Redbooks
ibm.com/redbooks

# Mainframe Application Modernization Patterns for Hybrid Cloud

Lydia Parziale

Yinka Adesanya

Elton de Souza

Peter Haumer

Sandor Irmes

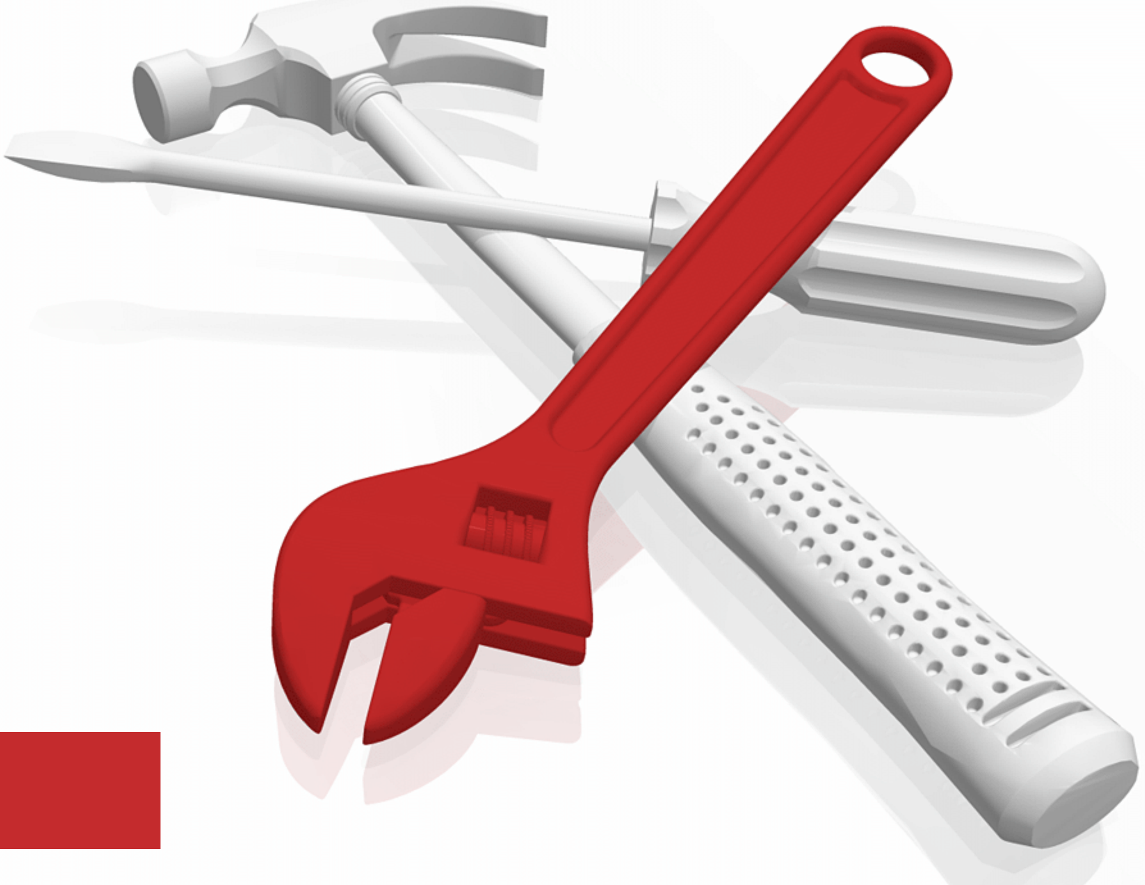
Amey Patil

Lauren K Li

Liyong Li

Filipe Miranda

Sidney Varoni



Cloud

IBM Z

IBM

Redbooks

IBM Redbooks

# Mainframe Application Modernization Patterns for Hybrid Cloud

June 2023

SG24-8532-00

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

**First Edition (June 2023)**

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

# Contents

# Notices

# Trademarks

# Preface

As businesses digitally transform, they can impose significant demands on existing mainframe applications and data requiring the need to modernize to achieve business outcomes.

To meet this need for modernization which includes greater agility, efficiency, and innovation, they are choosing a hybrid cloud strategy that brings together the best of IBM Z® and Cloud.

IBM Z integrated into a hybrid cloud based on Red Hat OpenShift provides resiliency and secure architecture that allows for application workload placement on the "best fit" infrastructure to maximize scale, performance, and efficiency. You can accelerate application modernization today by using architectural patterns that are building blocks and best practices to learn how to implement and deploy application modernization in an IBM Z environment integrated with public cloud.

In the IBM Redpaper, *Accelerate Mainframe Application Modernization with Hybrid Cloud,* REDP-5705, we discussed strategies and architectural solutions that can accelerate your mainframe application modernization by leveraging hybrid cloud environments.

This IBM Redpaper publication takes that IBM Redpaper on a deeper dive as it discusses and demonstrates implementation approaches to modernization when adopting a hybrid cloud with IBM Z. 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 applications.

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. She 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, US. 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 and public clouds, application and infrastructure performance, and delivering solutions at scale. He also has keen interests in security, networking, and blockchain.

**Elton de Souza** is a Senior Technical Staff Member for Hybrid Cloud Adoption Acceleration based at IBM Canada. He has worked on the IBM Z platform his entire 12-year career at IBM and leads hybrid cloud adoption as part of the IBM Hyper Protect Services organization. The first half of his career was spent on the internals of Java, where he worked on leveraging 200+ hardware instructions on IBM Z for mission-critical mobile and cloud workloads. He was one of the first technical experts for Docker and Kubernetes on IBM Z in early 2015, and since then has worked with IBM clients on successful adoption of cloud-native technology like Kubernetes, IBM Cloud® Private, and most recently Red Hat OpenShift and IBM Cloud Paks and IBM Hyper Protect Services. He has written over 50 publications, including patents, books, analyst reports, and peer-reviewed reference architectures. He has won several awards, including external industry awards, and he contributes to several open-source projects.

**Peter Haumer** is a development lead and senior technical staff member for the IBM Wazi product platform, which is composed of tools and solution packages for IBM z/OS® Cloud-native DevOps. He is at the IBM Silicon Valley Labs in California, US. He has 40 years of software development experience. For IBM, he has worked as an agile software development team lead, full-stack software developer, and senior researcher. He has created and released 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, IBM Jazz® Global Configuration Management, IBM 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 and IBM LinuxONE consulting services at EMEA IBM 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 networks 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 the IBM Z DevOps Acceleration Team, which helps clients begin their mainframe modernization journey by integrating IBM technologies with Git-based continuous integration (CI) and continuous deployment (CD) (CI/CD) pipeline solutions. In her 4 years with IBM, she has contributed code and documentation to the open-source Zowe Explorer Visual Studio Code extension and 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, where he helps customers adopt 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, IBM z/VM®, and Linux.

**Filipe Miranda** is a Principal Technical Specialist for Hybrid Cloud on IBM Z and LinuxONE. He is a member of the zAcceleration Team (ZAT). Filipe has 15+ years of experience working with open-source technologies. He has 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 (Kernel-based Virtual Machine (KVM) and z/VM, VMware, Xen, and other hypervisors), containers (Docker, Podman, and Cri-O), Kubernetes, Red Hat OpenShift (on x86, IBM Power, and IBM Z/LinuxONE), and other products from IBM and Red Hat.

**Amey Patil** is an Application Architect for the IBM CIO team in Raleigh, North Carolina, US. He has 15+ years of experience in IT consulting and software development. He holds a master's degree in Software Engineering from th Birla Institute of Technology & Science, Pilani, India. His areas of expertise include cloud computing, blockchain, business process management (BPM), service-oriented architecture (SOA), APIs and microservices, enterprise application integration, and middleware.

**Sidney Varoni** is an IBM Z Technical Sales Specialist at IBM Australia. He has 16 years of experience in the IT Infrastructure field, including IBM Z and storage solutions. He has worked at IBM Global Technology Services, IBM Storage, and currently at the IBM Z business unit. He holds a bachelor's 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, and 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
**IBM Redbooks, Poughkeepsie Center**

Patrik Hysky
**IBM Redbooks, Austin Center**

Tom Ambrosio and Bill Lamastro,
**IBM CPO**

Dennis Behm, Nicolas Dangeville, Nasser Ebrahim, Mike Fulton, Mythili Venkatakrishnan, Suman Gopinath, Peter McCaffrey
**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 by 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 describes reasons for adopting a hybrid cloud approach to a mainframe application modernization that leverages the IBM Z platform and cloud. It also introduces patterns that are covered more in-depth throughout this book by using examples that use hybrid cloud with IBM Z to resolve real business challenges. Also, this publication deals with patterns for the following entry points:

- Enhancing and modernizing applications: Drive enhancements to business functions by using modern languages on z/OS and extending them as cloud-native applications that can be deployed in the best-fit platform based on business needs.
- Integrating across hybrid cloud.
- Simplifying information sharing and data access.
- Getting more agile with enterprise DevOps.

This chapter covers the following topics:

- What does hybrid cloud mean in 2023
- The value of the hybrid cloud approach
- Application modernization
- What are the application modernization patterns
- How to apply application modernization patterns on IBM Z
- For more information, see Enterprise DevOps pattern.
- Security

## 1.1 What does hybrid cloud mean in 2023

The *cloud* was originally an umbrella term for resource capacity (primarily compute) either on-premises behind a corporate firewall, or on an infrastructure that is managed by a third party and beyond the company premises (public cloud). This form of compute delivery is known as infrastructure-as-a-service (IaaS). Since then, cloud has evolved to higher layers of the architecture stack, that is, platform, service, functions, and others, which all are offered as-a-service model. This approach offers higher layers of abstraction that enables rapid innovation and accelerated go-to-market for products and services. It also shifts responsibility for uptime service-level agreements (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, and others focus on specific workloads, for example, analytics, highly regulated industries, confidential computing, and hardware-based workload accelerators. IBM focuses on highly regulated industries (IBM Cloud for Financial Services®) because of our industry leading confidential computing capabilities and evolved operations around regulatory controls.

*Cloud-native* is another term that is used (wrongly) 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 environments that might not be traditionally considered to be in the purview of cloud. Although there is no industry standard definition of cloud-native, some of the characteristics are as follows:

- Elasticity (both vertical and horizontal)
- Source control-managed infrastructure as a code and platform configuration
- DevOps (the tools and culture of DevOps)
- Software that is packaged in a way that enables reproducibility (for example, containers)
- A microservice architecture (only where it makes architectural sense)
- Bring your own language (BYOL), which is also known as polyglot development)
- No hard dependencies on the base or host operating system (outside of container base images)
- End to end automation (where feasible)
- Standardized access points by using industry standard protocols (for example, HTTPS, JSON, protobuf, gRPC, and Avro)
- Separation of state (typically implemented by using stateful containers that are consumed by stateless containers)

Figure 1-1 on page 3 shows a sample hybrid cloud topology that is 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

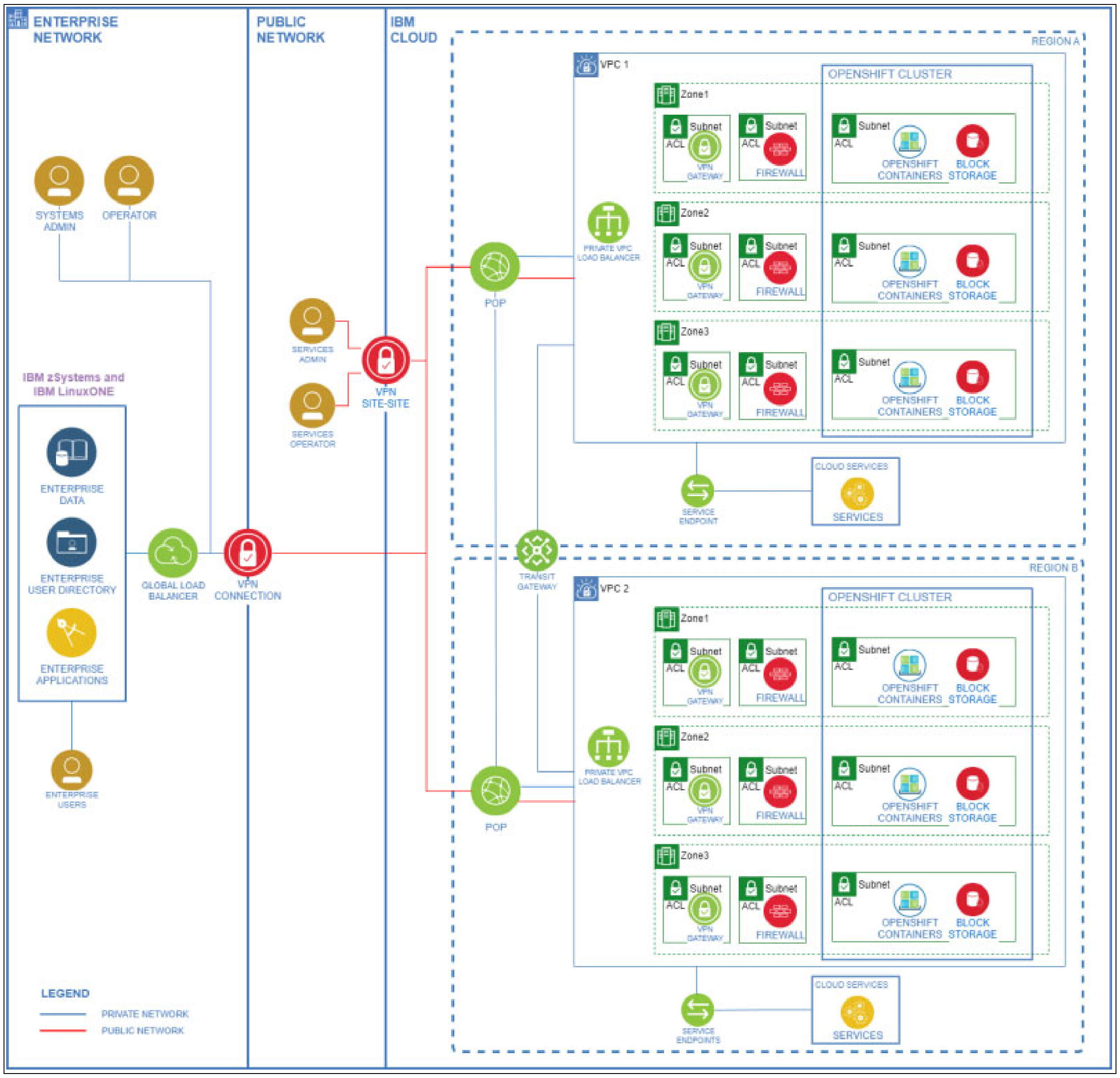


*Figure 1-1 Sample topology overview*

An application that is deployed on the public cloud might not be cloud-native, and conversely an application that is deployed on-premises in a heritage environment (such as VMWare or z/OS) might be cloud-native. Cloud-native implies the set of characteristics versus specifying an underlying platform or infrastructure as a prerequisite, although picking the right underlying platform might reduce friction in running and operations.

