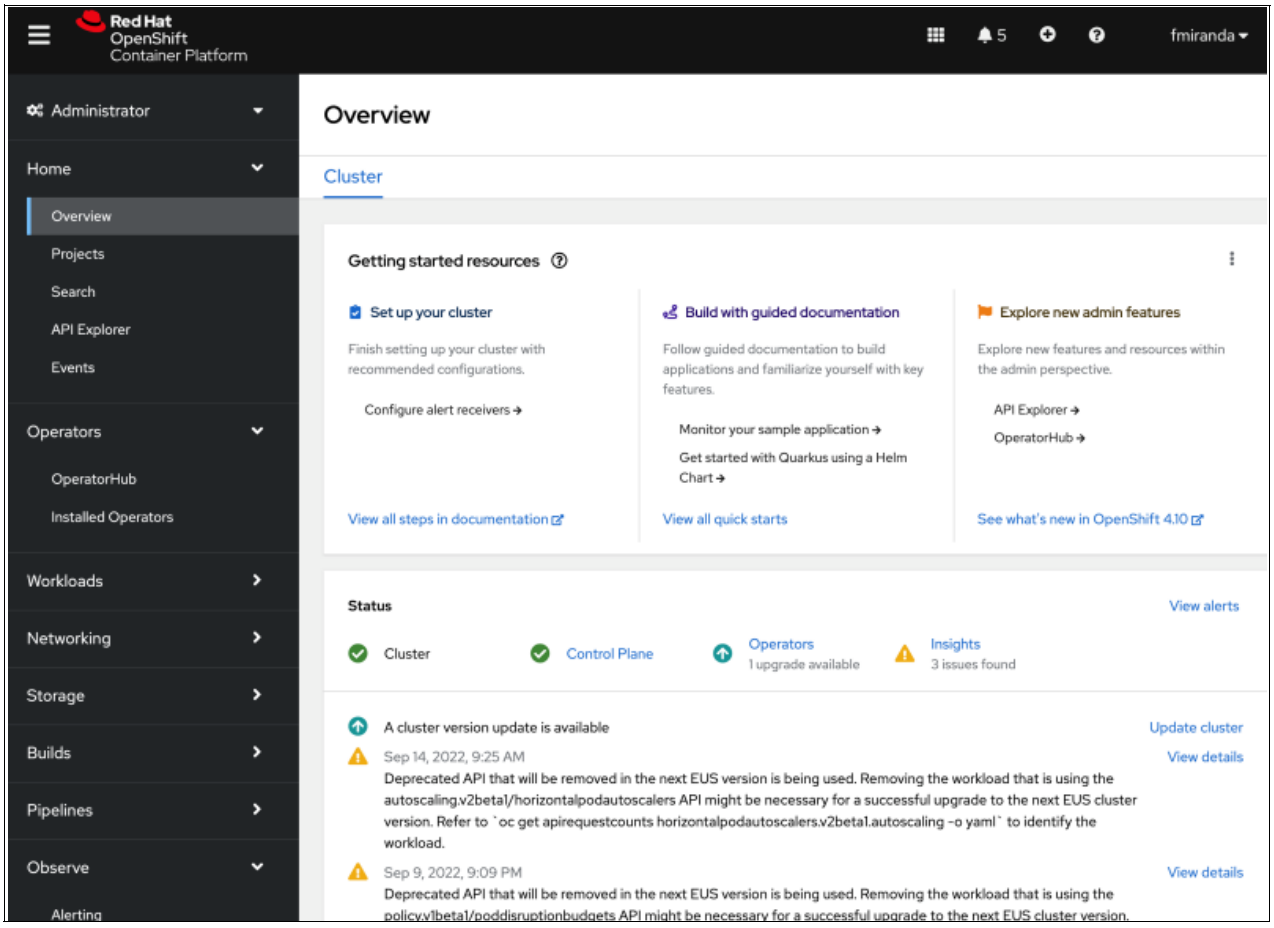


Figure 6-2 Red Hat OpenShift Dashboard Web Console Overview

Scrolling down through the Overview dashboard screen, there is real time monitoring of your Red Hat OpenShift cluster on total utilization of CPU, Memory, Filesystem, Network and Pod count as shown in Figure 6-3.

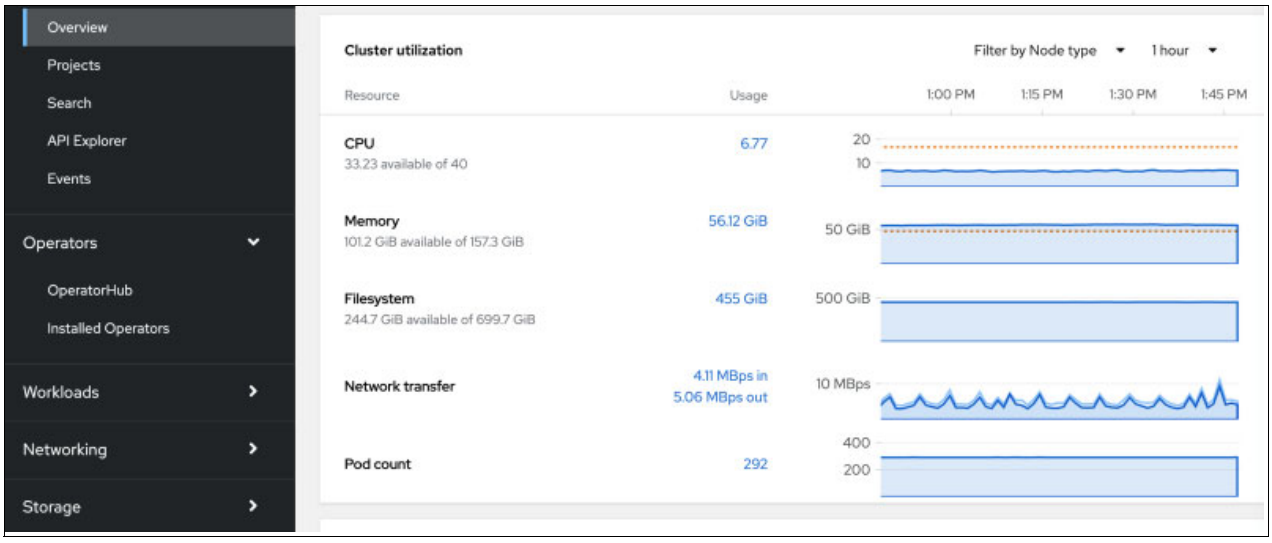


Figure 6-3 Red Hat OpenShift Dashboard web console real time monitoring

By default, historic monitoring data is kept for about 15 days. If no persistent volumes are used for the monitoring stack, data will be saved into the local node where the stack is running. Data is lost if the pod is lost or if the node is unavailable - this is the reason we recommend adding a persistent storage layer to your Red Hat OpenShift Cluster, so no historical data is lost.

To change the amount of time that historic monitoring data is kept, see section 2.9.3 on the web page found at the following website:
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.11/html-single/monitoring/index#modifying-retention-time-and-size-for-prometheus-metrics-data_configuring-the-monitoring-stack

Further down on the Overview screen, you can check the Activity of the cluster, where it explores the most recent events from this cluster, as shown in Figure 6-4.

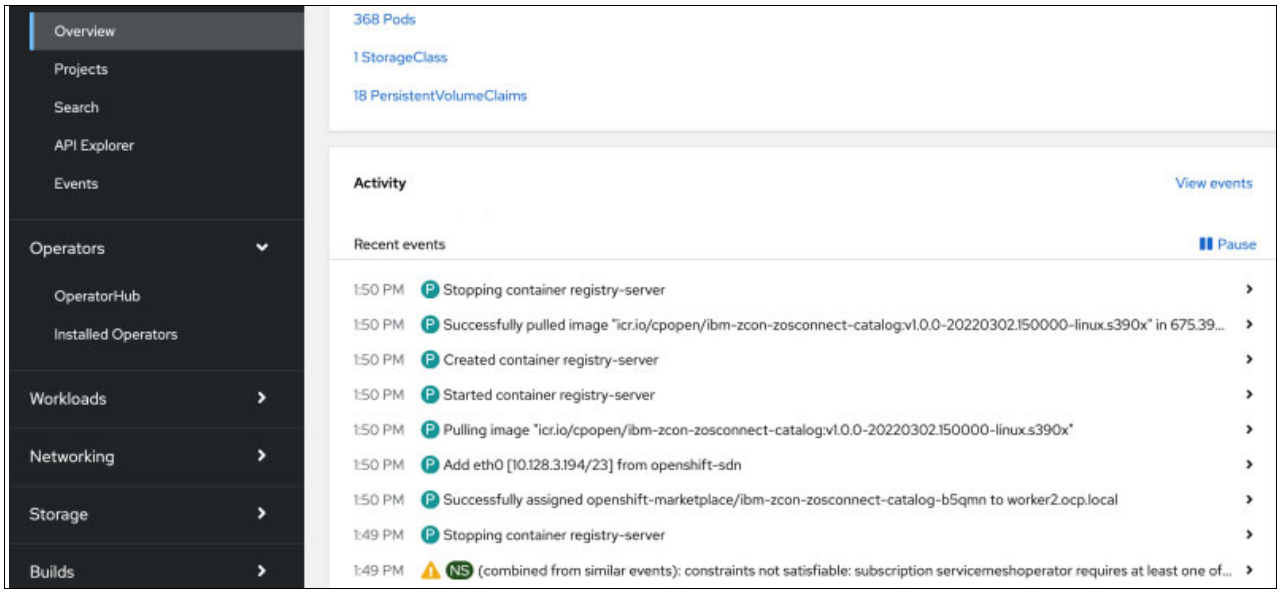


Figure 6-4 Red Hat OpenShift Dashboard Web Console real time monitoring

6.2.3 Exploring the default Alerting system

Moving from the Overview section to the Observe menu will provide us access to the Alerting mechanism of Red Hat OpenShift as well as other dashboards that show resources being monitored.

Figure 6-5 is the Alerting mechanism in Red Hat OpenShift.