*Multi-cloud* is the term that is 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, which includes software-as-a-service (SaaS), function-as-a-service (FaaS), platform-as-a-service (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; and other items.

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

In cases where latency impacts user experience (UX), it makes sense to move to a cloud-native in-place versus a “lift and shift” incrementally to a new location. This approach offers the best balance of agility, time to market, and SLAs while minimizing operational or compliance risk. In cases where latency is not as critical, the strangler pattern might be a good fit.

This publication provides a use-case-driven approach to designing various hybrid cloud solutions to optimize for enterprise KPIs while maintaining risk. Most books assume the same underlying architecture (typically amd64), but modern cloud designs include alternative architectures like arm64, s390x, and ppc64le because each Instruction Set Architecture (ISA) or hardware architecture offers unique benefits.

Modernization on IBM Z typically benefits from leveraging decades of investment by organizations into intellectual property that is critical to their operations on this platform. Also, there is programming language modernization that can 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, or Go. In both scenarios, the consumption of a developer friendly integrated development environment (IDE) that supports the source and target language is beneficial.

Tools that have the capabilities 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 application programming interface (API) extension of applications, data virtualization, caching, and event-based architecture that also are used for modernization applications. To accelerate modernization, IBM created an all-inclusive stack that is named the IBM Z and Cloud Modernization Stack that simplified procurement of the tools that are necessary for modernization. The following chapters describe how to implement a subset of those patterns.

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

## 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 for companies of all sizes in every industry. However, there are many ways of adopting cloud computing, and many things to consider when deciding on the best approach:

- Should I use a 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?

There is no single cloud solution for all business problems, but a cloud solution often is defined at the application level, depending on its unique requirements.

A hybrid cloud approach excels at a flexibility. 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 described in *Why IBM Hybrid Cloud for Your Journey to the Cloud?*, REDP-5653, *The Cloud Adoption Playbook*, and *Accelerate Mainframe Application Modernization with Hybrid Cloud*, REDP-5705.

## 1.3 Application modernization

Mainframe applications that are part of hybrid cloud enterprises can leverage a set of capabilities that are becoming an industry standard:

- Agility through automated DevOps practices.
- Integration through OpenAPIs that actively participate in the event-processing ecosystem.
- Ability to shift workloads around different enterprise platforms, including an on-premises infrastructure and various cloud providers.
- Provide intelligent data analytics as part of the enterprise data fabric.

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

- For new or recently deployed workloads, these capabilities are adopted as part of the development cycle by using modern tools and practices.
- For legacy workloads, typically written many years ago, applications might need to be modernized by using certain patterns.

## 1.4 What are the application modernization patterns

Application modernization involves cost, risk, change impact, and multiple variables that must be considered when deciding on the best approach. *Accelerate Mainframe Application Modernization with Hybrid Cloud*, REDP-5705 details a set of entry points that can be used as various starting points of mainframe application modernization to address specific business challenges. Here are the entry points:

- Optimizing the cost and performance of existing mainframe applications: Optimize and improve the efficiency of the existing IBM Z applications by adopting the right set of hardware and software features and pricing models.
- Enhancing and modernizing applications: Drive enhancements to business functions by using modern languages on z/OS and extending them as cloud-native applications that can be deployed in the best-fit platform based on business needs.
- Integrating across hybrid cloud: Making mainframe applications part of the OpenAPI ecosystem and event-driven architectures, and using the data and business processing that is embedded in these applications.
- Simplifying information sharing and data access: Gain data-driven business value from mainframe applications share system of record (SOR) data either through direct access, replication, caching, or data virtualization concepts that combine data assets across the enterprise.
- Getting more agile with enterprise DevOps: Derive deeper insights about the operations and enable agile solution development through techniques and tools that support DevOps practices and software pipelines.
- Making AI-driven decisions at scale: Infuse AI within z/OS applications and transactions to builder faster resilient and intelligent systems.
- Automating and standardizing IT: Standardize IT automation across all platforms, including IBM Z.

A set of patterns and accelerators is published by IBM at Application modernization for iBM Z architecture.

This set of technical building blocks is designed and published to provide guidance and best practices to address these entry points. Modernization is a continuous journey with incremental progress. Technology is not static and keeps evolving, so these patterns must be constantly adjusted and updated to remain current.

### 1.4.1 Accelerating modernization with application discovery

Discover and understand your applications before they are modernized or a modernization pattern is applied to them. Application discovery is an accelerator that kickstarts this modernization journey. You can extract consumable information about your software assets and build an inventory of applications and their resource usage and dependencies, and visualize an information flow across application components, perform impact analysis, and generate reports to act on your modernization strategy and plan increments with confidence.

In the next section, these patterns are applied in an IBM Z context.

## 1.5 How to apply application modernization patterns on IBM Z

There is no single solution for all applications or business problems. Therefore, the best modernization pattern depends on the wanted outcome for each application. Understanding application and data interdependencies, affinities, performance, security, and availability are a prerequisite to identifying the best application modernization strategy. Multiple patterns might apply to a single application, depending on its requirements.

The modernization patterns for IBM Z are described next.

### 1.5.1 Patterns for enhancing and modernizing applications

Enhance application functions by applying one or more of the following patterns:

- Extending existing code applications on z/OS with cloud-native applications
- Collocating applications with existing IBM Z applications and data
- Enhancing selected functions by incrementally rewriting as cloud-native
- Refactoring elements of an existing IBM Z application into discrete services

#### Extending 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.

For more information, see Extend with a new function pattern.

#### Collocating applications with existing IBM Z applications and data

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

For more information, see colocate applications pattern.

#### Enhancing selected functions by incrementally rewriting as cloud-native

Incrementally rewrite a part of a mainframe application, which is 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.

For more information, see Enhance an existing function pattern.

#### Refactoring elements of an existing IBM Z application into discrete services

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

For more information, see Refactor into discrete services pattern.

### 1.5.2 Hybrid cloud network architecture

Although a complete description of the network architecture is outside of the scope of this publication, in our environment, we worked with the following three main topologies:

- An application running in a non z/OS environment that accesses database or files on z/OS.
- An application running on z/OS, typically leveraging IBM CICS®, IBM IMS, IBM MQ, IBM WebSphere® Application Server, or equivalent middleware subsystems that access or store data in an external data repository, such as MongoDB, Postgres, or equivalent databases.
- Both the application and data repository running on Red Hat OpenShift, either on IBM zCX or Linux on IBM Z.

### 1.5.3 Patterns to simplify information sharing and data access

These patterns focus on eliminating the existence of data silos within organizations. Data that is on IBM Z should be accessible by applications regardless of their platforms, but you still respect data security practices.

To simplify information sharing and data access, use one or more of the following patterns:

- Enabling modern access to existing IBM Z data
- Virtualizing IBM Z data to provide access across data sources without replication
- Caching SOR data on IBM Z to create data
- Transforming SOR data on IBM Z to create data
- Replicating SOR data on IBM Z by using change data capture

#### Enabling modern access to existing IBM Z data

Provide modern support to access IBM Z data through SQL-based queries 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.

For more information, see Enable modern access to IBM Z data pattern.

#### Virtualizing IBM Z data to provide access across data sources without replication

Access data across IBM Z and other data sources, including joining data, without needing to copy and replicate that data. Deliver more current and accurate data with virtualized data access to consuming applications, including analytics.

For more information, see Virtualize IBM Z data pattern.

#### Caching SOR data on IBM Z to create 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.

For more information, see Cache IBM Z data pattern.

### Transforming SOR data on IBM Z to create data

Incrementally build new, modernized SOR data stores by tapping into IBM Z data traffic through a data adapter to transform to the modernized data format.

For more information, see Transform IBM Z data.

### Replicating SOR data on IBM Z by 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.

For more information, see Replicate IBM Z data.

## 1.5.4 Patterns to integrate across a hybrid cloud

IBM Z applications can interact in near real time with other applications regardless of their platform with various integration patterns.

Here are the integration patterns that can be applied:

- Exposing applications through APIs
- Responding in near real time to events occurring within IBM Z applications
- Responding to external events in near real time by leveraging IBM Z applications
- Optimizing Command Query Response Separation: delivering core systems integration

### Exposing applications through APIs

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

For more information, see Expose through APIs pattern.

### Responding 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.

For more information, see Respond to IBM Z application events pattern.

### Responding to external events in near real time by leveraging IBM Z applications

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.

For more information, see Respond to external events pattern.

#### Optimizing Command Query Response Separation: delivering core systems integration

Deliver an efficient Command Query Response Separation (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 (zDIH) to deliver a non-disruptive, low-latency, high-throughput, and cost-attractive solution.

For more information, see Optimize CQRS pattern.

### 1.5.5 Implementing enterprise DevOps and observability

DevOps focuses on enabling IBM Z applications to leverage common agile practices and tools that are used by modern development frameworks. It involves the following functions:

- Provides a cloud-native developer experience for IBM Z by fully integrating IBM Z development into enterprise continuous integration (CI) and continuous deployment (CD) (CI/CD) pipelines and embracing consistent open-source tools that are familiar to all developers.

  For more information, see Enterprise DevOps pattern.
- Improves visibility across z/OS systems and empowers operations teams with AI insights.

## 1.6 Security

Although security is an important topic, it is not covered in this book. 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 application changes.
- Quantum-safe protection: Protect your data, applications, and infrastructure from possible 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 while 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 IBM DS8900F, or between IBM Z platforms.

For more information, see the IBM Z Enterprise Security portal at IBM Z Mainframe Enterprise Security.

2

# Modernized application architectures

IT environments are now fundamentally hybrid 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 applications to leverage cloud-native technologies, which enable consistent and reliable development, deployment, management, and performance across cloud environments and across cloud vendors.

This chapter describes how to accelerate application modernization by using the following application-centric[1] architectural patterns that are 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 information, see Application modernization for IBM Z architecture.

- Expose through application programming interfaces (APIs)

  Expose applications and data through APIs. Access mainframe applications and data by using standards-based REST APIs with IBM z/OS Connect. 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. 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.

[1] The application-centric patterns are part of IBM Z application modernization patterns, which are described at `https://www.ibm.com/cloud/architecture/architectures/application-modernization-mainframe/patterns`.

- 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 service-level agreement (SLA) objectives.

- Enhance as cloud-native

  Enhance selected functions by incrementally rewriting as cloud-native. Incrementally rewrite a part of a mainframe application, which is 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.

This chapter covers the following topics:

- Expose through APIs pattern
- Extend with cloud-native pattern
- 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 must 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 that is 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 must 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

Figure 2-1 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. When the API gateway receives a request, it 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- or JSON-based API request into a payload according to the specified copybook format. The server 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.

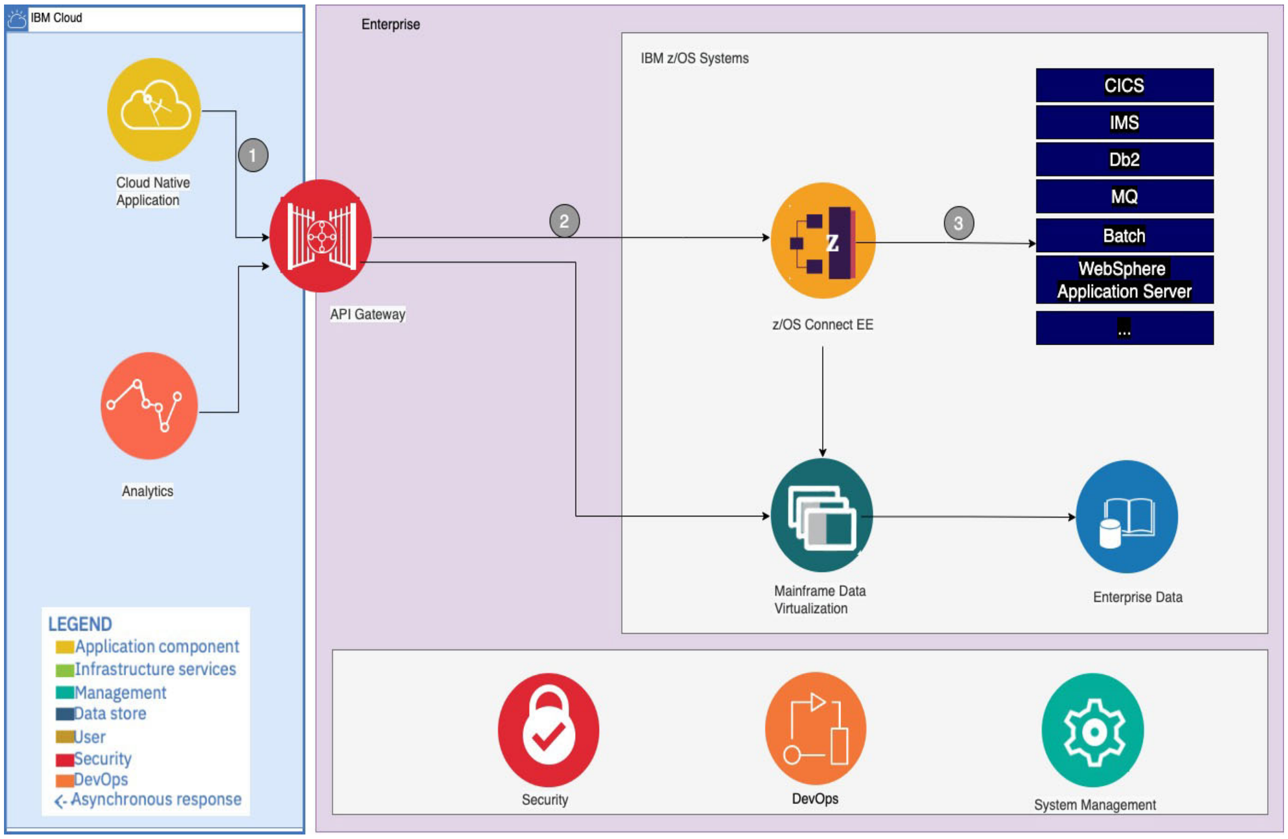


*Figure 2-1 Exposing through APIs on IBM Z*

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 the IBM z/OS Connect API toolkit.

  The ability to use an API from a z/OS application starts with using an API description format to document and understand the interface for 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 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:** For more information about IBM z/OS Connect, see Overview of IBM z/OS Connect (OpenAPI 3).

- Use IBM Application Discovery and Delivery Intelligence.

  To discover and understand the code that is 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, see Application discovery for business alignment pattern.

- Enable monitoring and management.

  Consider integrating the monitoring of APIs with the monitoring of the end-to-end solution, which span the cloud-based application that invokes APIs and the API enabling run times and core applications that run on IBM Z. For more information, see Chapter 6, “Managing your applications” on page 109.

- 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. For more information, see Chapter 7, “Deploying production applications” on page 125.

## 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 cannot 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 because 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 that is shown in Figure 2-2 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 occurs by using well-defined APIs.

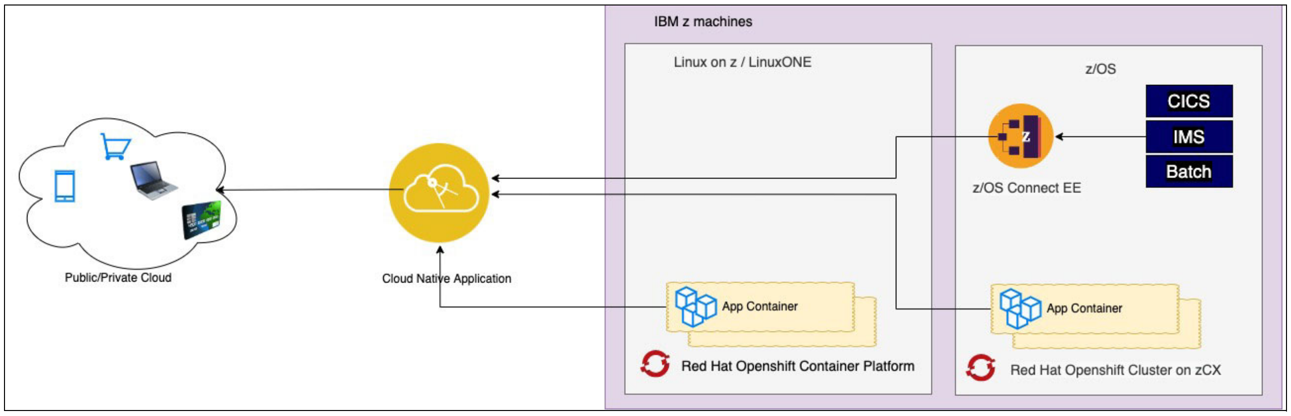


*Figure 2-2 Extending core applications on IBM Z with cloud-native*

A key business benefit of using this pattern is that enterprises do not need to abandon their investment in core applications on IBM Z. Instead, they can extend the capabilities of their applications with cloud-native applications. The usage 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 the speed to develop and deploy new capabilities and respond to market demands by using modern development tools and processes.
- Addresses skills shortages of older technology by using open standards-based languages.
- Leverages security frameworks that are available with modern applications to meet compliance standards.
- Avoids the high risks of rebuilding business-critical applications.
- Provides a way to deploy updates without redeploying the entire application.

## Considerations

- The usage 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 by using the IBM z/OS Container Extensions (zCX) 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 a public cloud.
– Deploy containers on any Kubernetes platform.

- The key decision criteria include reducing latency with the collocation of cloud-native applications closer to z/OS based applications and data and meeting stringent SLAs on security, availability, and scalability. For more information, see Collocate applications pattern.

## 2.3 Sample application architecture

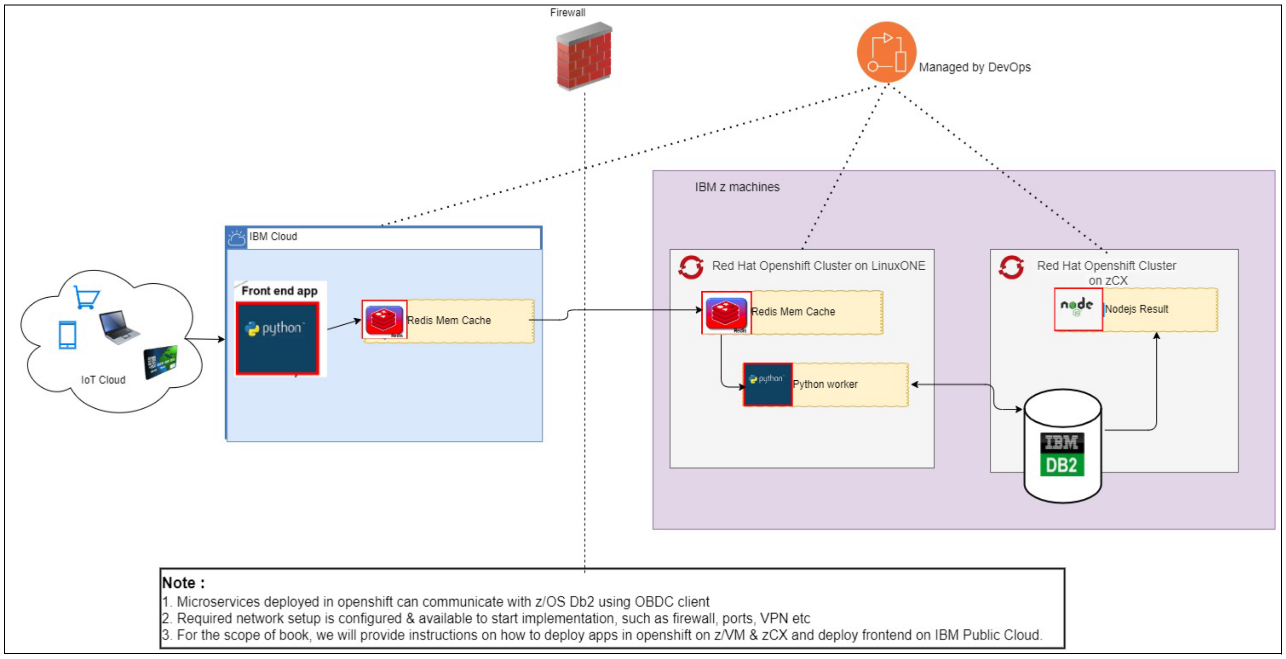
In our sample application architecture, we deploy an open-source, lightweight, and microservices-based application that is called the “Voting app”. It is cross-platform and can be deployed on any architecture. We use this application as an example to demonstrate how to leverage the application modernization 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 GitHub.

In the original source code, some programs are using a Postgres database, and we changed these programs to support IBM Db2, as described in Appendix A, “Voting app changes to support an IBM Db2 database” on page 139.

### 2.3.1 Application architecture

Our application architecture is shown in Figure 2-3.



**Note :**
1. Microservices deployed in openshift can communicate with z/OS Db2 using OBDC client
2. Required network setup is configured & available to start implementation, such as firewall, ports, VPN etc
3. For the scope of book, we will provide instructions on how to deploy apps in openshift on z/VM & zCX and deploy frontend on IBM Public Cloud.

*Figure 2-3 Sample Voting app architecture*

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

- On IBM public cloud

  The web front end is composed of a Python microservice that displays the options that users choose. 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 that are received by the Python web microservice.

  Another microservice, also written in Python, is running in the background, and it takes the votes from Redis and stores them in an IBM z/OS Db2 database.

- On IBM z/OS

  A running Db2 database stores the votes.

- On Red Hat OpenShift Cluster on zCX

  A Node.js microservice 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

The architecture is deployed on the following environment:

- Red Hat OpenShift cluster on IBM public cloud
- Red Hat OpenShift cluster on IBM LinuxONE
- IBM zCX Foundation for Red Hat OpenShift (zCX for Red Hat OpenShift)
- Db2 on z/OS

To deploy the application, you must get the GitHub personal access token that will be used for generating a secret. For more information, see Creating a personal access token.

To start the deployment, complete the following steps:

1. Deploy the z/OS Db2 database with the following settings:

   ```
   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
   ```

2. On all Red Hat OpenShift clusters, complete the following steps:

   a. Set the following environmental variables: **`$PROJECT`**, **`$GIT_REPO`**, and **`$GIT_TOKEN`**.

   b. Create a project by using the following command:

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

c. Create the secret by using the following command:

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

d. Import images by using the following commands:

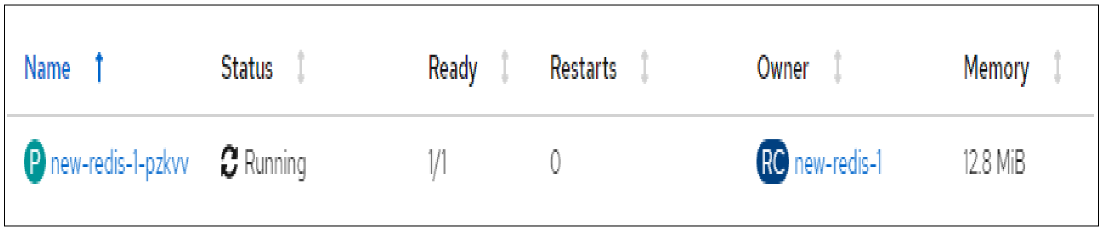
```
oc import-image rhel8/nodejs-12 --from=registry.redhat.io/rhel8/nodejs-12
--confirm
oc import-image ubi8/python-38 --from=registry.redhat.io/ubi8/python-38
--confirm
```

3. On Red Hat OpenShift cluster on LinuxONE, complete the following steps:

   a. Deploy the Redis service by using 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
```

Figure 2-4 shows a sample output for the Redis pod.



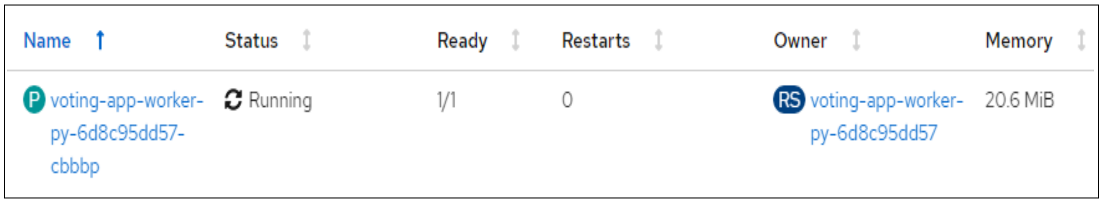
| Name | Status | Ready | Restarts | Owner | Memory |
|---|---|---|---|---|---|
| P new-redis-1-pzkvv | Running | 1/1 | 0 | RC new-redis-1 | 12.8 MiB |

*Figure 2-4 Sample output for Redis pod*

   b. 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"
```

Figure 2-5 shows a sample output for the Python worker 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 |

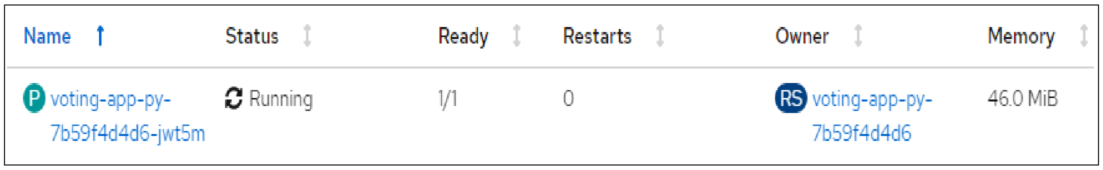
*Figure 2-5 Sample output for Python worker pod*

4. On Red Hat OpenShift cluster on IBM public cloud, complete the following steps:

   a. Deploy the Voting app front end by using 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"
```

   Figure 2-6 shows a sample output for the Voting app front-end pod.



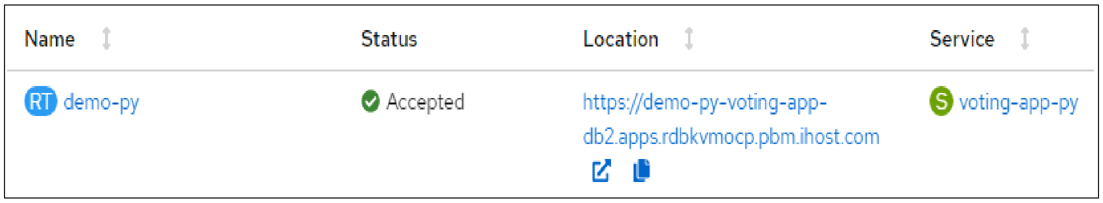
| Name | Status | Ready | Restarts | Owner | Memory |
|---|---|---|---|---|---|
| P voting-app-py-7b59f4d4d6-jwt5m | Running | 1/1 | 0 | RS voting-app-py-7b59f4d4d6 | 46.0 MiB |

Figure 2-6 Sample output for the Voting app front-end pod

   b. Create the Voting app front-end route by using the following commands:

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

   Figure 2-7 shows a sample output for the Voting app front-end route.



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

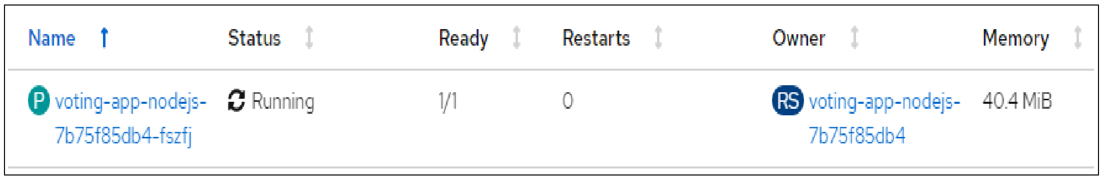
Figure 2-7 Sample output for the Voting app front-end route

5. On Red Hat OpenShift cluster on zCX, complete the following steps:

   a. Deploy the Node.js 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;p
rotocol=tcpip;uid=admin;pwd=admin"
```

   Figure 2-8 shows a sample output for the Node.js result front-end pod.



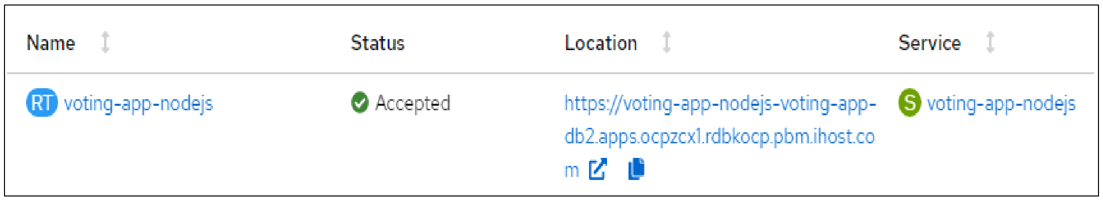
| Name | Status | Ready | Restarts | Owner | Memory |
|---|---|---|---|---|---|
| P voting-app-nodejs-7b75f85db4-fszfj | Running | 1/1 | 0 | RS voting-app-nodejs-7b75f85db4 | 40.4 MiB |

Figure 2-8 Sample output for the Node.js result front-end pod

b. Create the node.js result front-end route by using the following commands:

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

Figure 2-9 shows a sample output for the Node.js result front-end route.