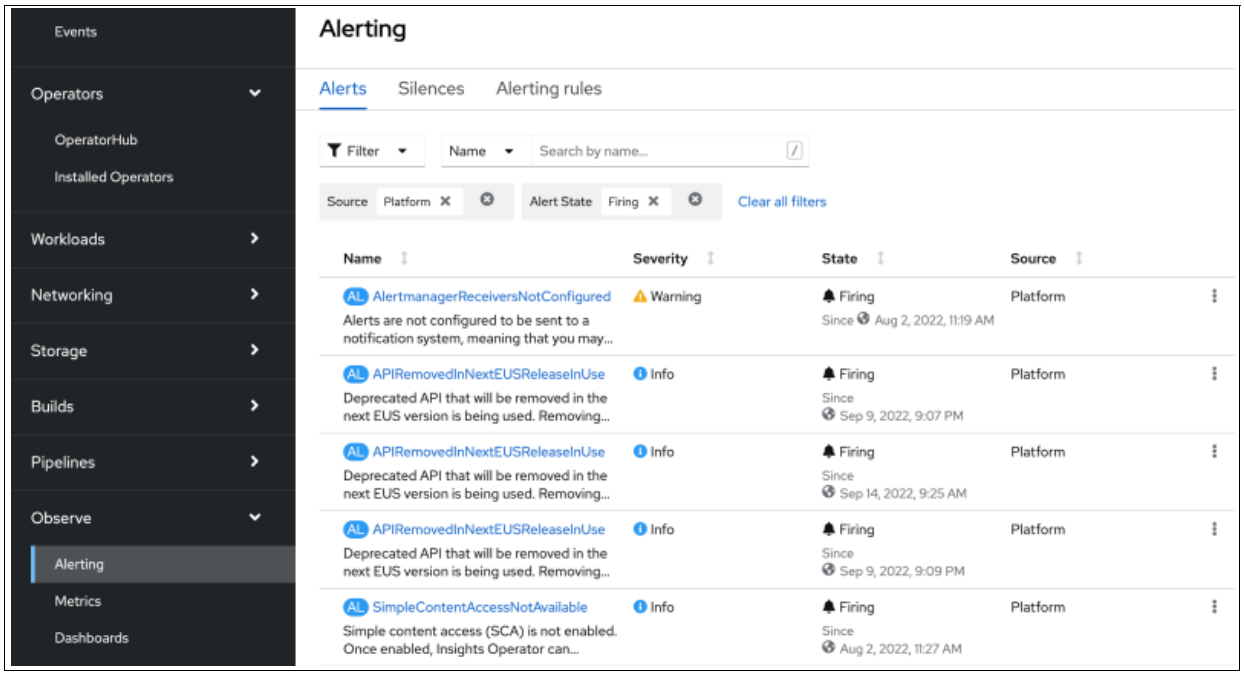


Figure 6-5 Red Hat OpenShift Alerting feature

Alerts can be used to monitor certain condition of the cluster as well as aspects of user defined workloads. Configuration of a new alert is beyond the scope of this section but we will show you an example of a default alert to track, for example, new versions of a software stack for Red Hat OpenShift Container Platform. Figure 6-6 shows an alert to inform the administrator that there is an update available to the Red Hat OpenShift cluster. This way, with connected or disconnected deployments of Red Hat OpenShift, the system administrator will receive an alert stating whenever there is an update available to this particular cluster.

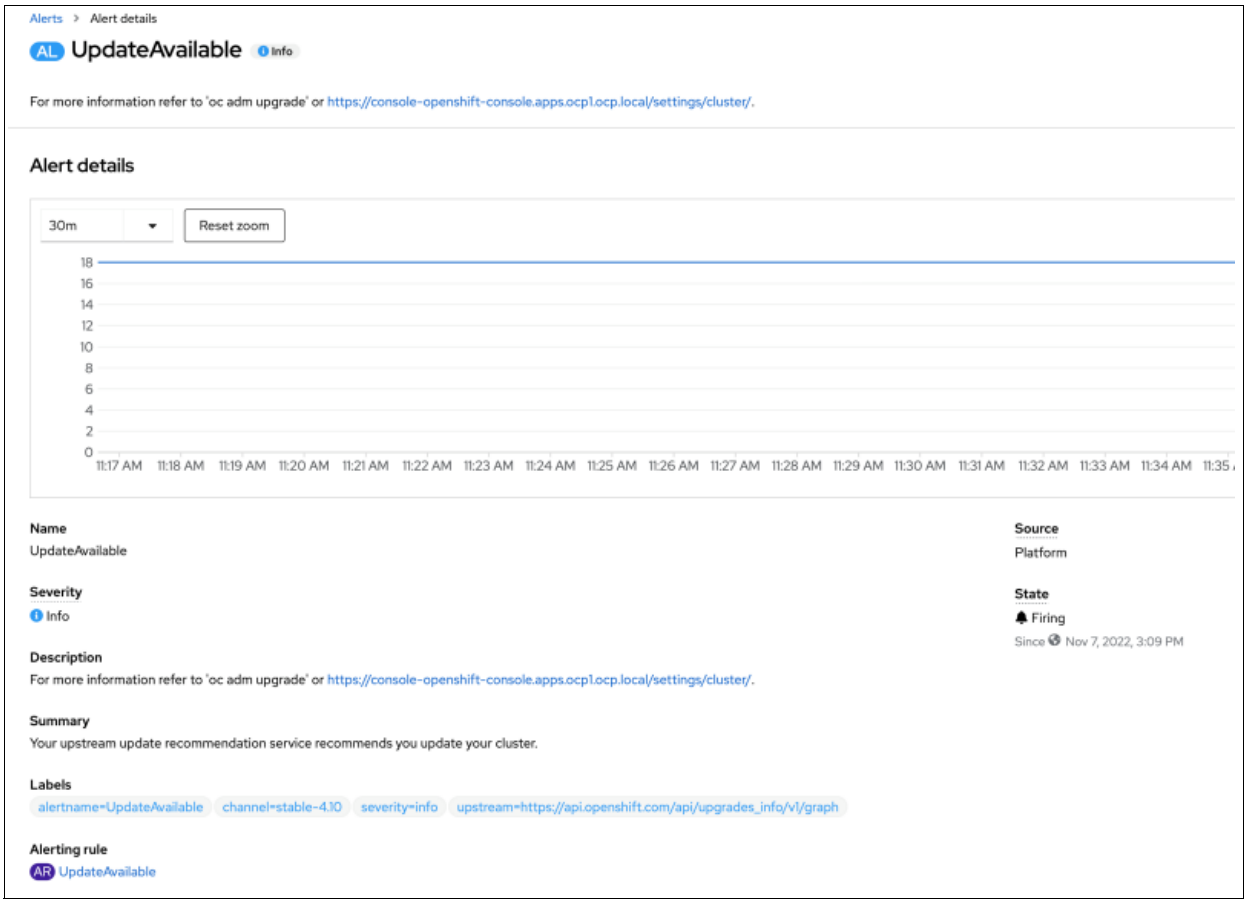


Figure 6-6 Red Hat OpenShift Alerting example

In this cluster, as an example, this alert was triggered, and it will show in the Red Hat OpenShift web console in the bell icon at the top right of the screen (Figure 6-7).

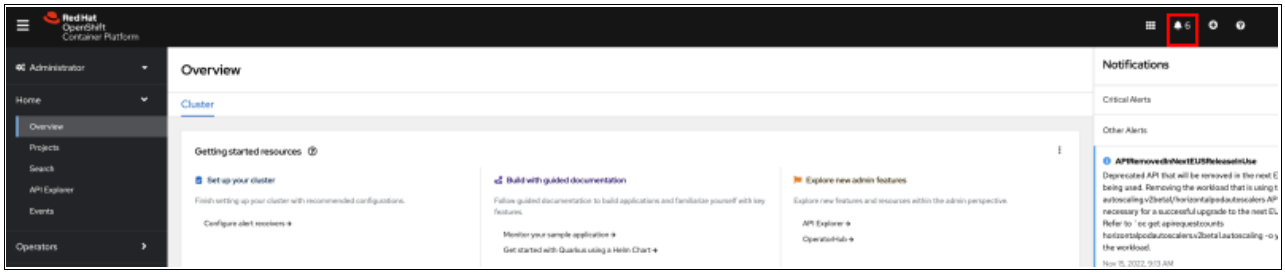


Figure 6-7 Red Hat OpenShift Notification bell

In Figure 6-7, there are six (6) alerts triggered, if we scroll down the list, there is a cluster update available shown, as shows in Figure 6-8.

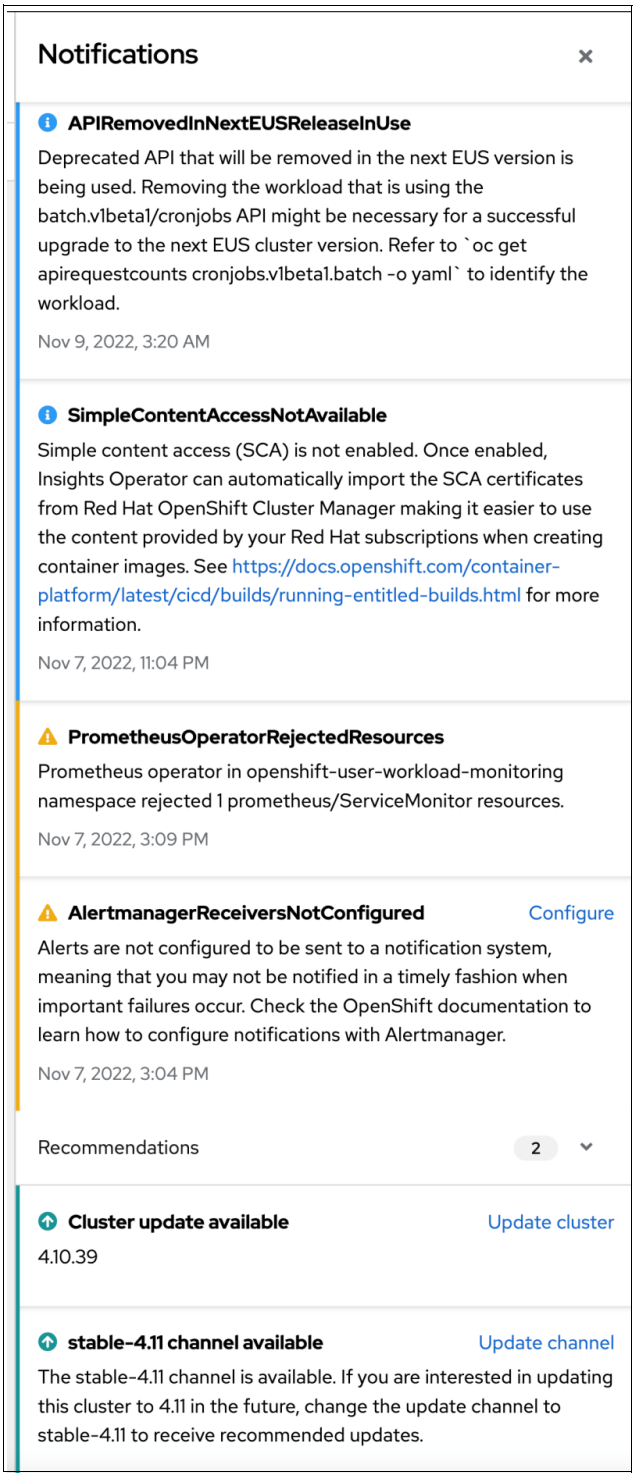


Figure 6-8 Red Hat OpenShift notifications expanded

6.2.4 Explore cluster monitoring data from different sources, such as cluster nodes, projects, or pods

Selecting the option, Dashboards, from the Observe menu of the Red Hat OpenShift web console, a dropdown menu offers many options to show monitoring data about the Red Hat OpenShift cluster, as Figure 6-9 shows.