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

*Figure 2-9 Sample output for the Node.js result front-end route*

6. Access the application by using a web browser:
   – To see the Voting app front end, go to the following URL:

     ```
     https://demo-py-voting-app-db2.apps.rdbkvmocp.pbm.ihost.com
     ```

   – To see the Voting app result, go to the following URL:

     ```
     https://voting-app-nodejs-voting-app-db2.apps.ocpzcx1.rdbkocp.pbm.ihost.com
     ```

## 2.3.3 Consideration

- Latency is an important aspect to address in any cloud-native application:
  – Platform matters.

    With a hybrid cloud with IBM Z technologies, the application can scale in three dimensions (horizontally, vertically, or both), which achieves superior performance and meets SLAs.

  – Collocation of cloud-native applications and data within or closer to IBM Z.

    By collocating supporting applications on the same IBM Z platform as core systems of record (SORs) applications, you can leverage the data gravity and reduce latency, and provide better security, availability, and scalability. For more information, see Collocate applications pattern.

  – Introduce memory cache.

    Caching is a key implementation when it comes to production-level deployment of services that helps to increase the application performance by acting as a middle layer between the application component and the persistence system. In this sample application, we used 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 also can be performed on IBM Z by using the same DevOps tools, and provide the same DevOps experience that is on other platforms. A range of solutions helps integrate systems, empowering developers with an open and familiar development environment with enterprise-wide, platform-neutral standardization, which helps developers build, test, and deploy code faster. For more information, see Chapter 5, “Modernizing Enterprise DevOps” on page 59.

3

# Modernized data access architectures

This chapter describes how to modernize data access architectures.

This chapter covers the following topics:

- Data fabric as the basis for modern data access
- Enabling modern access to IBM Z data patterns
- Virtualize IBM Z data pattern
- Cache IBM Z data pattern
- Transform IBM Z data pattern

## 3.1 Data fabric as the basis for modern data access

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

Most of today's successful businesses already understand this fact, but unfortunately they do no always take steps to extract more actionable information from their data. They are not planning any data strategy or dealing with the extra costs of poor data quality.

The core of a typical IT solution is the system that supports the core business, which is complemented by other subsystems that work on specific subtasks. The core also includes leveraging new technologies, cloud computing, mobile and social technologies, and security; incorporating the unstructured data that is generated by IT systems that pervade all aspects of life (called *big data*); and the analysis of the totality of these items.

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 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 everyone will use it everywhere.

One of the most serious challenges to maximizing the usage 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.

Most companies already are distinguishing between structured and unstructured data and considering how to link them together. But, there is still the challenge of data diversity.

A typical example is when you must run reports and analytics on mixed-source data. Combining different data sources is not easy, but processing also is 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 eventually becomes a problem.

In addition, recent events brought a new challenge, that is, the organizational transformation of companies. Because of the COVID-19 pandemic, many office workers are working remotely, and many of them might continue to do so. We also see more mobile workers and ones that work without a permanent office. Of course, their data needs also must be met. Most of these employees with changing work styles generally want to use data in a self-service way. A new approach that is called *data fabric* can provide answers to these challenges.

### 3.1.1 What a data fabric is

A data fabric is not a single product or 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 by 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 users will use the data fabric to spend more time analyzing data than manipulating it. Data consumers will have access to integrated, high quality, and usable data.

Figure 3-1is a high-level representation of data fabric integration.

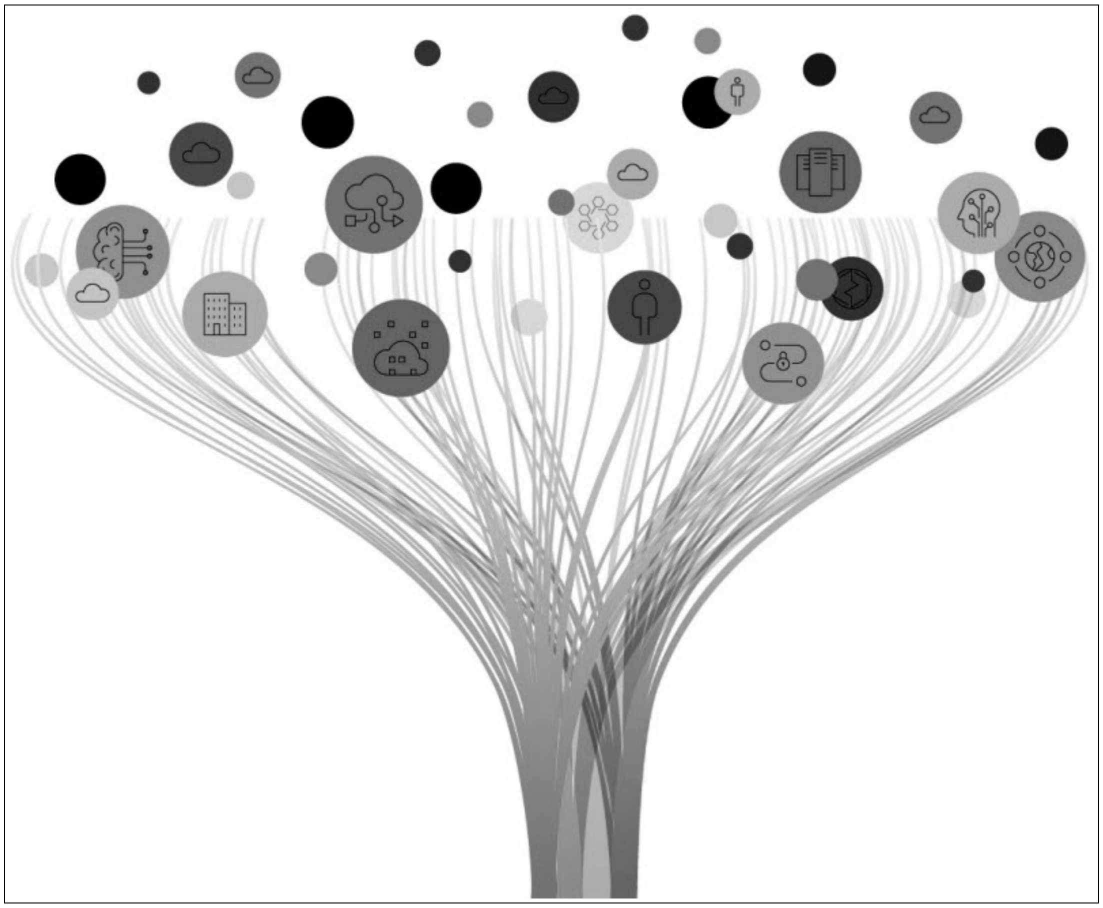


*Figure 3-1 Data fabric*

The data fabric architecture is needed to make it easier to find data in a reliable and accessible way for anyone. With the data fabric, decision makers can 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

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 while trying to avoid the problem where 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.

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 approach that we call the data fabric.

An example of a data fabric architecture is in a multi-cloud environment, for example, where 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 might be used.

The data fabric architecture can integrate these environments to provide a single view to decision makers, who then can see relationships between data that were previously unknown.

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

However, different data fabric architectures have common elements:

- Data organization

  Data organization is where some of the most important tasks of the data fabric occur, such as 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 and data access

  Data management and data access enforce data policies and maintain data quality. It helps users establish policies, processes, and accountability, and ensures that data quality remains satisfactory. It enables data usage, ensures that users in each group or department have the right to access the data they need, and ensures that everyone has access only to the data that is relevant to them.

- Data preparation and data quality

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

- Data integration and data processing

  Data integration and data processing take data from different sources and combine 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

  With data discovery, you can 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 and data visualization

  This layer is where the data is examined to identify trends and draw conclusions about the information that is contains. Data visualization is a way of seeing what the data tells us. Rather than being presented in numerical, tabular, or other formats, data is presented graphically in charts and graphs, which 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 brings business benefits only 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.

Here are the some of the advantages of data fabric architectures:

- Intelligent integration.

  The implementation uses metadata, machine learning (ML), and artificial intelligence (AI) to unify data of different data types and endpoints, which 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 using a tedious or error-prone coding process.

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

- Democratizing data, and self-service data retention.

  Data fabric architectures facilitate self-service applications by 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 users who need it. It helps business users use the data themselves by 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, and using active metadata.

  Widening access to data should *not* compromise data security and privacy. In fact, it means introducing more data management barriers around access control, ensuring that certain data is available only 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 AI and ML technologies can increase the level of automation, which enables organizations to create and implement data governance policies that ensure the ethical usage of data wherever it is at 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, which reduces 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 centralized master and metadata management, you minimize the risk of incorrect data entry, increase accuracy, and improve decision-making speed. New ways of accessing data create previously unavailable decision and reporting capabilities for companies to better understand their customers' needs and position their products and services from a new, comprehensive view.

  Using the data fabric has the following benefits:

  - Enables the creation of customized and reliable customer views.
  - Standardizes management policies and processes run times with workflow automation features.
  - Provides more valuable and accurate reports on customer interactions.

- Data fabric functions that are supported by AI.

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

  Using AI has the following benefits:

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

### 3.1.4 How IBM Cloud Pak for Data helps you 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 data analysis; and accelerates the flow of AI throughout the enterprise.

IBM Cloud Pak for Data can connect data from disparate data sources and run workloads in a 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 by providing the following benefits:

- Uses the services that are 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 on page 27 shows the IBM Data Fabric approach.

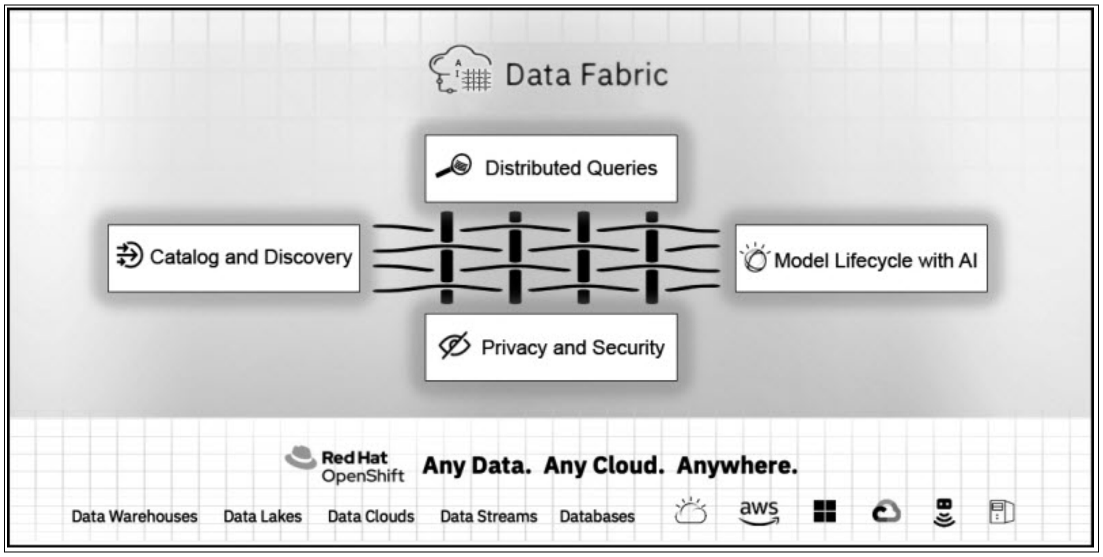


*Figure 3-2 IBM Data Fabric approach*

This platform provides users with the foundation to have up-to-date and 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 that is required by data fabrics. The following capabilities support the design and implementation of data fabric architectures:

- Metadata-based knowledge core

  This core facilitates the discovery of data sources and catalogs, and it enriches data sets. It also performs various analyses by 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 catalog 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 also can 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

  Catalogs and maintains metadata, defines privacy policies, maintains data, records data provenance, and performs other tasks that are related to security and compliance.

  This layer understands the different data formats (such as structured or unstructured data) and the meaning of the data (such as public or proprietary data). Apply the appropriate security policies to each piece of data and each user. Rather than manually applying standards and rules to data, this integrated capability means that they are applied at the organizational level and to the appropriate data sources. Analysis models in different tools can communicate with each other, and enforcement of data policies at the elementary level can be highly automated.

- Unified development and operations

  Enables unified lifecycle tracking across all components of the data platform, automates configuration, and runs in production.

## 3.2 Enabling modern access to IBM Z data patterns

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 using 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 infusing AI models and historical data, within service processing. You can dramatically simplify application development by using broad application programming interface (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, as shown in Figure 3-3. 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 use information from core Db2 and IMS systems can use SQL to detect the underlying data format and contexts from SORs, which often are required.

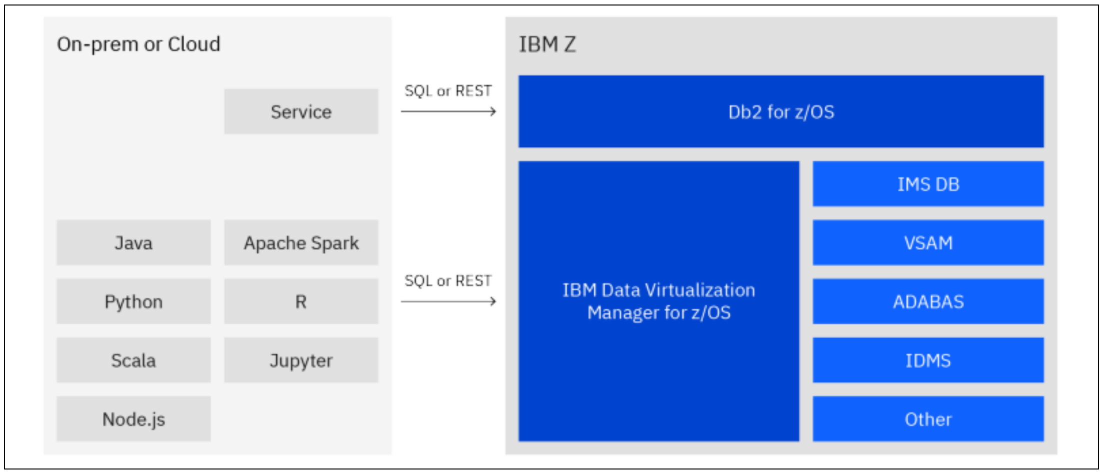


*Figure 3-3 Real-time access to transactional data*

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 shows the various means of access that z/OS Connect EE can use.

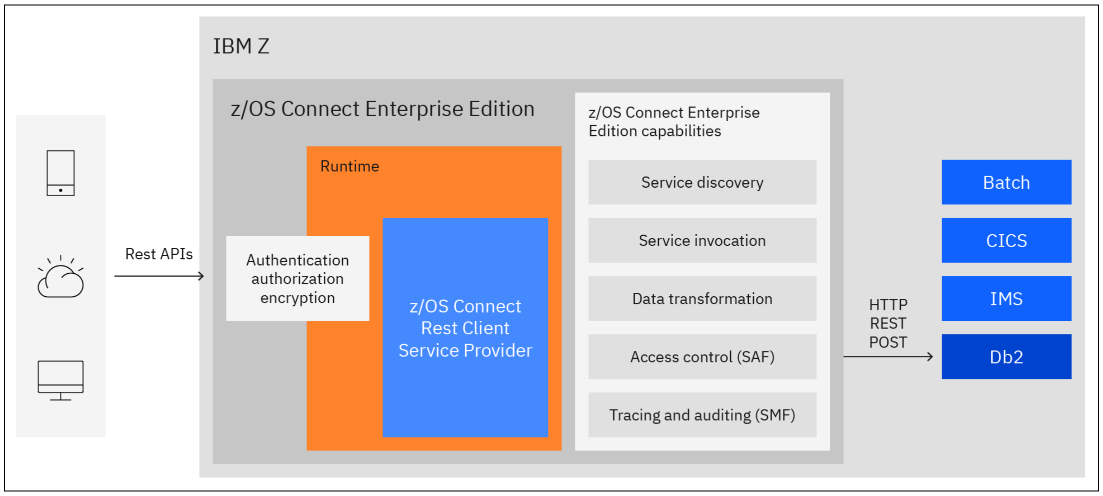

IBM Z
Rest APIs
z/OS Connect Enterprise Edition
Runtime
Authentication authorization encryption
z/OS Connect Rest Client Service Provider
z/OS Connect Enterprise Edition capabilities
Service discovery
Service invocation
Data transformation
Access control (SAF)
Tracing and auditing (SMF)
HTTP REST POST
Batch
CICS
IMS
Db2

*Figure 3-4 z/OS Connect EE capabilities*

With z/OS Connect EE, you can use IBM Z data that is stored in Db2, IMS, or data sources through a 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:

- 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 because 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 because the pattern 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/write updates through APIs. Frequently, IBM Z data also must 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 operational data store (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 using 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, JDBC, and a 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 access through the REST API, you can add z/OS Connect EE.

The term *data virtualization* is overused. 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 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, ML, and deep learning.

As shown in Figure 3-5 on page 31, 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.

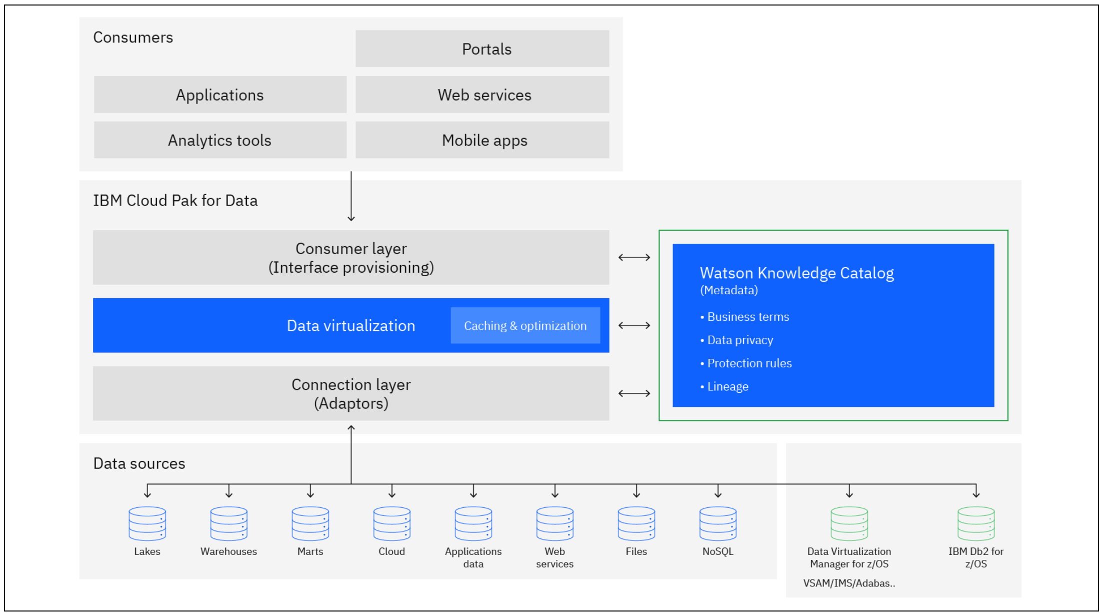


*Figure 3-5 Data Virtualization Manager*

IBM data virtualization is designed as a peer-to-peer computational mesh, which offers an 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 parallel 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 th development of 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 that are described in “Advantages” on page 29.

### Considerations

Data virtualization in IBM Cloud Pak for Data is the foundation for rapid ML model development and deployment by infusing AI into business applications, as shown in Figure 3-6.

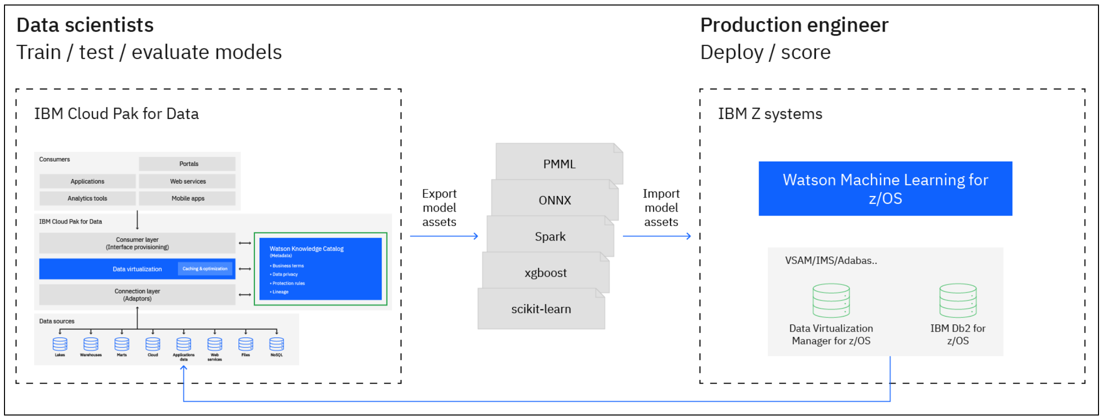


*Figure 3-6 Data virtualization and machine learning*

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. Then, you can deploy AI models to Watson Machine Learning for z/OS to address more complex information needs within business services that run on z/OS.

For more information about deploying and using the IBM Data Virtualization Manager for z/OS, see *IBM Data Virtualization Manager for z/OS*, SG24-8514.

## 3.4 Cache IBM Z data pattern

The surge of new digital applications is driving growth in data access. Those new applications typically run read-only queries of up-to-date data, but they are not 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 it difficult to plan and size for the following situations:

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

To address those challenges, organizations used several methods to extract data for use on other platforms. It did not matter whether the applications were analytic or simple query-type access because 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 extract, transform, and load (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.

Because these applications are often customer-facing, the data must be up to date and only moments behind a transactional system. This requirement drives the need for more complex application logic that ensures data currency. In addition to the processes to refresh the read-only data store, some organizations use customized code to ensure this consistency, which adds more complexity, instability, and cost to many environments.

Figure 3-7 shows the common approach to new, highly intensive, and read-only applications.

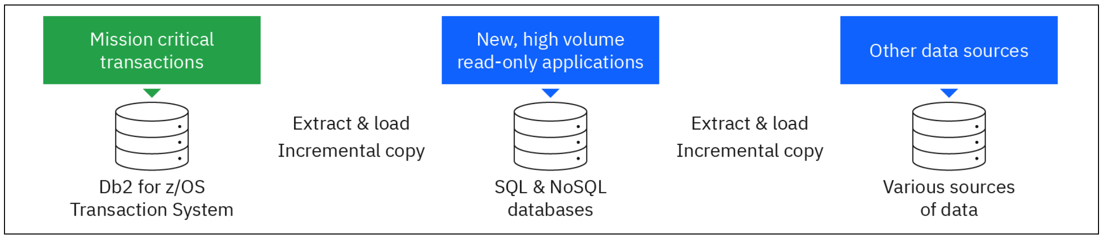


*Figure 3-7   Cache IBM Z data pattern*

Many IT organizations consider these applications necessary to support customer service but want to minimize their associated cost. However, these applications often are customer-facing, so 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 (zDIH), 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/A | N/A | N |
| Ad hoc analytic processing on data that is stored in the Table Accelerator through Db2 for z/OS | N/A | Y | N/A | N |
| Access to Db2 for z/OS (and other) data that is outside of IBM Z | N/A | N/A | Y | N |

| Data access type | Db2 for z/OS | Db2 Analytics Accelerator | Db2 Data Gate | IBM Z Table Accelerator |
|---|---|---|---|---|
| High-speed access to relatively static data from within an IBM Z infrastructure (CICS, Cobol, and others) | N/A | N/A | N/A | Y |
| Offers general processor offload | (Requires IBM Z Integrated Information Processor (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 | IBM Db2 Data Stage | IBM Data Virtualization for z/OS with Cache Options | Change Data Capture |
|---|---|---|---|---|---|
| Use case | Use current and consistent IBM Z data on a modern platform. | Accelerate analytical queries on Db2 for z/OS. | Automate ETL. | Data integration and virtualization, make IBM Z data accessible and consumable for new users. | Stream data from data sources into a data lake and warehouses. |
| Is data copied or kept in place? | Copied. | Copied, but kept In place if you use Db2 Analytics Accelerator on IBM Z. | Copied. | In place or in memory. | Copied. |
| If copied, what is the typical latency? | 1 - 10 seconds. | 1 - 10 seconds. | Hours or days. | No copy. | Depends (30 seconds for warehouse, and 1 - 10 for OLTP). |
| Query performance expectation | Depends on the target. | Optimized for transactions and analytics (hybrid transaction/analytical processing (HTAP)). | Depends on the target data store. | Data is moved or re-accessed on every query. | Depends on the target. |

| Description | Db2 Data Gate | Db2 Analytics Accelerator | IBM Db2 Data Stage | IBM Data Virtualization for z/OS with Cache Options | Change Data Capture |
|---|---|---|---|---|---|
| Data ownership and access control. | Target store. | Db2 on z/OS. | Target store. | Source. | Target store. |
| IBM Z Integrated Information Processor (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, or flat files. | In-memory virtual tables. | Data warehouse, data lake, or flat files. |
| Continuous replication? | Y. | Y. | N. | N/A. | Y. |

Figure 3-8 shows how the approach that is shown in Figure 3-7 on page 33 can be improved.

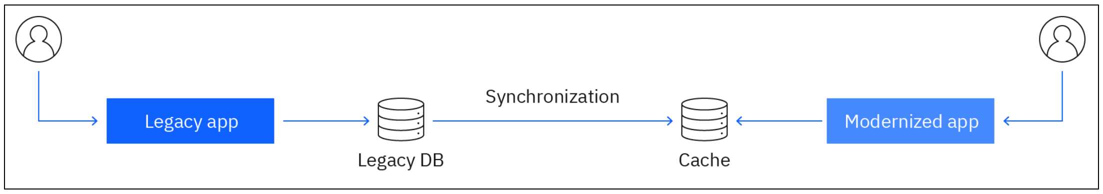


*Figure 3-8 Cache IBM Z Data: improved approach*

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 does not 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.

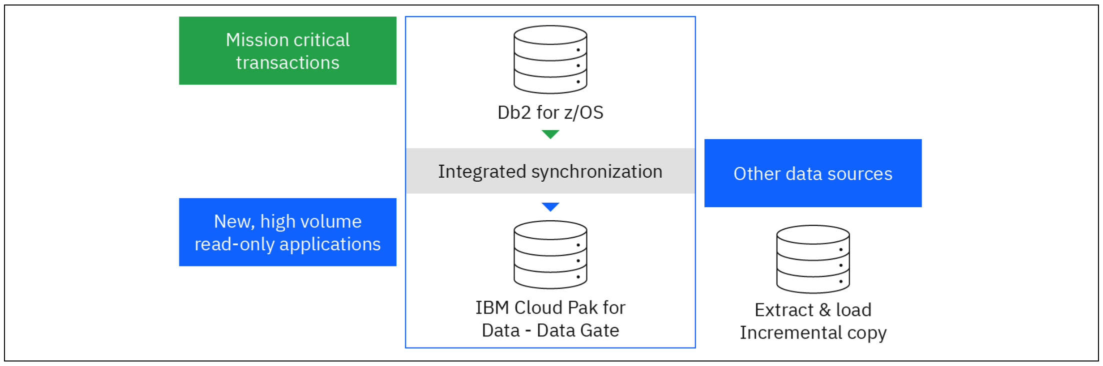

Mission critical transactions
Db2 for z/OS
Integrated synchronization
Other data sources
New, high volume read-only applications
IBM Cloud Pak for Data - Data Gate
Extract & load Incremental copy

*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, a REST API, and JDBC or Open Database Connectivity (ODBC). However, Db2 Data Gate provides the best 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 contacting databases or other data sources every time for the same request.
- Integrated and lightweight data synchronization.
- Excellent performance and resiliency.
- Lower latency and better data currency.

Although caching is commonly used to improve application latency, a highly available and resilient cache also can 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 on the nature of the specific application. For example, a data warehouse reporting system that runs a balance sheet and income statement after a close does not 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 are leaving the Db2 for z/OS environment, the only security that you can depend on is the security of the target system to which the data is being moved. You must 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

Here are examples of typical data-caching scenarios:

- 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 or protect the original source system from new, read-intensive workloads.
  - Save processing costs on the mainframe.
  - One-way synchronization to avoid data integrity problems, and the need to update conflict resolution.
  - Achieve data and function colocation.
  - The low latency data access is important for hybrid cloud scenarios where the on-premises SOR must stay as-is.
- Low-risk modernization:
  - Keep the 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 the original source or SOR to avoid any form of data integrity issue.
- “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 as a replicated cache that is maintained by the data team, and provided “as a service”.
- Mobile applications need current and low latency data but have high variability in workloads (day, night, and weekend). Extend the transactional system with a new mobile app where the back end is developed as cloud-native, but data access is to the SOR as current data (second accurate) is needed.

- Data cache as a data delivery method for a data fabric:
  - Even though data is copied, the fabric helps to enforce lineage, security, and policy rules to make sure that, for example, only in-country access to data is possible.
  - Also, since cached data is registered in the catalog, including relation to the original source (connection), this situation helps “smart data integration” use cases where applications decide whether direct access or cached access is better.
  - Data discovery: Represents cached data in a data catalog. Assigns business terms. Policy rules can be applied to the discovered metadata (that is, this column is PII, and mask PII with every access that is not in-country).