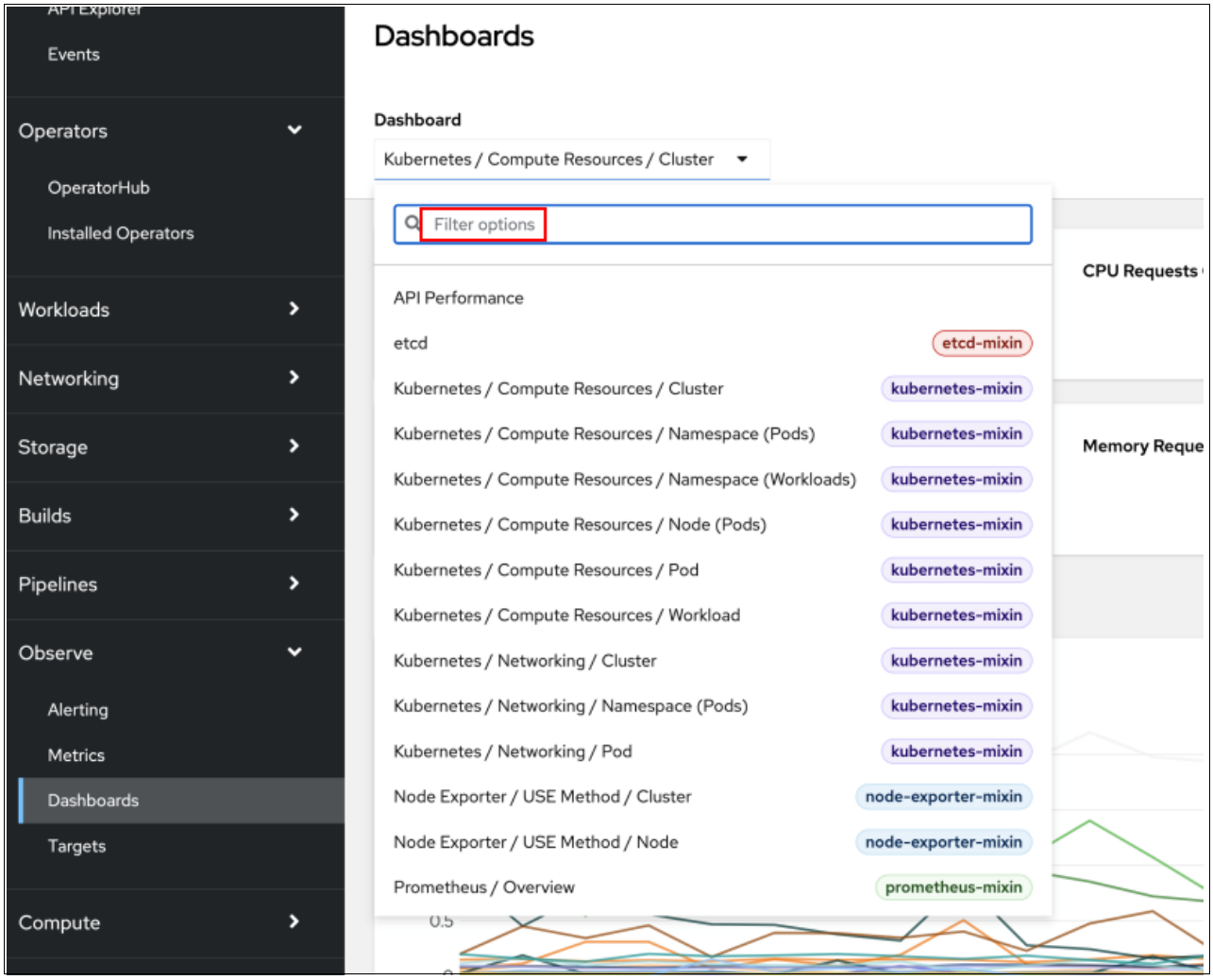


Figure 6-9 Red Hat OpenShift Observe dashboard

For demonstration purposes, a few options will be selected from that list of items. Starting with Kubernetes /Compute Resources / Namespaces (Pods) as shown in Figure 6-10. It shows monitoring information about a user workload (not user defined monitoring information) about a specific namespace (project). Inside this project (pdf-voting-app-demo) we have an application that is composed of several microservices, and this dashboard shows what each pod inside this namespace is consuming such as CPU, memory, network usage, etc.

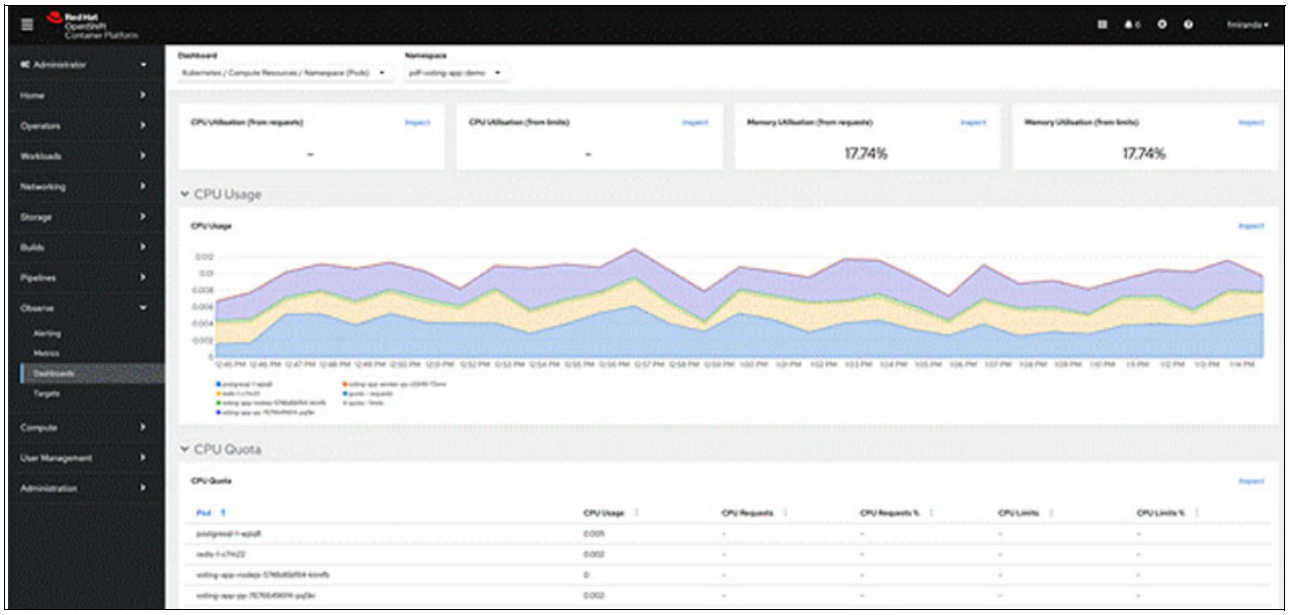


Figure 6-10 Red Hat OpenShift dashboard namespace pod monitoring

A different perspective for the same collection of microservices can be shown if the option Kubernetes /Computer Resources / Workload is selected as Figure 6-11 shows.

By using this option, you can select the Namespace, the Type of Deployment and Workload (in this case the different microservices of this application).

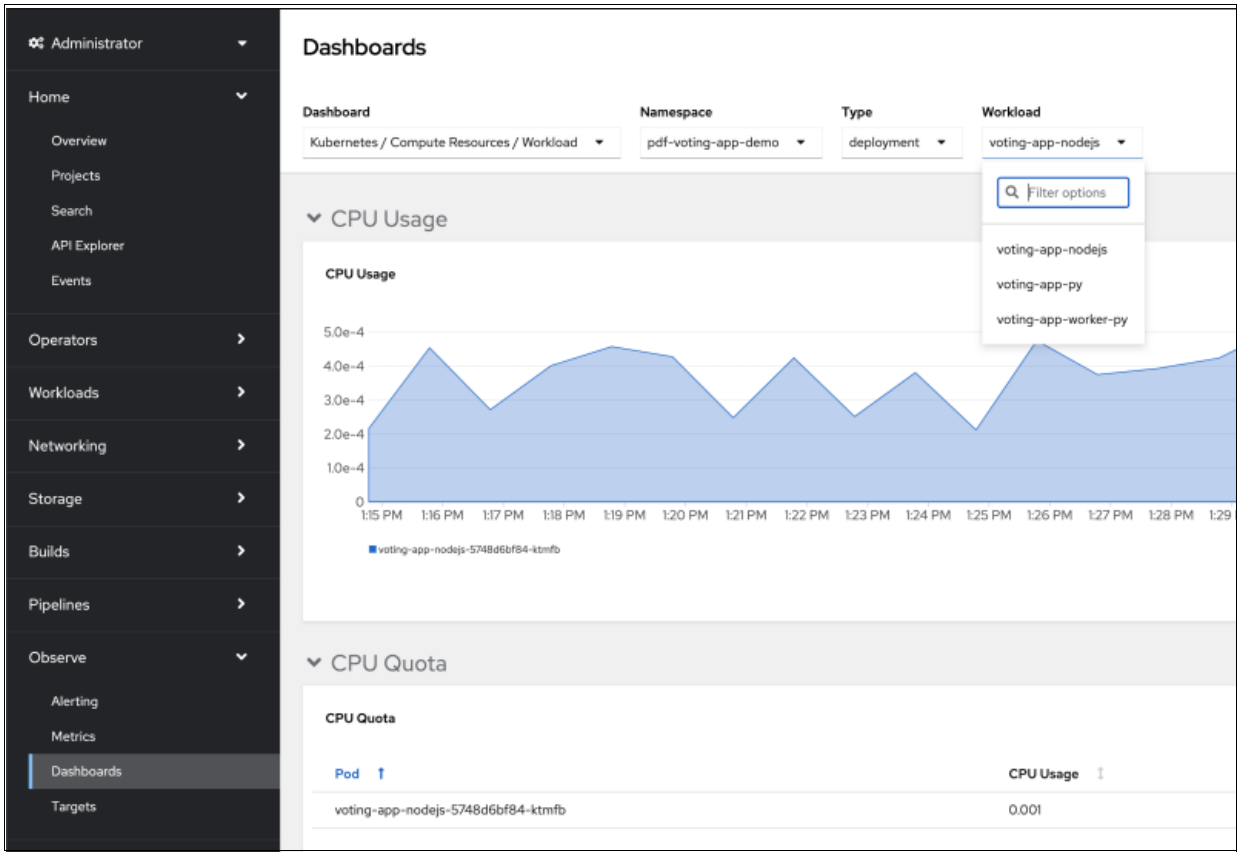


Figure 6-11 Red Hat OpenShift dashboard namespace, deployment type and workload monitoring

Figure 6-12 shows the same dashboard, but with a different Deployment Type (this collection of microservices uses two types of deployments). Both Figure 6-10 and Figure 6-12 provide mode details about resource consumption per workload inside that specific namespace.

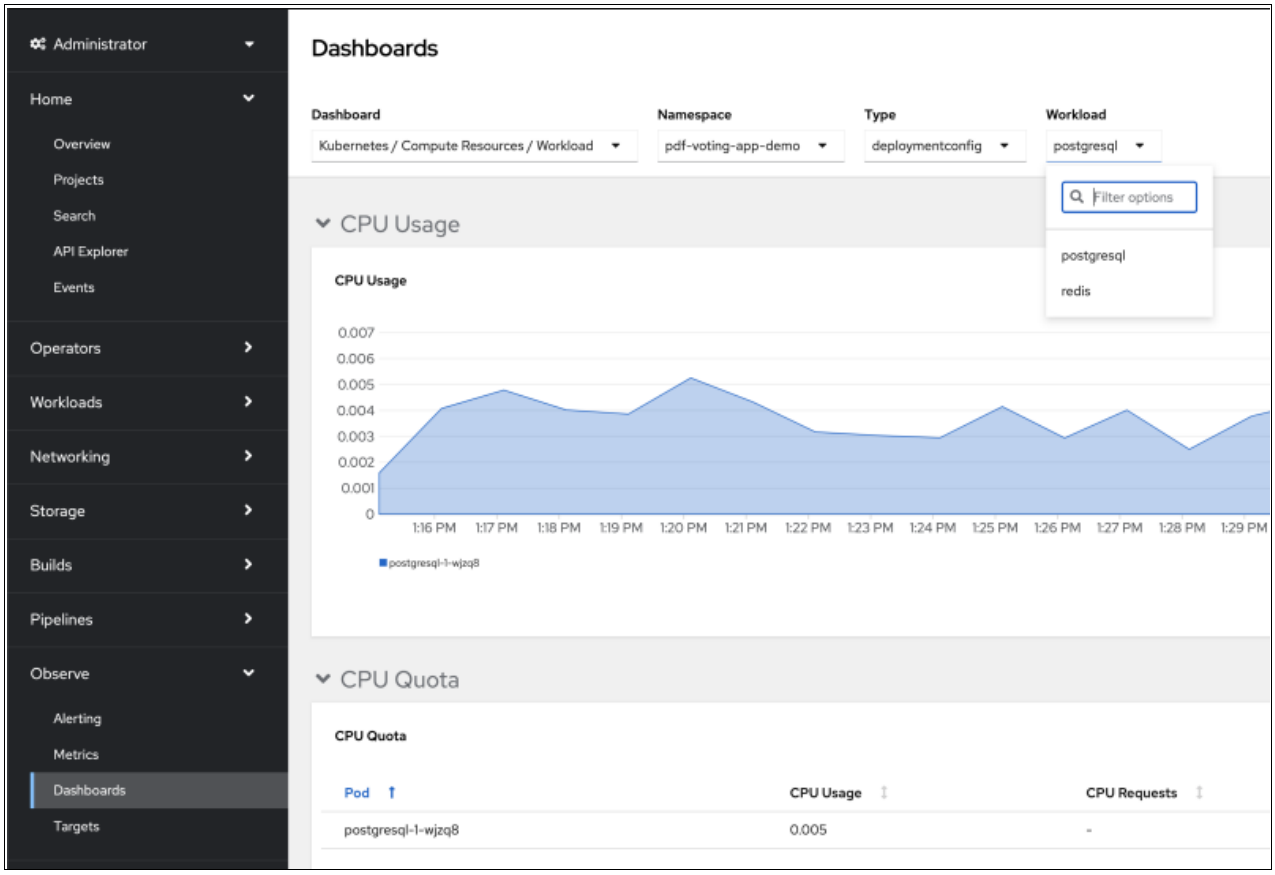


Figure 6-12 Red Hat OpenShift dashboard different deployment types example

6.2.5 Using the oc client tool to monitor resources

As a last set of examples, a little bit of command line for the command line heroes. We now dive into how to obtain the same information from the command line option by using the **oc** client tool. For more details on this tool, see the following website:
https://docs.openshift.com/container-platform/4.11/cli_reference/openshift_cli/getting-started-cli.html

Once the **oc** client tool is installed on your client workstation, you can manipulate your cluster remotely.

Example 6-1 demonstrates how to monitor specific namespaces by using the **oc** client tool.

Example 6-1 oc adm top pod -n <namespace> command

```
$ oc adm top pod -n pdf-voting-app-demo
NAME                       CPU(cores)   MEMORY(bytes)

postgresql-1-wjzq8         3m           39Mi
redis-1-c7m22              2m           14Mi
voting-app-nodejs-5748d6bf84-ktmfb   0m           50Mi
voting-app-py-76766496f4-pq5kr       3m           56Mi
```

```
voting-app-worker-py-c5949-72vnn      0m           20M
```

Example 6-2 shows how to monitor all namespaces in your Red Hat OpenShift Cluster by using the **oc** client tool.

Example 6-2 oc adm top pod --all-namespaces command

```
$ oc adm top pod --all-namespaces
NAMESPACE        NAME        CPU(cores)   MEMORY(bytes)
…
pdf-voting-app-demo   postgresql-1-wjzq8      3m           39Mi
pdf-voting-app-demo   redis-1-c7m22           2m           14Mi
voting-app-nodejs-5748d6bf84-ktmfb            0m           50Mi
pdf-voting-app-demo   voting-app-py-767664…   3m           56Mi
pdf-voting-app-demo   voting-app-worker-p…    0m           20M
openshift-multus      network-metric…         0m           47Mi
…
```

You can monitor resources consumed by nodes on the Red Hat OpenShift cluster with the **oc** client tool, as shown in Example 6-3.

Example 6-3 Monitor resources used by nodes

```
$ oc adm top node
NAME      CPU(cores) CPU% MEMORY(bytes) MEMORY%
infra1.ocp.local     1478m        42%    4001Mi          26% master0.ocp.local
1557m        44%    12825Mi         85% master1.ocp.local     768m         21%
7740Mi          51%
master2.ocp.local    1262m        36%    8557Mi          57% infra0.ocp.local
2243m        29%    10007Mi         32% worker1.ocp.local     503m         6%
7805Mi          25%
worker2.ocp.local     579m         7%    6962Mi          22%
```

This concludes our simplified overview of the Red Hat OpenShift Container Platform built-in monitoring stack. For more complete information refer to the official Red Hat OpenShift documentation found on the following website:
https://access.redhat.com/documentation/en-us/openshift_container_platform/4.11/html-single/monitoring/index#about-openshift-monitoring

6.2.6 Using RMF to monitor z/OS resources for Red Hat OpenShift Container Platform

In this IBM Redbooks publication, the Red Hat OpenShift Container Platform was implemented on top of z/OS. As such, the z/OS Workload Manager (WLM) is used to control the assignment of resources to the Red Hat OpenShift cluster, as with any other started task.

When using z/OS Container Extension (zCX) to run Red Hat OpenShift components and workloads, there are specific z/OS Workload Manager (WLM) definitions to be made as part of the Red Hat OpenShift implementation, such as Service Classes. Guidance and implementation steps to configure WLM policies are described in the following web site for the zCX Foundation for Red Hat OpenShift:
https://www.ibm.com/docs/en/zcxrhos/1.1.0?topic=openshift-workload-management-considerations

After the implementation, monitoring resource utilization and adjusting definitions is equally important. Standard z/OS monitoring tools, such as z/OS Resource Measurement Facility (RMF) can be used to monitor Red Hat OpenShift related zCX started tasks, to ensure velocity goals are being achieved and CPU consumption for general purpose processors and zIIP offload engines are within the desired levels.

Monitoring zCX started tasks with RMF is discussed in more detail in Chapter 7.5 *Monitoring with RMF on zCX instance level* of the *Getting started with z/OS Container Extensions and Docker*, SG24-8457 IBM Redbooks publication.

6.3 Observability on z/OS

In a hybrid cloud architecture, it is important to monitor applications and all of their involved infrastructure components from a single tool. Monitoring refers to being able to assess, based on logs, events and traces, whether an application and/or its components are available, healthy, malfunctioning, heavily utilized, etc. However, when applications involve integration between several different layers and services, that may be running in various platforms or clouds, it may become challenging to understand if service levels, end-to-end, are being met, or to predict and prevent failures.

To overcome this challenge, new techniques are applied to cloud native applications and microservices, to complement traditional monitoring tools. As such, being able to measure the state of applications end-to-end as well as in each of its processing phase, over time, is known as *Observability*.

This can be accomplished by continuously analyzing logs, events, metrics and trace data, recording success or failure, individual response times and other telemetry data, and generating insights that can trigger automated recovery actions, for example.

Due to existing requirements of availability, there is a shift to AI-Driven observability, which brings specific metrics with it:

- MTTD - Mean Time To Detect describes how long it takes to detect incidents or issues.
- MTTP - Mean Time To Prevention describes how long it takes to automatically prevent incidents or issues from negatively impacting application performance and users.
- MTTN - Mean Time To Notify refers to the amount of time it takes to raise alerts to relevant teams about incidents or issues.
- MTTR - Mean Time To Repair is used to describe the amount of time taken during automated, semi-automated or manual remediation of incidents or issues.

6.3.1 Instana on IBM z/OS

One of the leading observability tools available in the market and made available to cover z/OS components is IBM Instana® APM. It is an enterprise observability solution that automatically makes your applications and services visible, providing context to that observed information. It then enables you to take intelligent action based upon that information. Instana monitors and analyzes your applications, services, infrastructure, web browsers, mobile applications, and more.

With Instana, you will instantly know if any of your customers are impacted by performance or stability issues in your applications, within a few seconds of impact. Instana provides a graphical user interface to guide you to the root cause in just a few clicks.

Some key capabilities of Instana on IBM z/OS are as follows.

- It provides end to end observability for applications from mobile through mainframe, in a single solution
- Combines the industry's leading capabilities from the Instana observability platform, which are automation, context and intelligent action together with the world's most powerful enterprise data processing hub, the IBM mainframe.
- It does not replace traditional z/OS tools, but allows Operations teams to form an application point of view, quickly determine and point exactly where there is an issue, such as unavailability or slowdown, is coming from.
- Smooth operational visibility from all application intersections, including application dependency maps for every application, their flows and calls.

Note: At the time of this writing, the Instana backend on-premises installation is only supported on certain Linux distributions running on x86/64bit processors, as documented in the https://www.ibm.com/docs/en/instana-observability page. The IBM Statement of Direction on this topic an be found at the following website: https://www.ibm.com/common/ssi/ShowDoc.wss?docURL=/common/ssi/rep_ca/7/897/ENUS222-177/index.html

IBM's Statement of Direction is subject to change to support on-premises installation of Instana backend servers running on Linux on IBM Z. For more details, reach out to your IBM Sales representative.

6.4 Logging

Red Hat OpenShift Logging aggregates all the logs from Red Hat OpenShift Container Platform cluster such as node system audit logs, application container logs, and infrastructure logs. Red Hat OpenShift Logging stores them in a default log store where Kibana web console can be used to visualize log data.

Red Hat OpenShift Logging aggregates the following types of logs:

- Application - Container logs generated by user applications running in the cluster, except infrastructure container applications.
- Infrastructure - Logs generated by infrastructure components running in the cluster and Red Hat OpenShift Container Platform nodes, such as journal logs. Infrastructure components are pods that run in the openshift*, kube*, or default projects.
- Audit - Logs generated by the node audit system (auditd), which are stored in the /var/log/audit/audit.log file, and the audit logs from the Kubernetes API server and the Red Hat OpenShift API server.