## 3.5 Transform IBM Z data pattern

The transformation of SOR data by software processes to create a data set is a specialization of a broader copy-and-access use case. That use case also includes ETL processes, software replication processes, and virtualization and federation processes. All those broader processes include some transformation capabilities, and it is 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 usage 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. Although this pattern applies to SOR data on any platform, its usage with SOR data on IBM z/OS is relevant and valuable because many large enterprises use z/OS as an SOR.

In Figure 3-10, the transformation processes are indicated by the Data Adapter box. In practice, these transformations can span both processes and time.

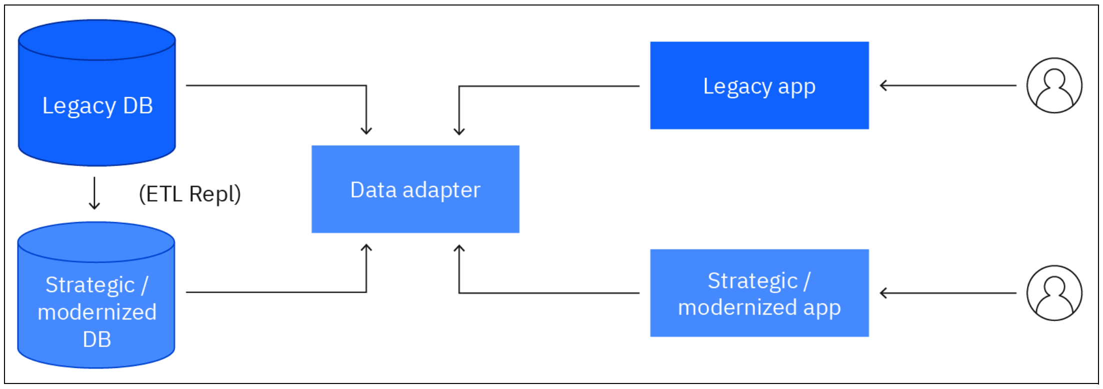


*Figure 3-10 Transforming data*

## 3.5.1 Data transformation solution and pattern for IBM Z

On z/OS, it is 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 itself. Add the ability to extend the aggregation to derive data (summations and transformations) and to add sources (distributed databases and 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 IBM Db2 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 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 from which they are derived.
- 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 must 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 IBM z/OS

In this chapter, we describe event-driven architectures with event-driven solutions.

This chapter covers the following topics:

- Overview of an event-driven architecture
- Introducing the event-driven architecture in the z/OS ecosystem
- Conclusion

## 4.1 Overview of an 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 is necessary for businesses to reinvent themselves in a competitive world by developing inter-operable, loosely coupled, and high-throughput systems that offer better, reliable, and near-real-time user experiences (UXs), which requires an event-driven architecture.

From a non-technical perspective, an *event* can be considered as anything and everything happening or changing around us. An event-driven architecture builds a system that 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 architectures are rising to popularity because of its ability to complement microservice architecture styles 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.

For more information about event-driven architectures, see Application modernization for IBM Z architecture.

### 4.1.1 Simplified reference architecture

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

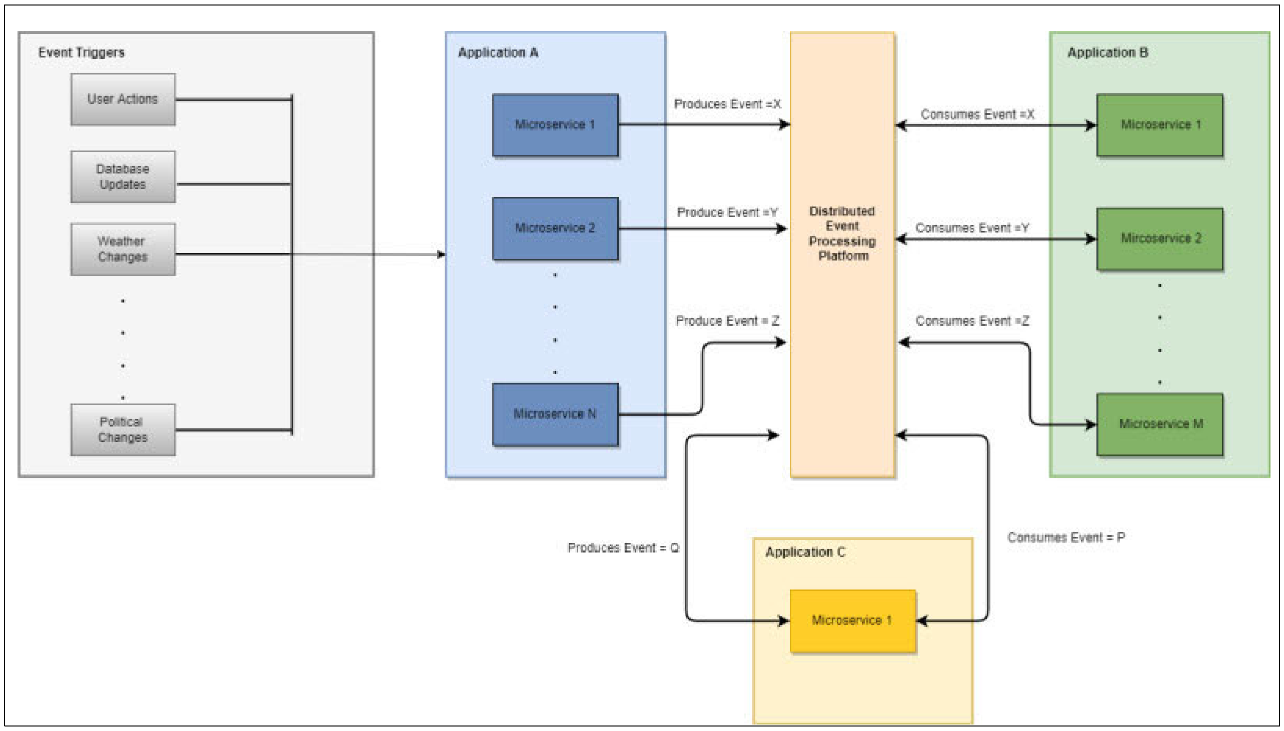


*Figure 4-1 Simplified event-driven architecture*

The basic components of simplified event-driven architectures include:

- *Event triggers* might be part of the technical system. They cause changes in state for a subject of interest in real world. Triggers might be technical, political, environmental, or others. Depending on the scope of th application or project, we can accommodate the triggers that we want to cover.
- *Producers* are hosted either on the same server or distributed servers. They capture real-life events, convert them into a technical format, and emit them to an event-processing platform that can be understood by other technical systems.
- An *event-processing platform* is a distributed system for high availability that acts as temporary storage for an event to ensure that an event goes to the correct consumers. It also can be visualized as message broker among disparate systems.
- *Consumers*, hosted either on the same server or on distributed servers, continuously poll the event-processing platform. Generally, it is designed so that each consumer is a microservice that is triggered only when an event of interest is found. Sometimes, consumers also can act as producers to publish separate events, which cause a chain of events, as shown in Figure 4-1 on page 42 (see the microservice inside Application C).

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

- Message topics are generally used in the publish/subscribe model, where each message can be used by multiple subscribers. For example, a business publishes a message to all customers that are subscribed for promotional events regarding holiday offers.
- *Kafka* is used when we expect high throughput, that is, processing a large load of data in less time. The processed data is used to feed an analytical engine to build intelligence around information. For example, depending on videos that are seen by users, the engine builds a user profile and recommend new videos.
- *File systems* are used to automate processes to avoid manual intervention. For example, receiving an invoice at an FTP location triggers a bot to send an email to vendors.
- Message queues are generally used when we must maintain the sequence of each message while processing. For example, in supply chain management, the shipment goes through different statuses, such as departed, booked, landed, and delivered. These events must be processed in order without jumbling.

### 4.1.2 Types of event processing

To discover a use case and design, we must categorize events as follows:

- Simple Event Processing typically detects a change in state in the context of a specific workflow or business process, which determines the next action. The scope of event is limited to current and next state only. For example, in a payroll system, an employee submitting a leave application triggers a simple event, which is used by the approval process and the salary distribution process.
- Complex Event Processing typically detects and correlates a change in state to other past or future events, which establishes a pattern to determine the next 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 notifies the user about a possible plumbing issue.

- Event Stream Processing is typically used when high throughput is expected to build an intelligence in real time over streams of events that are received. For example, in GPS, when all cars on a road slow down due to traffic, the GPS device in each car emits an event about th current speed. The system correlates all these events and modifies the route color from blue to yellow or red to signify delays, on th GPS user interface.

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

In this section, we provide a detailed description of a couple of patterns. We use a sample scenario and describe the current versus proposed architectures, their benefits, some considerations, and provide a brief implementation guide. The patterns that we describe in this section are as follows:

- Respond to IBM Z application events pattern

  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.

- Optimize CQRS pattern

  Deliver an efficient CQRS system that is based on IBM Z to optimize the synchronization between the command access, which is system of record (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 (zDIH) to deliver a non-disruptive, low-latency, high-throughput, and cost-attractive solution.

An extra pattern that is not described 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 in to the core applications.

### 4.2.1 Respond to IBM Z application events pattern

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

#### Sample scenario

A bank with thousands of customers that processes millions of transactions a day decides to extract meaningful information from their customers’ credit card transactions to serve customers better and increase customer satisfaction without disrupting business as usual (BAU). This task includes establishing a prototype that ingests credit card transactions in real time to build an intelligence model. This prototype is meant to be a foundation that might further be extended, enriched, and enhanced for the following example use cases:

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

Therefore, in the scope of design and implementation, we describe different alternatives to integrate an event-driven architecture with z/OS; provide instructions to set up an infrastructure that is suitable for one of the designs; deploy services to connect with the event-processing platform; and provide sample code that produces and consumes events. We do not provide any business logic in this example.

## Typical as-is (current) architecture

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

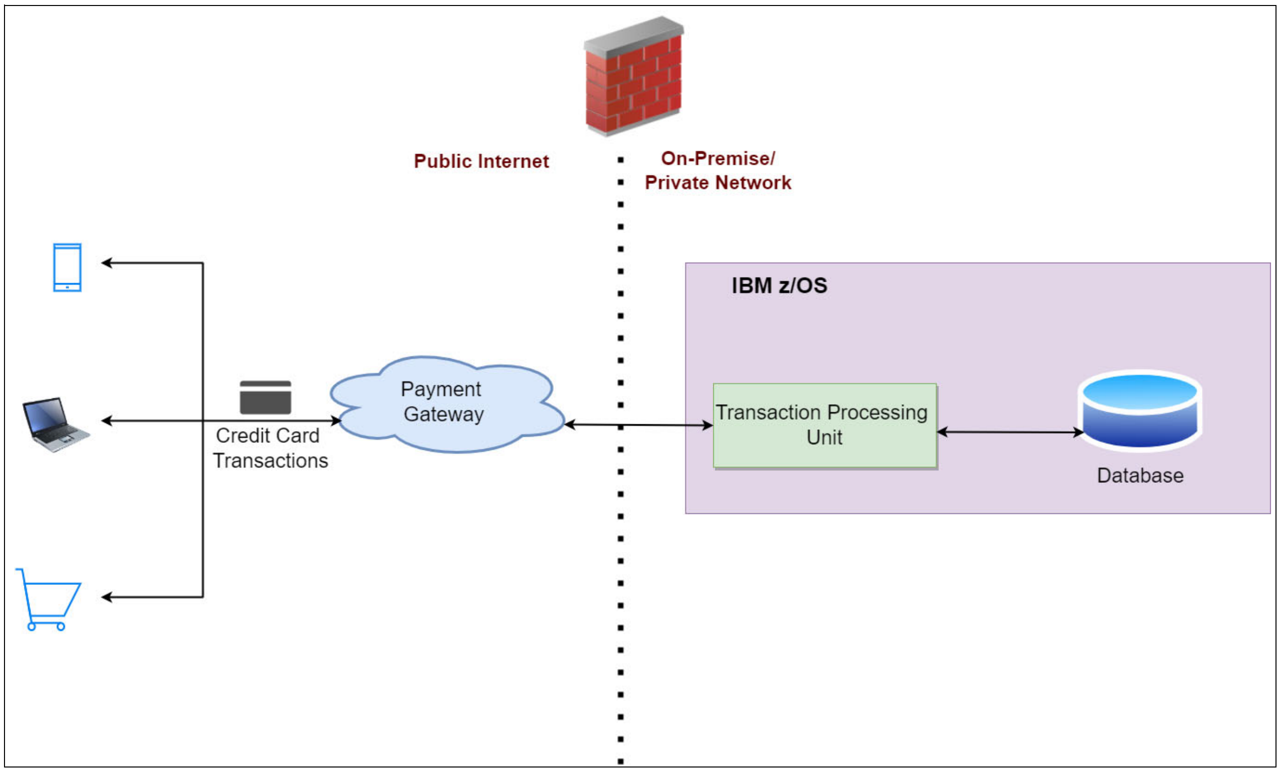


*Figure 4-2 Typical as-is architecture*

Here is the simplified processed that is used in a typical as-is architecture:

1. A user makes a transaction by using a mobile device, a laptop, or any other means, over the public internet.
2. The payment gateway sends the transaction to the bank to process it.
3. The transaction processing unit, which is hosted on IBM z/OS, receives, processes, and approves or rejects the request.
4. The result is returned to the payment gateway and stored in a database.

We enhance this system by feeding data that is received by the transaction processing unit to more components.

## Proposed architecture: Phase 1

To leverage the power of z/OS with a hybrid cloud, we 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.