6.5 Metering

Metering is a deprecated feature starting with Red Hat OpenShift 4.7. Deprecated functionality is still included in the Red Hat OpenShift Container Platform and continues to be supported; however it will be removed in a future release of this product and is not recommended for new deployments.

7

Deployments of production applications

This chapter builds on the modernization patterns introduced in Chapter 1, “Introduction” on page 1. We will focus on the delivery of an app using the application centric modernization pattern to production with a chosen production deployment strategy. The Application centric modernization pattern is used to enhance functions by developing cloud native functions to extend or enhance an application.

You will be provided a sample application, implementation details, and implementation best practices.

In this chapter, we discuss the following:

7.1 Production deployment strategies

Production deployment strategies involve delivering an application from development to the production environment, where it is consumed (used) by end-users. There are various factors that inform the choice of deployment strategy which includes; application architecture, business requirements, size of change and impact to application end-users.

The following are common deployment strategies.

- Recreate deployment

This deployment terminates the current version of an application and recreates a new version. This deployment type requires a maintenance window which incurs downtime to end-users. If there are any issues with the application, another maintenance window is required to roll back changes.

- Blue/Green deployment

This deployment has two releases deployed in production. The current version and the new version are both functional and end-user traffic is switched from the current version to the new version. This deployment type requires almost no maintenance window and offers faster rollbacks if major issues are discovered in the new version. Although, having two releases in production means it is costlier as more resources are required.

- Rolling deployment

This involves the incremental roll out of a new version to replace the current version of an application. This deployment type requires almost no maintenance window depending on the size of deployment. While a rollout or rollback is happening, there is no control over which application version accepts traffic.

- Canary deployment

This is used to expose a new version to a small subset of end-users. Using weights, traffic is partitioned to multiple versions of an application running concurrently in production. For example, a stable release and canary release is deployed in production with 90% traffic going to the stable release and 10% going to the canary release. This deployment provides the capability of testing features in production with minimum impact.

We will be using the canary deployment strategy in the upcoming sections

- A/B Testing deployment

This deployment is similar to the canary deployment strategy but has fine grained controls on the subset of end-users that have access to a new feature. It uses statistical evidence from observing and monitoring end-user engagement to make a decision if a new feature is viable. This deployment has full control on traffic distribution but is more complex to implement

7.2 Expose on-premises applications via a public cloud

In the section, we will be preparing an on-premises application deployed to Red Hat OpenShift on IBM Z to accept traffic from the internet via an IBM Cloud. You will learn how to connect your on-premises application to the IBM Cloud in a secure fashion. This is the building block of our canary style deployment in section 7.3, “Extend on-prem App with application in Public Cloud” on page 138.

7.2.1 Prerequisites

The following are the required components four our application deployment.

- Red Hat OpenShift cluster and registry that is provisioned and the developer has sufficient access permissions (roles), on the cluster.
- If the cluster is on-premises or private network, the user is connected to the network via a VPN.
- Podman is installed on the developer machine.
- Client tools to access the cluster from a command line are installed on the developer machine. Installation instructions can be found on the following website: https://docs.openshift.com/container-platform/4.11/cli_reference/openshift_cli/getting-started-cli.html
- The server certificate is installed on the developer machine, to enable secured connection. You will need to obtain a server certificate from your cluster administrator.
- Access to github.com to access sample Golang source code.
- An IBM Cloud account with Identity and Access and Management (IAM) services to Virtual Private Cloud (VPC) infrastructure services.
- The domain or subdomain for delegation to the IBM Cloud.

7.2.2 Current architecture

In our lab environment scenario, we have an application named *Supply Chain App* deployed on-premises with local access to a database. The only means to use the application is by being physically present in the corporate network or using a VPN to access the corporate network. The CIO will likely make the application available to business partners. A high level overview of this architecture is shown in Figure 7-1.

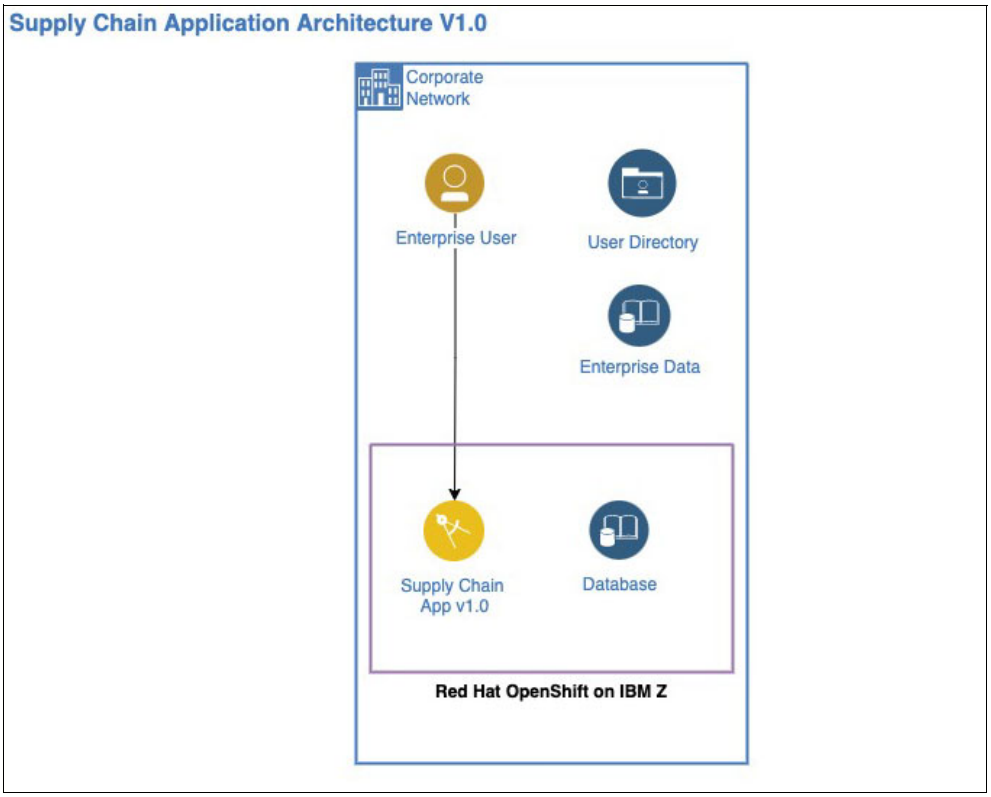


Figure 7-1 Current architecture

7.2.3 Target Architecture

The business wants to allow access via the Internet and also enable business partners to access the application in a secure fashion. We will be making use of the IBM Cloud to deploy the front door components and perform a network integration via a Site-to-Site VPN Gateway as well as using Internet Protocol Security (IPsec). And overview of our architecture is shown in Figure 7-2.

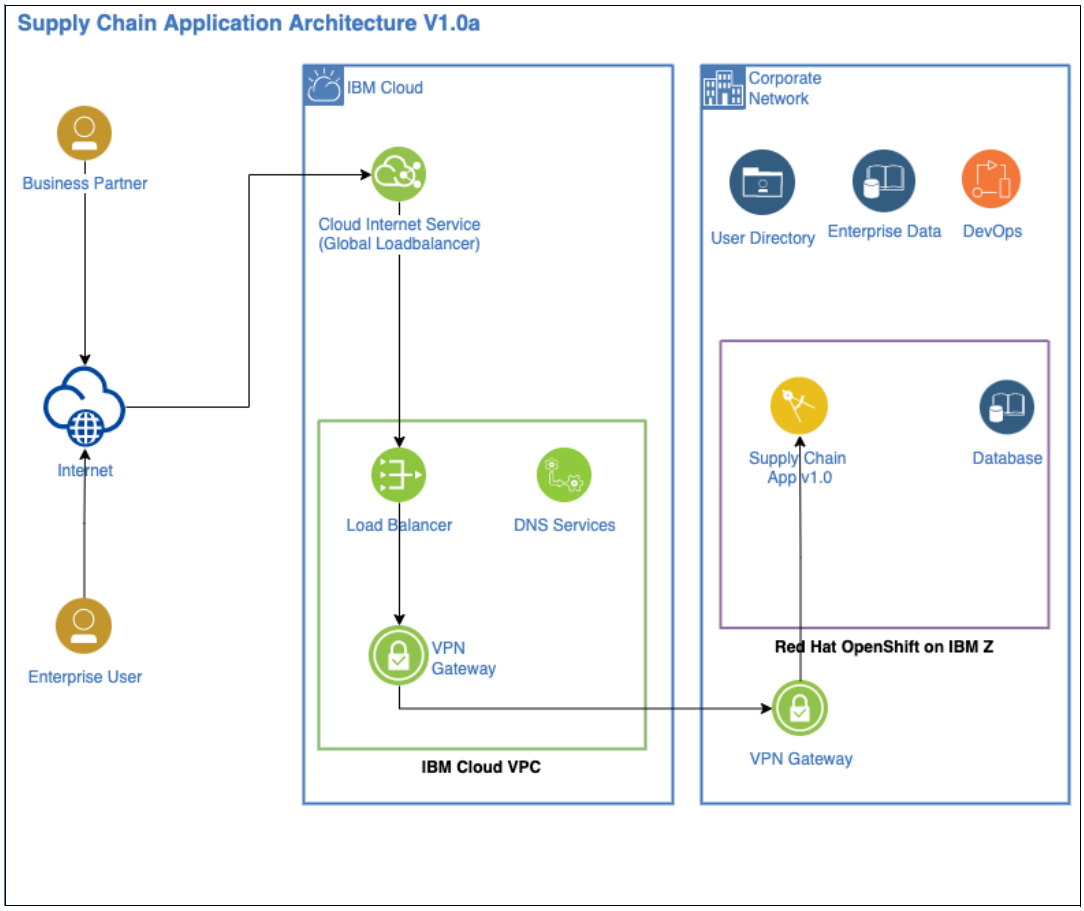


Figure 7-2 Target architecture

Benefits

Some benefits that are afforded from this target architecture include the following:

- Seamless application deployment. End user has no clue where infrastructure is located.
- Easy to implement multiple points of failures
- Leverage multiple networking and cloud security tools

Considerations

Latency requirements should be factored in when determining the location of the IBM Cloud VPC components. There are existing solutions that provide high bandwidth low latency links between an on-premises datacenter and various IBM Cloud locations.

7.2.4 Current architecture implementation

We will be deploying a simple application, written in Golang, to Red Hat OpenShift on IBM Z. The application listens on port 8080 and provides a health endpoint.

Build and deploy the Supply Chain App 1.0 from source code

While the Red Hat OpenShift source to image process was introduced in previous chapters, here we will be manually building the application by using Podman and uploading the image to the Red Hat OpenShift Container registry. For more information on the registry, see the following website:

https://access.redhat.com/documentation/en-us/openshift_container_platform/4.11/html-single/registry/index

The following are the steps to build and deploy an application.

1. Download Source Code by using Git by using the command shown in Example 7-1.

Example 7-1 Command to download source code

```
git clone github.com/redbook
cd redbook/ch7/scapp-1.0
podman build --pull --rm -f "dockerfile" -t scapp:1.0
```

2. Build and push the application to the Red Hat OpenShift registry, as shown in Example 7-2.

Example 7-2 Use Podman to build and push the application

```
podman build --pull --rm -f "dockerfile" -t scapp:1.0
```

3. Deploy the application to Red Hat OpenShift by using the command shown in Example 7-3.

Example 7-3 Deploy the application

```
oc create-project scapp
oc deploy bla
```

4. Check application with the command shown in Example 7-4.

Example 7-4 Check the application

```
oc describe deployment
```

5. Test the application with the command shown in Example 7-5.

Example 7-5 Command to test the application

```
curl app url
```

This command will return a response with the application version.

We have built and deployed a simple application. This application is now currently deployed.

7.2.5 Target architecture implementation

This section involves the creation of the components representative of the architecture shown in Figure 7-1 on page 128. Most of the steps will be conducted on the IBM Cloud web graphical user interface. The following are the high level steps:

“1. Create and configure a load balancer in IBM Cloud” on page 131

“2. Create the IBM Cloud internet service to manage domain” on page 135

“3.Create IBM Secrets Manager to order Public TLS certificates” on page 136

6. Configure the IBM Cloud Load Balancer with the TLS certificate.
7. Create and configure a site-to-site VPN between the IBM Cloud and the corporate network.

8. Configure the Red Hat OpenShift application as backed to the IBM Cloud Load Balancer.

In our example we have two domains, one internal and one external.

The Red Hat OpenShift cluster is configured to use the internal domain rdbkvmocp.pbm.ihost.com and applications are reachable on-premises via https://<app>.app.rdbkvmocp.pbm.ihost.com

We named our external domain zhybridcloud.centers.ihost.com. The application we will be deploying will be reachable in our lab environment via https://scapp.zhybridcloud.centers.ihost.com on the internet.

1. Create and configure a load balancer in IBM Cloud

In this step, we will be creating an application load balancer to receive traffic from the internet.

a. Create a resource group.

   Creating a resource group helps with grouping all the resources deployed for this IBM Redbooks publication. This can be done via the IBM Cloud command line interface (CLI) or via a graphical user interface (GUI).

   We went to https://cloud.ibm.com/account/resource-groups to create a resource group named *redbook*.

b. Navigate to the VPC Infrastructure page on IBM Cloud.

   On the IBM Cloud dashboard, click the Navigation Menu on the top left corner, then click *VPC Infrastructure* (Figure 7-3) or navigate directly to the following website: https://cloud.ibm.com/vpc-ext/provision/vpc

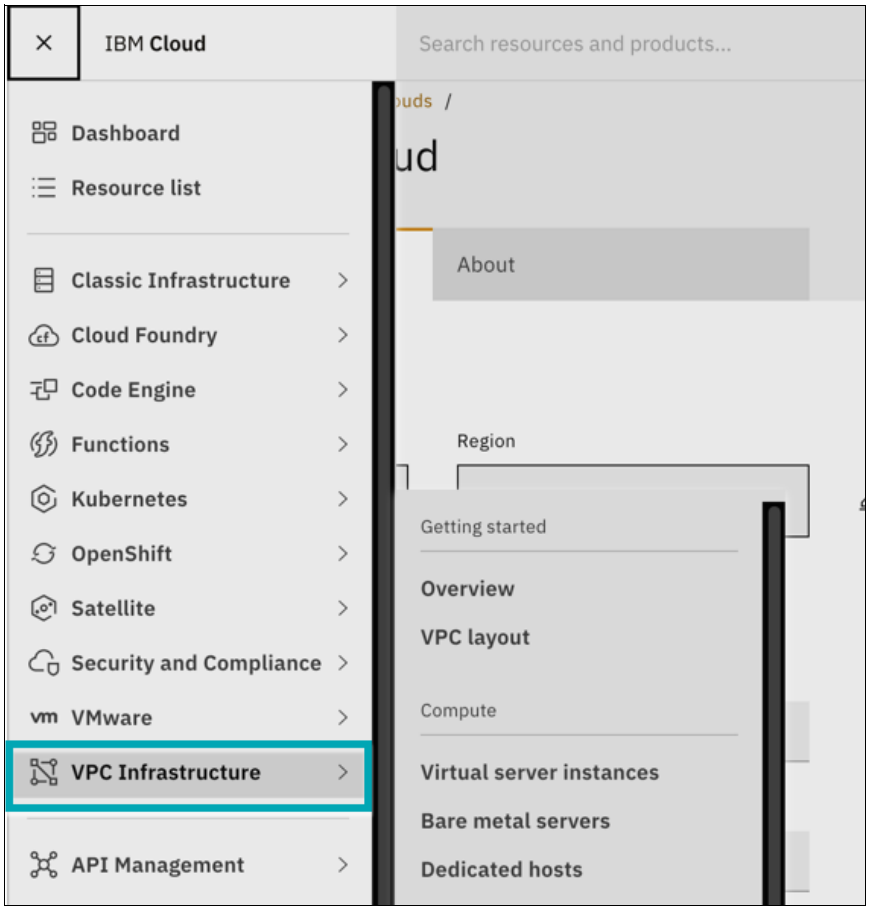


Figure 7-3 VPC Infrastructure navigation

c. Create a VPC.

In the VPC Infrastructure page, scroll the left pane to the Network section, Click *VPC* then click *Create*. A panel like that shown in Figure 7-4 will be displayed.

Alternatively you can click *Catalog* at the top right menu option and search for Virtual Private Cloud. We will be deploying in the North America Geography and Dallas region. We input the Name as *redbook* and select *redbook* as the resource group. No changes are required for the other options.

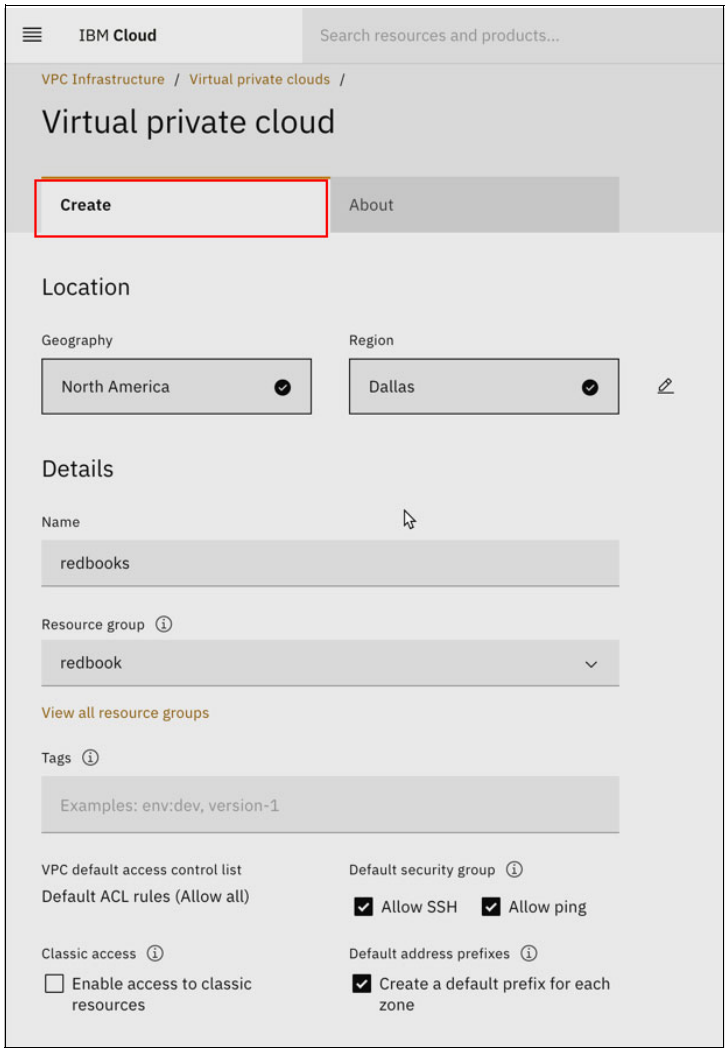


Figure 7-4 Create VPC

Now click *Create virtual private cloud*

d. Create the security group for the load balancer.

This step is used to configure the traffic flow roles for the Load balancer instance we will be creating. The planned traffic rules are to accept HTTPS traffic from any IP address and allow traffic of any type to any IP address. Add the rules as shown in Figure 7-5.

Figure 7-5 Security group rules for load balancer

We entered the name as *redbook-alb-sg*. Ensure the Location is the same as the VPC you created and the resource group is the same as the one you created. In our case, *redbook*.Tthe virtual private cloud we selected is also *redbook*. Now click “Create security group”

Note: In a real production environment, the outgoing traffic should be limited to only permitted protocols, ports and IP addresses.

e. Create the load balancer.

On the VPC Infrastructure page, scroll the left pane of to the network section, Click *Load balancers* then click *Create*.

Alternatively, navigate to https://cloud.ibm.com/vpc-ext/network/loadBalancers, Click *Create*.

Enter ‘zhybridcloud-alb‘ as the name of the load balancer. Ensure the Geography/Region is the same as in step c., “Create a VPC.” on page 132, the resource group is *redbook* and the *Application Load Balancer* is selected. Type should have “Public” selected, “Subnets” should have all the subnets selected and no backend pool or listeners need be configured at this time. For security groups, deselect the default highlighted as VPC default and select redbook-alb-sg that was created. See Figure 7-6.

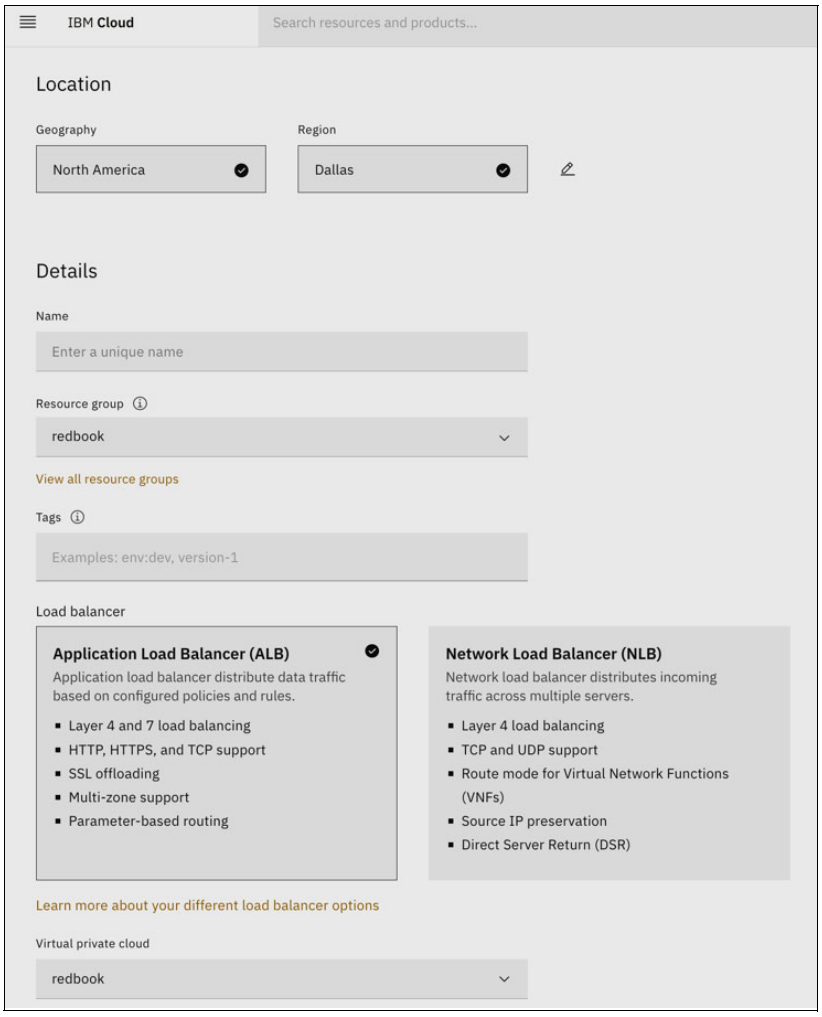


Figure 7-6 Create Load balancer

Now click the “Create load balance” button. Verify the load balancer was properly created. You should have two public IP addresses allocated to the load balancer. Take notes of the IP addresses as they will be referenced soon.

2. Create the IBM Cloud internet service to manage domain

In these next configuration tasks, we will be performing the following tasks:

- Use the IBM Cloud Internet Services to manage a delegated subdomain: zhybridcloud.centers.ihost.com
- Use the IBM Secrets manager to order TLS certificates for the subdomain.
- Configure HTTPS front end listener on load balancer with TLS certificate

1. Create IBM Cloud Internet Services Instance

   On the IBM Cloud dashboard, click “Catalog” at the top right menu option and search for “Internet Services”, Click on “Internet Services by IBM” tile. Select pricing plan, enter service name “cis-redbook” and select resource group “redbook”. Read the licence terms and click “Create” if you agree.

Note: As of this publication, a free 30 day trial plan exists which you can use for your test implementation.

2. Configure IBM Cloud Internet Service

   From the previous task, you should be redirected to the Internet service instance. If that does not occur, you can navigate to your resource list by clicking the top right IBM Cloud menu navigation and clicking “Resource List” and expanding the “Services and Software” section. Alternatively, you can navigate directly to https://cloud.ibm.com/resources.

   The next step here is to add a domain. Click the “Add a domain button”. Enter a domain name or subdomain that can be delegated, skip import of DNS records and move on to Configure Domain management.

   The Domain Management configuration requires delegating the domain or subdomain to an IBM cloud name server for DNS management.

   In our case, our network admin delegated the zhybridcloud.centers.ihost.com subdomain by configuring the name server (NS) records to the following:

   – ns001.name.cloud.ibm.com
   – ns096.name.cloud.ibm.com

   After the configuration has successfully completed, your domain should be active on the overview page.

   We will be coming back here to create a DNS A entry for a subdomain named scapp.zhybridcloud.centers.ihost.com that will resolve to our load balancer

3. Create a subdomain for the load balancer.

   On the Internet Service page of the cis-redbook, navigate to Reliability > DNS. Scroll down to the “DNS Records” section, and Click “Add”.

   Select A as type, TTL as Automatic and name as “scapp”, and enter one of the public IP addresses of the load balancer we created, then click “Add”

   To test this configuration is successful, run the command nslookup scapp.<domain>. It should resolve to one of the public addresses of the load balancer as shown in our Example 7-6.

Example 7-6 Results from NSLOOKUP

```
Non-authoritative answer:
Name:scapp.zhybridcloud.centers.ihost.com
Address: 52.116.129.47
```

3.Create IBM Secrets Manager to order Public TLS certificates

Perform the following steps to create your IBM Secrets Manager.

1. Create Secrets Manager instance

   Navigate to the catalog and search for secrets manager. Click on the “Secrets Manager by IBM” tile. Select location, this can be any location but we used “Dallas”.

   To configure the resources, enter “redbook-scm” as the service name, select “redbook” as the resource group and leave all the remaining defaults. The configuration should be similar to that shown in Figure 7-7.

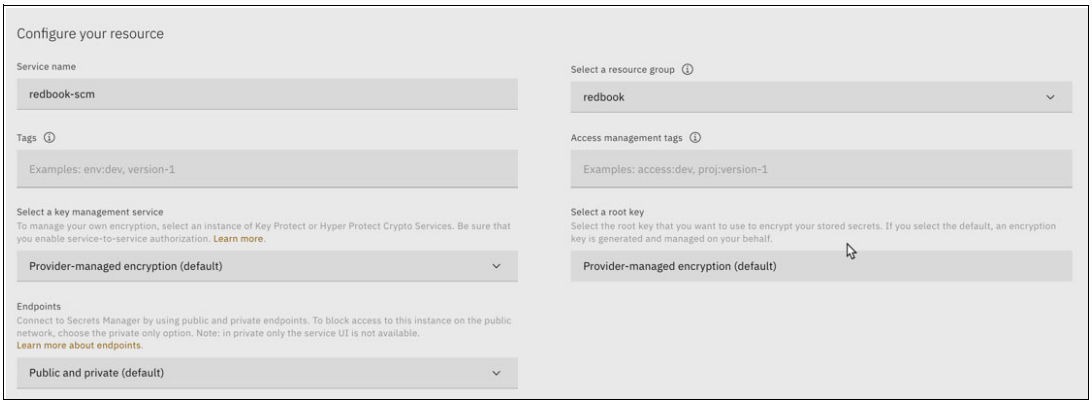


Figure 7-7 IBM Cloud Secrets Manager Config

Read the licence terms and if you agree, click the checkbox and click “Create”.

2. Create Identity Access & Management (IAM) access between Cloud Internet Services and Secrets Manager

   You can view instructions on the following website:

   https://cloud.ibm.com/docs/secrets-manager?topic=secrets-manager-prepare-order-certificates#authorize-cis

3. Configure the DNS provider in the Secrets Engine for Public Certificates:

   a. In the secrets manager UI, navigate to “Secrets engine” ==> “Public Certificates”.

   b. Click “Add” in the DNS provider section, enter “redbook-cis” as the Name and select “Cloud Internet Services” as the DNS Provider and click next

   If you have the IAM authorization properly configure, you should see the ‘redbook-cis‘ instance in the dropdown list in the “Authorization” tab. Select the ‘redbook-cis‘ and click add.

4. Create an ACME account for use with Let’s Encrypt Public certificate authority (CA).

   An account with the Automatic Certificate Management Environment (ACME) protocol will provide you with a means of installing a certificate management agent on your web server.

   You will create your account at the following website:
   https://github.com/ibm-cloud-security/acme-account-creation-tool

   You will need your account credentials in the following step.

5. Configure the Certificate Authority in the Secrets Engine for Public Certificates.

   To do this, perform the following steps:

   a. In the secrets manager UI, navigate to “Secrets engine” ==> “Public Certificates”:

   b. Click “Add” in the Certificate Authorities section, enter “redbook-letsencrypt” as Name and select “Let’s Encrypt” as Certificate Authority and click next.

   c. Click the “Enter value” tab, then copy and paste your private key from the ACME account you created in step 4.

6. Request TLS certificate for subdomain.

   In our case, we will be requesting a TLS certificate for scapp.zhybridcloud.centers.ihost.com

4. Configure IBM Cloud Load Balancer with TLS certificates

Create IAM access between Load balancer and Secrets Manager. In order to access the TLS certificates from the load balance, IAM access will need to be granted.

7.3 Extend on-prem App with application in Public Cloud

Lorem Ipsum

7.3.1 Architectural overview

Lorem Ipsum

7.3.2 As-Is Implementation

Lorem ipsum

7.3.3 To-be Implementation guide

Lorem Ipsum

7.4 Conclusions

A

Voting App changes needed to support IBM Db2 database

In section 2.3, “Sample application architecture” on page 18, we deployed an open-source, lightweight and microservices based application called the “Voting app”. You can find the public repository with the application source code at the following website:
https://github.com/OpenShift-Z/voting-app

In the original source code, the following programs are using a postgres database:

- voting-app/worker-python/app.py
- voting-app/result/server.js

In this appendix we provide the changes we made to the Voting app to support IBM Db2:

```
---text in red : Original source code
+++text in blue: Updated source code
```

- voting-app/worker-python/app.py as shown in Example A-1.

Example A-1 voting-app/worker-python/app.py

```
#!/usr/bin/env python3

from redis import Redis
import os
import time
---import psycopg2
+++import ibm_db
import json

def get_redis():
   redishost = os.environ.get('REDIS_HOST', 'new-redis')
   redispassword = os.environ.get('REDIS_PASSWORD', 'password')
   print ("Connecting to Redis using " + redishost)
   #redis_conn = Redis(host=redishost, db=0, socket_timeout=5)
   redis_conn = Redis(host=redishost, db=0, socket_timeout=5, password=redispassword)
   redis_conn.ping()
   print ("connected to redis!")
   return redis_conn

---def connect_postgres():
   ---host = os.getenv('POSTGRES_SERVICE_HOST', "new-postgresql")
+++def connect_db2():
   +++host = os.getenv('DB2_SERVICE_HOST', "new-db2ql")
   db_name = os.getenv('DB_NAME', "db")
```

```
   db_user = os.getenv('DB_USER', "admin")
   db_pass = os.getenv('DB_PASS', "admin")
   +++hostnm = os.getenv('HOST_NAME', "wtsc75.pbm.ihost.com")
   +++portno = os.getenv('PORT_NO', "38010")
   try:
      print ("connecting to the DB")
      ---conn = psycopg2.connect ("host={} dbname={} user={} password={}".format(host, db_name, db_user, db_pass))
      ---print ("Successfully connected to Postgres")

+++conn_str='database='+db_name+";hostname="+hostnm+";port="+portno+";protocol=tcpip;uid="+db_user+";pwd="+db_pass
      +++conn = ibm_db.connect(conn_str,'','')
      +++print ("Successfully connected to Db2")
      return conn

   except Exception as e:
      print ("error connecting to the DB")
      print (e)
---def create_postgres_table():
+++def create_db2_table():
    try:
       ---conn = connect_postgres()
       +++conn = connect_db2()
    except Exception as e:
       ---print ("error connecting to postgres")
       +++print ("error connecting to Db2")
       print (str(e))
    try:
       ---cursor = conn.cursor()
       ---sqlCreateTable = "CREATE TABLE IF NOT EXISTS public.votes (id VARCHAR(255) NOT NULL, vote VARCHAR(255) NOT
NULL);"
       ---cursor.execute(sqlCreateTable)
       +++sqlCreateTable = "CREATE TABLE votes (id VARCHAR(255) NOT NULL, vote VARCHAR(255) NOT NULL);"
       +++ibm_db.exec_immediate(conn, sqlCreateTable)
       print ("votes table created")
       ---conn.commit()
       ---cursor.close()

    except Exception as e:
       print ("error creating database table")
       print (e)

    try:
      ---conn.close()
      +++ibm_db.close(conn)

    except Exception as e:
       ---print ("error closing connection to postgres")
       +++print ("error closing connection to Db2")
       print (str(e))

---def insert_postgres(data):
+++def insert_db2(data):
    try:
       ---conn = connect_postgres()
       +++conn = connect_db2()

    except Exception as e:
       ---print ("error connecting to postgres")
       +++print ("error connecting to Db2")
       print (str(e))

    try:
---cur = conn.cursor()
       ---cur.execute("insert into votes values (%s, %s)",
       ---(
          ---data.get("voter_id"),
          ---data.get("vote")
       ---))
       ---conn.commit()
       +++insert = "insert into votes values(?,?)"
       +++stmt_insert = ibm_db.prepare(conn, insert)
       +++ibm_db.execute(stmt_insert,(data.get("voter_id"),data.get("vote")))
       print ("row inserted into DB")
       ---cur.close()

    except Exception as e:
       ---conn.rollback()
       ---cur.close()
       ---print ("error inserting into postgres")
       +++print ("error inserting into Db2")
       print (str(e))

    try:
```

```
      ---conn.close()
      +++ibm_db.close(conn)

    except Exception as e:
       ---print ("error closing connection to postgres")
       +++print ("error closing connection to Db2")
       print (str(e))


def process_votes():
    redis = get_redis()
    redis.ping()
    while True:
       try:
          msg = redis.rpop("votes")
          print(msg)
          if (msg != None):
             print ("reading message from redis")
             msg_dict = json.loads(msg)
             ---insert_postgres(msg_dict)
             +++insert_db2(msg_dict)
          # will look like this
          # {"vote": "a", "voter_id": "71f0caa7172a84eb"}
          time.sleep(3)

       except Exception as e:
          print(e)

if __name__ == '__main__':
    ---create_postgres_table()
    +++create_db2_table()
    process_votes()
```

- voting-app/result/server.js as shown in Example A-2

Example A-2 voting-app/result/server.js

```
var express = require('express'),
    async = require('async'),
    ---pg = require('pg'),
    ---{ Pool } = require('pg'),
    +++ibmdb = require('ibm_db');
    path = require('path'),
    cookieParser = require('cookie-parser'),
    bodyParser = require('body-parser'),
    methodOverride = require('method-override'),
    app = express(),
    server = require('http').Server(app),
    io = require('socket.io')(server);

io.set('transports', ['polling']);

var port = process.env.PORT || 8080;
---var pgconnectstr = process.env.POSTGRES_CONNECT_STRING;
+++var connStr = process.env.DB2_CONNECT_STRING;

io.sockets.on('connection', function (socket) {

  socket.emit('message', { text : 'Welcome!' });

  socket.on('subscribe', function (data) {
    socket.join(data.channel);
  });
});
---var pool = new pg.Pool({
---//  connectionString: 'postgres://postgres:'+passwd+'@db/postgres'
---//  connectionString: 'postgres://pfruth:pfruth@new-postgresql/postgres'
---//  connectionString: 'postgres://pfruth:pfruth@10.130.3.185:5432/postgres'
---  connectionString: pgconnectstr
---});

async.retry(
  {times: 1000, interval: 1000},
  function(callback) {
    ---pool.connect(function(err, client, done) {
    +++ibmdb.open(connStr, function (err,client) {
      if (err) {
        console.error("Waiting for db");
        ---console.log("pg error code:", err.code);
        +++console.log("db2 error code:", err.code);
```

```
      }
      callback(err, client);
    });
  },
  function(err, client) {
    if (err) {
      return console.error("Giving up");
    }
    console.log("Connected to db");
    getVotes(client);
  }
);

function getVotes(client) {
  client.query('SELECT vote, COUNT(id) AS count FROM votes GROUP BY vote order by vote', [], function(err, result) {
    if (err) {
      console.error("Error performing query: " + err);
    } else {
      var votes = collectVotesFromResult(result);
      io.sockets.emit("scores", JSON.stringify(votes));
    }

    setTimeout(function() {getVotes(client) }, 1000);
  });
}

function collectVotesFromResult(result) {
  var votes = {a: 0, b: 0};

  ---result.rows.forEach(function (row) {
  ---  votes[row.vote] = parseInt(row.count);
  ---});
  +++result.forEach(function (row) {
    +++votes[row.VOTE] = parseInt(row.COUNT);
  +++});

  return votes;
}

app.use(cookieParser());
app.use(bodyParser());
app.use(methodOverride('X-HTTP-Method-Override'));
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
  res.header("Access-Control-Allow-Methods", "PUT, GET, POST, DELETE, OPTIONS");
  next();
});

app.use(express.static(__dirname + '/views'));

app.get('/', function (req, res) {
  res.sendFile(path.resolve(__dirname + '/views/index.html'));
});

server.timeout = 0;
server.listen(port, function () {
  var port = server.address().port;
  console.log('App running on port ' + port);
  ---console.log('Postgres connect string ' + pgconnectstr);
  +++console.log('Db2 connect string ' + connStr);
});
```

B

Additional material

This book refers to additional material that can be downloaded from the Internet as described in the following sections.

Locating the web material

The web material associated with this book is available in softcopy on the Internet from the IBM Redbooks GitHub repository:

https://github.com/IBMRedbooks/SG248532-zos-hybrid-cloud-examples

Using the web material

The additional web material that accompanies this book includes the following files:

File name	*Description*
../tree/main/Chapter04	kafka-demo-consumer - a sample application that is a proof of concept for the proposed architecture in “Proposed to-be architecture: Phase 2” on page 48. kafka-demo-producer - a sample application that is a proof of concept for the proposed architecture in “Proposed to-be architecture: Phase 2” on page 48.

Additional requirements

The web material requires the following system requirements in order to build and deploy this code on any machine:

- Java - an object-oriented programming language
- Maven - a build automation tool used primarily for Java projects.

Related publications

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

IBM Redbooks

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

- *Why IBM Hybrid Cloud for Your Journey to the Cloud?*, REDP-5653
- *IBM Data Virtualization Manager for z/OS*, SG24-8514
- *Getting started with z/OS Container Extensions and Docker*, SG24-8457
- *Red Hat OpenShift on IBM Z Installation Guide*, REDP-5605

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

Online resources

These websites are also relevant as further information sources:

- DevOps from APIs to z Systems For Dummies

  http://www.recarta.co.uk/wp-content/uploads/2017/05/DevOpsforDummies-ilovepdf-compressed.pdf

- IBM Wazi as a Service, Bringing your own image with Wazi Image Builder

  https://www.ibm.com/docs/en/wazi-aas/1.0.0?topic=bringing-your-own-image-wazi-image-builder

- The Cloud Adoption Playbook

  https://www.ibm.com/cloud/architecture/adoption/the-cloud-adoption-playbook/

- Git repository for kafka-demo-producer

  https://github.com/IBMRedbooks/SG248532-zos-hybrid-cloud-examples/tree/main/Chapter04/kafka-demo-producer

- Git repository for kafka-demo-consume

  https://github.com/IBMRedbooks/SG248532-zos-hybrid-cloud-examples/tree/main/Chapter04/kafka-demo-consumer

Help from IBM

IBM Support and downloads

ibm.com/support

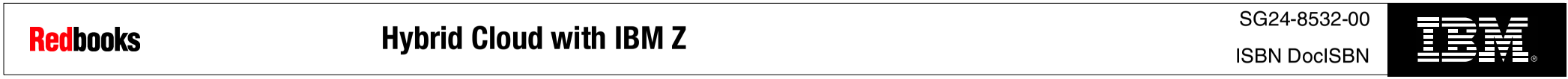
IBM Global Services

ibm.com/services

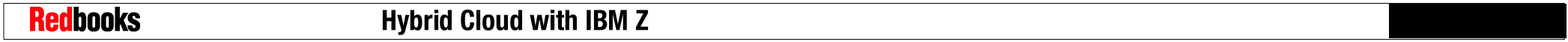
To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5” spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(-->Hide:)>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine.fm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.

(1.5” spine)
1.5”<-> 1.998”
789 <->1051 pages

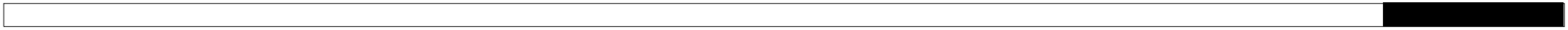
(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages



(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

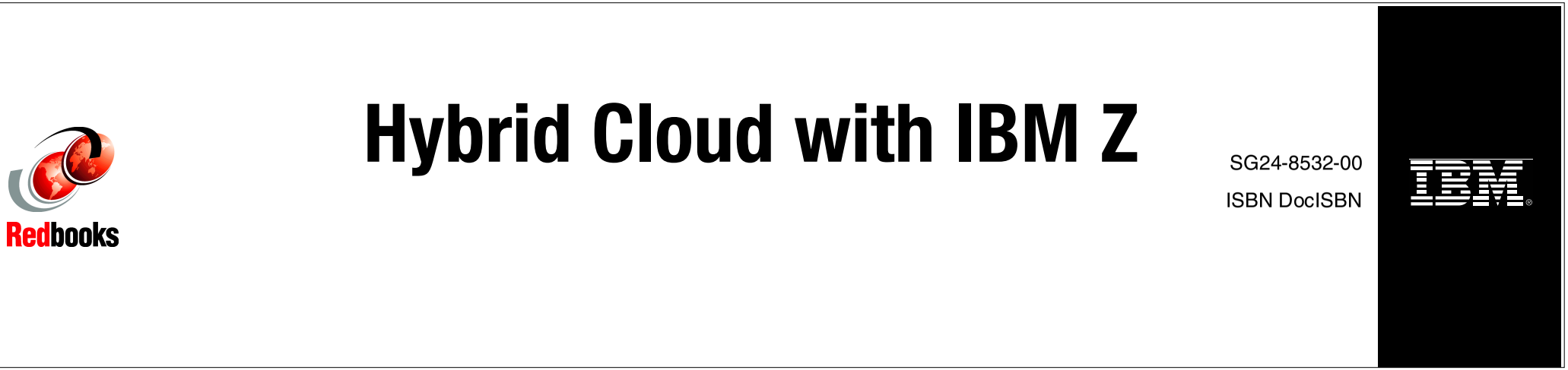


(0.2”spine)
0.17”<->0.473”
90<->249 pages



(0.1”spine)
0.1”<->0.169”
53<->89 pages

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5” spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(-->Hide:)>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine.fm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.



(2.5” spine)
2.5”<->nnn.n”
1315<-> nnnn pages

(2.0” spine)
2.0” <-> 2.498”
1052 <-> 1314 pages

SG24-8532-00

ISBN DocISBN

Printed in U.S.A.