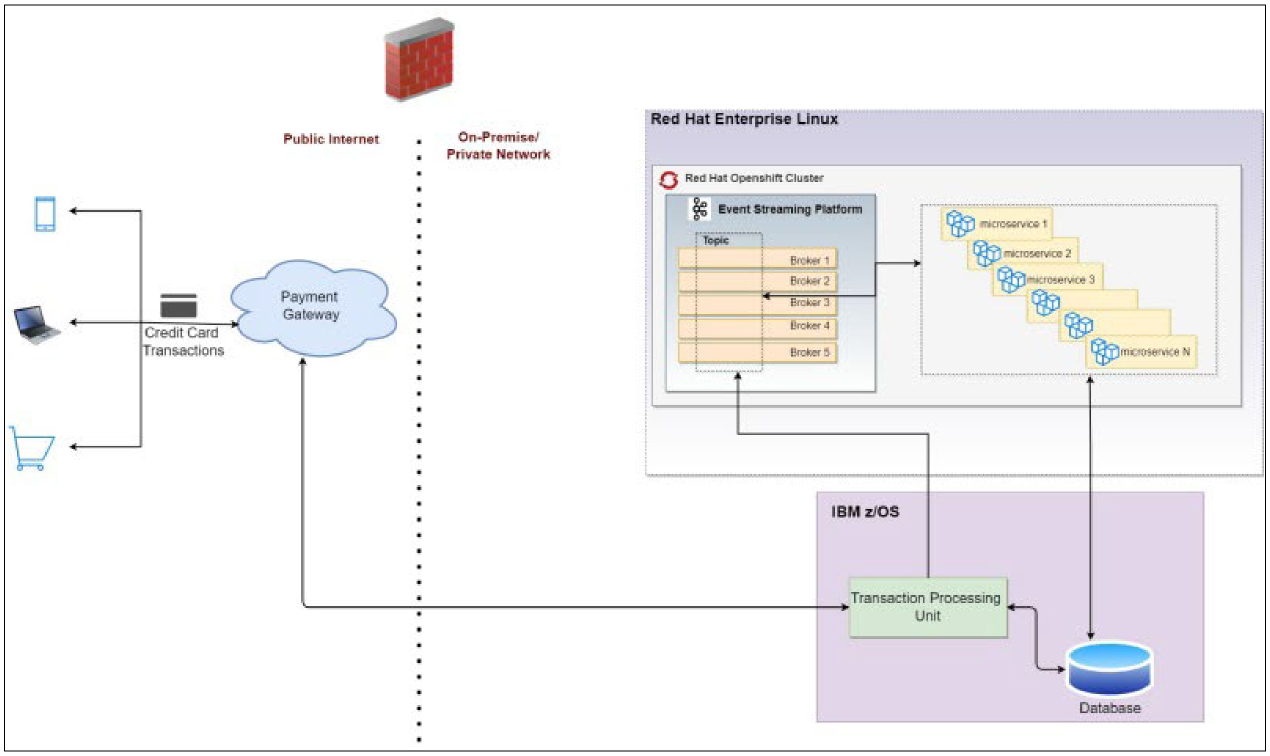


*Figure 4-3 Proposed architecture: Phase 1*

Figure 4-3 introduces the components to ingest existing data. The new setup is put on a separate Red Hat Enterprise Linux box, either on-premises or on a private network, and decoupled from the existing system, which adheres to the principle of not disrupting the existing system (BAU). The setup also helps us to measure the performance matrix separately for a new implementation, along with separate application management (deployments, logging, and monitoring) if needed. The containerization of the event streaming platform and microservices in Red Hat OpenShift makes the implementation platform independent, for example, the solution provides flexibility to deploy Red Hat OpenShift on-premises or on cloud, if needed with the cloud provider of the customer's choice.

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

The existing transaction processing unit is extended to emit events for each transaction that is made on the event streaming platform. Containerized microservices are continuously polling the event streaming platform. When an event is received, the respective microservice of interest triggers and runs the business logic.

## Proposed architecture: Phase 2

Once we have functioning prototype, we perform a "lift and shift" of components from one Red Hat OpenShift cluster on Red Hat Enterprise Linux to another cluster on z/OS, as shown in Figure 4-4 on page 47.

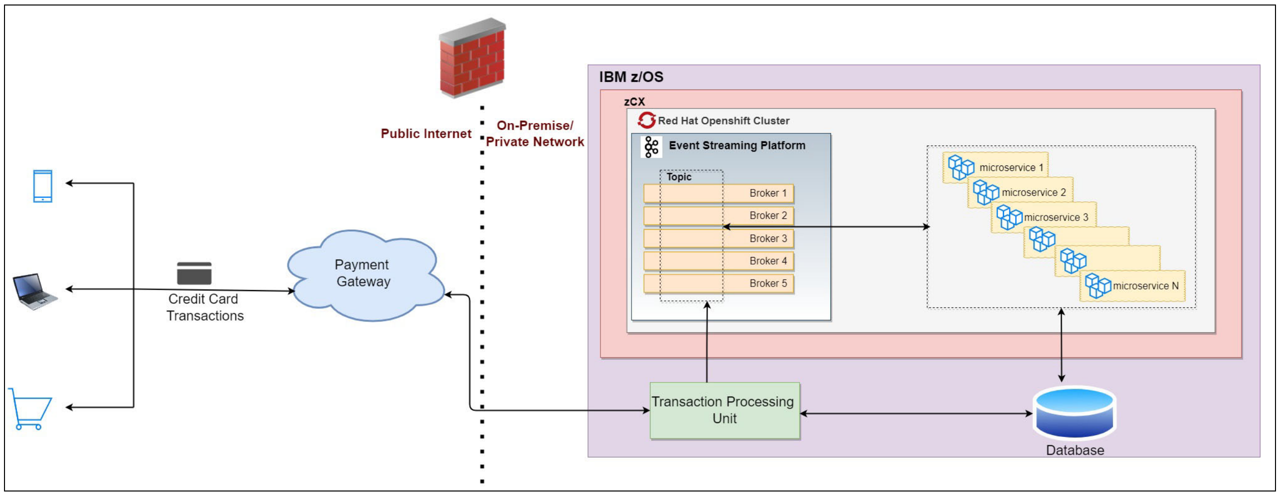


*Figure 4-4   Proposed architecture: Phase 2*

However, IBM z/OS Container Extensions (zCX) is a prerequisite to install Red Hat OpenShift on z/OS (for more information, see "Installing Red Hat OpenShift Container Platform" on page 48). From a technical perspective, the interfaces among components are unchanged, but getting Red Hat OpenShift closer to th existing system helps to reduce latency and provides closer control to the developer for existing and new implementations.

### *Benefits of proposed architecture*

The event-driven architecture that is described in this chapter combined with microservices and agile methodologies provides the following benefits:

- Asynchronous processing: Provides efficient usage of resources and loosely coupled resilient services. It can replay events if needed.
- Cross-technology/platform integration: Allows developers to choose different languages for different services and still integrate seamlessly with other modules.
- Continuous integration (CI) and continuous deployment (CD) (CI/CD): Automates the process from source code development to deployment on server or run time, which helps with scalability, reliability, and availability.
- Rapid go to market strategy: Helps businesses to react to events and act quickly without compromising any of the existing applications or systems.
- Cost-effective: Solutions and components can be rolled out in phases. Technical resources are used or scaled only when events occur and not blocked.
- Separation of responsibilities: With independently deployed components, the proposed architecture can help to detect, correct, and deploy a solution for a component without affecting other components or systems.
- Maintaining a skillset in a team: This task is easier compared to monolithic systems because components are not technology or platform-dependent

## Considerations

Consider the following items when you use this proposed architecture:

- The design is technology-neutral, giving the user the flexibility to choose technologies. The microservices can be in Java, Python, Node.js, or any other language. The database can be Db2, SQL, or others.
- Although the design depicts connectivity to a database from technical components in z/OS, the implementation of this aspect already is proven, so it is not included in the implementation guide.
- The architecture can be mirrored if events are sourced from external components by putting event streaming platform and microservices in a public cloud.
- Other existing systems, functions, and non-functional parameters are not part of the architecture to focus on event-driven architecture, which is expected to be integrated into real-time applications

## Implementation

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

For our use case, we use the following components:

- IBM Event Streams as our event processing platform
- Red Hat OpenShift Container Platform as our hosting platform
- Java microservices as our consumers of events
- A transaction processing unit, which acts as producer (simulated with another Java microservice)

### Installing Red Hat OpenShift Container Platform

To learn how to install Red Hat OpenShift Container Platform, see Red Hat OpenShift Container Platform installation interview.

To install Red Hat OpenShift on z/OS by using zCX, see *Red Hat OpenShift on IBM Z Installation Guide*, REDP-5605.

### Installing an event processing platform on Red Hat OpenShift

Your application team can choose the event processing platform as-a-service on Red Hat OpenShift, such as IBM MQ, Red Hat AMQ, Apache Kafka, IBM Event Streams, Strimzi Kafka, and OperatorHub on Red Hat OpenShift console. For our use case, we used IBM Event Streams.

To install IBM Event Streams, see Installing on Red Hat OpenShift.

To use Strimzi Kafka, see Deploy Apache Kafka in a Few Seconds with Strimzi and Operator Framework and Operator Lifecycle Manager.

### Deploying microservices on Red Hat OpenShift

Because this solution is containerized, an application team can choose various programming languages without worrying about the environment, deployment, or networking rules. Developers can enjoy CI/CD with built-in support for various buildpacks from Red Hat OpenShift, and focus more on business functions.

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

- `kafka-demo-producer`

  This microservice simulates the events that the transaction processing unit emits on the topic that is configured in the Kafka service. To obtain `kafka-demo-producer`, see GitHub.

- `kafka-demo-consumer`

  This microservice is the listener application that polls a Kafka topic, where actual business logic should be when we extend the functions. To obtain `kafka-demo-consumer`, see GitHub.

## Prerequisites

Here are the prerequisites that are needed to deploy the microservices:

- Red Hat OpenShift cluster is provisioned and the developer has sufficient access permissions or roles on the cluster.
- If the cluster is an on-premises or private network, the user is connected to the network through a virtual private network (VPN) or similar.
- Client tools to access the cluster from a command-line interface (CLI) are installed on th developer machine. For more information, see Getting started with the Red Hat OpenShift CLI.
- The server certificate is installed on the client machine to enable a secured connection. You can secure a certificate from your cluster administrator.
- Because the example is developed in Java Spring Boot, you should be familiar with it and Maven.
- A Personal Access Token is set up if the source code is referred from GitHub. For more information, see GitHub.

### *Deployment instructions*

To deploy the Java microservices, complete the following steps:

1. Log in to the Red Hat OpenShift cluster by running the following command:

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

   You are prompted for the password. After you enter it, you see an output likeExample 4-1.

*Example 4-1 Logging in to the cluster*

```
Authentication required for <server_url> (Red Hat 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 project. It is a best practice to separate services per functional domain for better maintenance. The following command creates a project in the container:

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

The output should be like Example 4-2.

*Example 4-2 Creating a project*

```
Now using project “<project_name>” on server “<server_url>”
```

3. Build and deploy the microservice. Red Hat OpenShift supports an array of options, such as using containerized images that are based on Docker. Build your Dockerfiles and push the image to a repository by using one of the following methods:
   - Run the **`docker build`** command. For more information, see `docker build`.
   - Use Quay.io to build Dockerfiles and push the resulting image to a repository. For more information, see Building Dockerfiles.
   - Perform a native build or source to image (S2I) build by using various technology stacks. For more information, see S2I Requirements.

   We used an S2I build for our use cases. This process has multiple steps:

   a. First, pull a buildpack that is compatible with Red Hat OpenShift, which is needed to build an image (in our case, a Java one) by using the following command:

   ```
   $ docker pull <JDK_BUILD_TAG>
   ```

   For example:

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

   The output should be like 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
```

   b. Run the build to create an application in the Red Hat OpenShift project. The following command fetches source code from a centralized repository; builds the deployable unit of source code by using the downloaded buildpack; and pushes the deployable unit to the Red Hat 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
   ```

   The output should be like Example 4-4 on page 51.

*Example 4-4 Creating 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
class path)
    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
```

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

```
$ oc status
```

The output should be like Example 4-5.

*Example 4-5 Validating the 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
```

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

```
$oc expose svc <APP_NAME>
```

For example:

```
$oc expose svc kafka-demo-producer
```

The output should be like Example 4-6.

*Example 4-6 Create route*

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

Perform the steps from step 3 on page 50 to step d for `kafka-demo-producer` and `kafka-demo-consumer`.

4. Testing the Kafka connection. Now your application is ready to be tested. We produce an event by using a REST end point, which should be consumed by the consumer application in real time. The business logic that is built to process simple, complex, or stream events is placed in the consumer application by using the application programming interfaces (APIs).

   To emit a simulated event to Kafka, run the following command:

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

   `SRVC_ROUTE` is the URL of the REST end point that we created for `kafka-demo-producer`.

   The output should be like 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 that accurate properties are set according to the Kafka provider documentation.

5. Monitoring the application is possible by using a CLI or an administrator console. For more information, see Monitoring application health by using health checks.

## Possible alternative solutions

The solution that is outlined here is for a client business that is 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 on page 53 is a slight variation of the design that we described in detail and showed in Figure 4-3 on page 46. The difference is that the event streaming platform is not containerized, but instead is installed directly on Red Hat Enterprise Linux or on any machine on-premises. It decouples the business logic in microservices from the infrastructure.

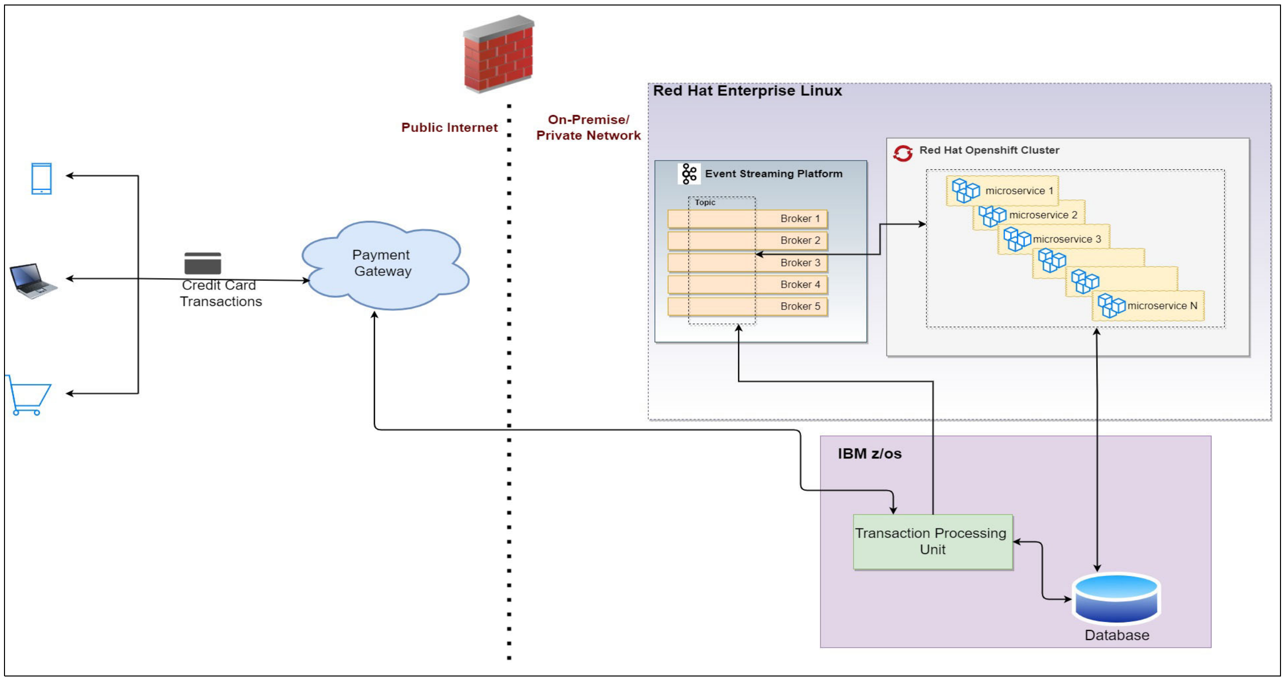


*Figure 4-5 Proposed architecture: Alternative 1*

### Alternative 2

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

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

For more information, see the following resources:

- IBM Cloud Pak for Integration
- IBM Cloud Pak for Business Automation
- IBM Cloud Pak for Data

## 4.2.2 Optimize CQRS pattern

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 SORs 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 SORs 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 CQRS. This trend includes even those industry use cases that traditionally are satisfied with latent information. Typical CQRS patterns leverage event-based mechanisms 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, and modern way to share information with hybrid cloud. The Optimize CQRS pattern is an optimized pattern for SORs that are on IBM z/OS. For IBM Z enterprise clients, the application systems include core banking, claims adjudication, wealth management, card processing, and more.

## Sample scenario

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

Across the banking industry, many fintech (technology that is 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 that is 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 SORs, 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 much 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, and the ability to achieve functional requirements such as sharing composed information.

As a more specific example, one set of information that might be highly valuable for downstream hybrid cloud ecosystem applications is running balances or account balances 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, so there is a need for an Optimized CQRS approach. The high-level scenario is shown in the Figure 4-6.

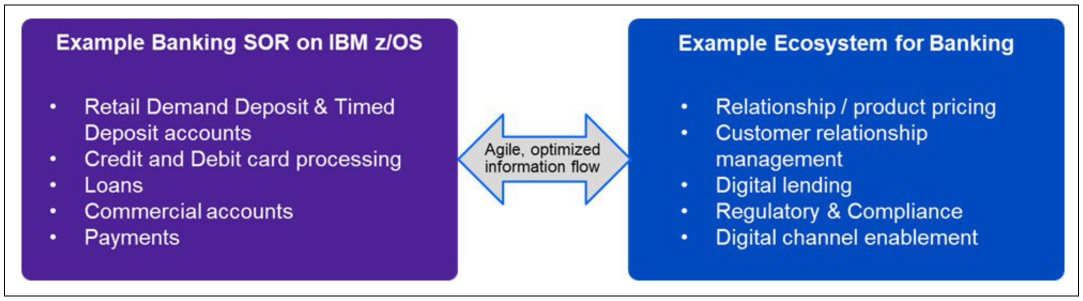

Example Banking SOR on IBM z/OS
Retail Demand Deposit & Timed Deposit accounts
Credit and Debit card processing
Loans
Commercial accounts
Payments
Agile, optimized information flow
Example Ecosystem for Banking
Relationship / product pricing
Customer relationship management
Digital lending
Regulatory & Compliance
Digital channel enablement

*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 that are sent as events to streaming architectures, such as the ones that are built on Kafka. Although 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 architecture is shown in Figure 4-7.

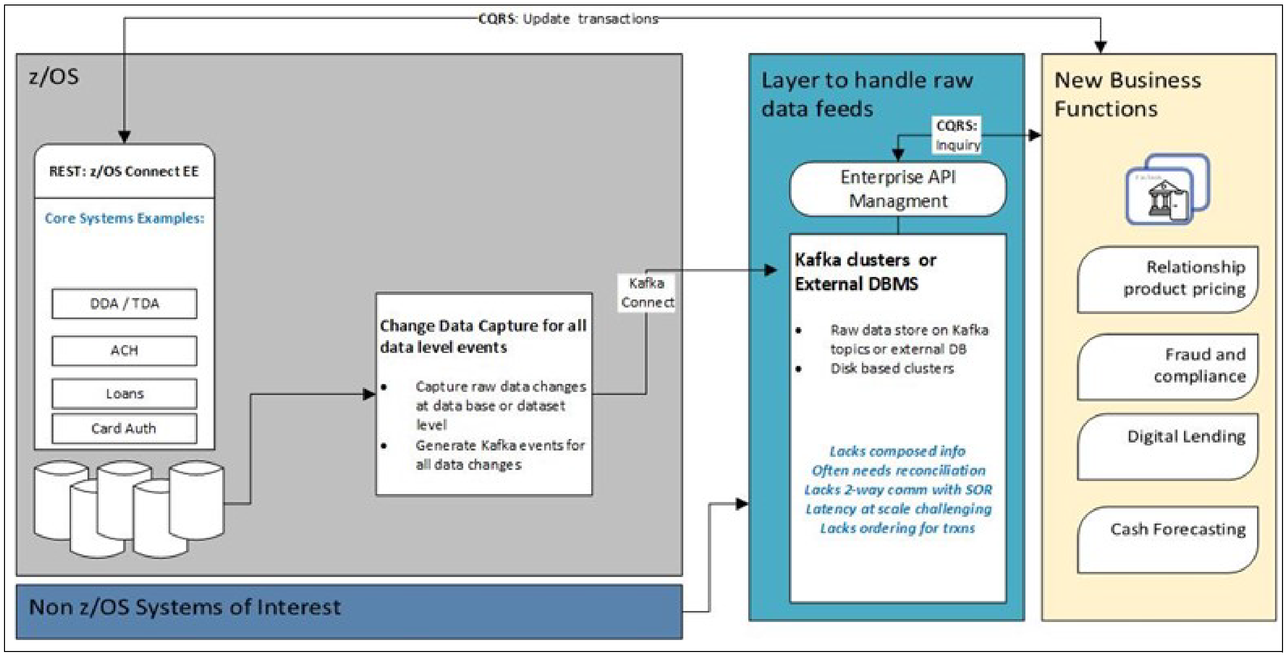

CQRS: Update transactions
z/OS
REST: z/OS Connect EE
Core Systems Examples:
DDA / TDA
ACH
Loans
Card Auth
Change Data Capture for all data level events
Capture raw data changes at data base or dataset level
Generate Kafka events for all data changes
Kafka Connect
Layer to handle raw data feeds
CQRS: Inquiry
Enterprise API Managment
Kafka clusters or External DBMS
Raw data store on Kafka topics or external DB
Disk based clusters
Lacks composed info
Often needs reconciliation
Lacks 2-way comm with SOR
Latency at scale challenging
Lacks ordering for trxns
New Business Functions
Relationship product pricing
Fraud and compliance
Digital Lending
Cash Forecasting
Non z/OS Systems of Interest

*Figure 4-7 As-is architecture*

In Figure 4-7, data that is 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 through Kafka topics. Typically, these data feeds go to a layer that ingests the events (for example, through Kafka based topics), and either populates in a different data store or are held in the Kafka cluster itself. From this point, the broader ecosystem of SaaS applications that are shown in the yellow box in Figure 4-7 consumes the data for inquiry purposes. If the SaaS ecosystem applications want to transact for an update to the core systems, they communicate directly with the core SORs application through REST interfaces, for example.

Here are some key challenges of this approach:

- For SORs 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, but 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 produce results, recommendations, scores, and other items that should be shared back with core systems. In many cases, clients prefer that communication is 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 SOR.

## Proposed architecture

In the proposed architecture, the typical CQRS pattern is optimized for high-volume transactional information that must be shared in real time. While the event streaming capability is leveraged, the content of these events is composed and aggregated into 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 through an in-memory set of caches on z/OS to facilitate both low latency needs and asynchronous two-way communication.

The proposed architecture is shown in Figure 4-8.

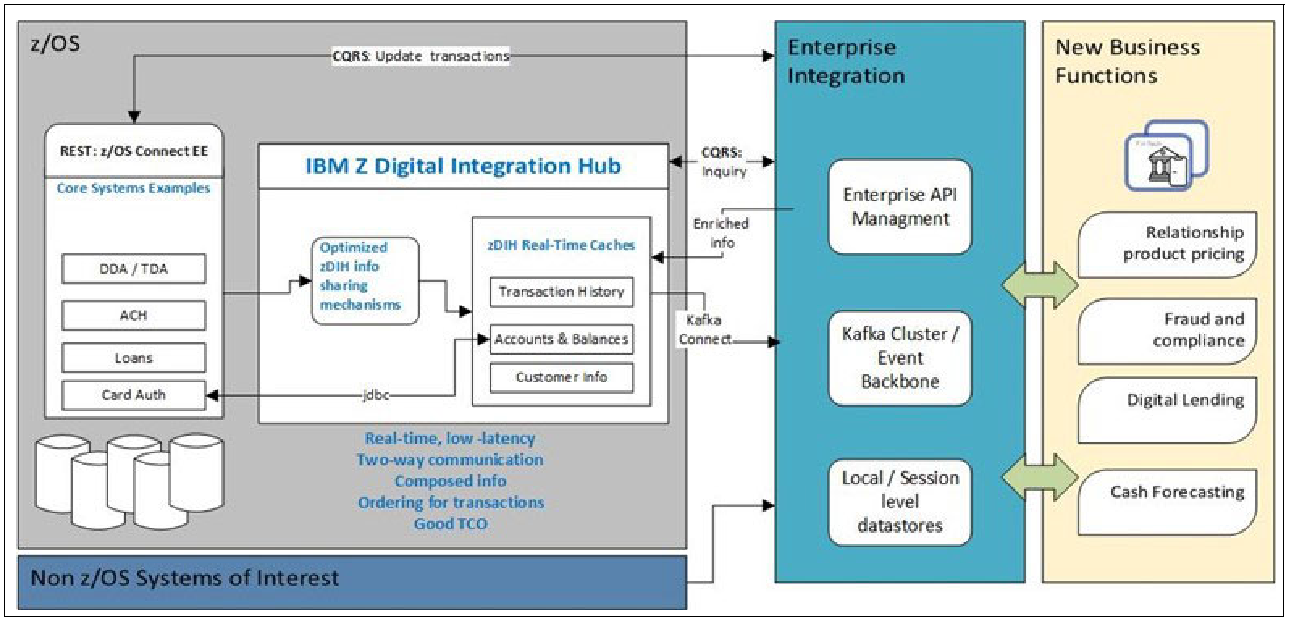


*Figure 4-8 Proposed architecture*

In this model, the SaaS solutions leverage the 'Query' parts of the architecture through the real-time in-memory caches that are on z/OS. These caches are kept current in real time with only the necessary information that is shared rather than all the raw data. The 'Update' part of the flow still is driven to the core SORs through tools such as z/OS Connect EE for REST-based interaction. The architecture facilitates the following activities:

- Fast performance with subsecond concurrency 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 through application-level integration.

- Cost advantages from avoiding recomputing of information if it is not changed and leveraging of z/OS specialty engines (IBM Z Integrated Information Processor (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 SORs on z/OS is zDIH. zDIH addresses these needs for clients that run core systems on z/OS through several key features, among which are the following ones:

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

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

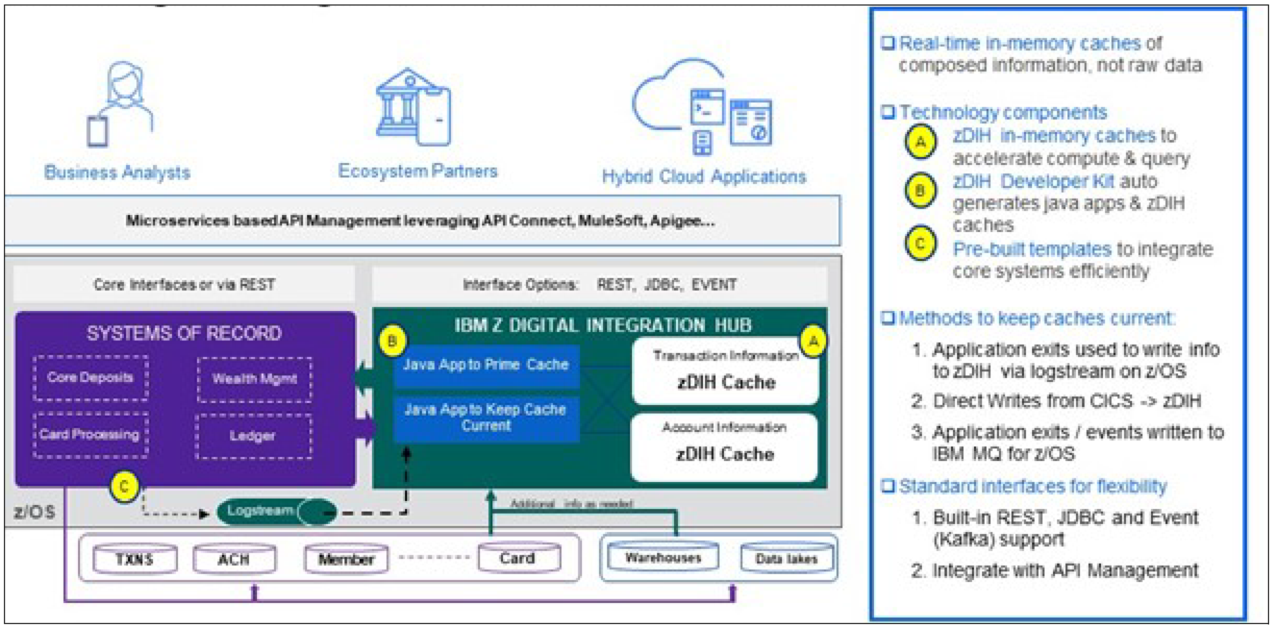

Business Analysts
Ecosystem Partners
Hybrid Cloud Applications
Microservices based API Management leveraging API Connect, MuleSoft, Apigee...
Core Interfaces or via REST
Interface Options: REST, JDBC, EVENT
SYSTEMS OF RECORD
Core Deposits
Wealth Mgmt
Card Processing
Ledger
IBM Z DIGITAL INTEGRATION HUB
Java App to Prime Cache
Java App to Keep Cache Current
Transaction Information
zDIH Cache
Account Information
zDIH Cache
A
B
C
z/OS
Logstream
Additional info as needed
TXNS
ACH
Member
Card
Warehouses
Data lakes
Real-time in-memory caches of composed information, not raw data
Technology components
zDIH in-memory caches to accelerate compute & query
zDIH Developer Kit auto generates java apps & zDIH caches
Pre-built templates to integrate core systems efficiently
Methods to keep caches current:
1. Application exits used to write info to zDIH via logstream on z/OS
2. Direct Writes from CICS -> zDIH
3. Application exits / events written to IBM MQ for z/OS
Standard interfaces for flexibility
1. Built-in REST, JDBC and Event (Kafka) support
2. Integrate with API Management

*Figure 4-9 Overview of zDIH*

zDIH can be used for various use cases in addition to optimized CQRS, and it delivers value through its implementation natively on z/OS. Specifically, the zDIH can perform the following tasks:

- Integrates through application-level exits with core SORs.
- Leverages z/OS capabilities such as logstream for z/OS to share real-time information at scale and in order.
- Uses z/OS functions of Parallel Sysplex enabled logstreams for capturing information consistently across multi-system workloads.

- Reduces skills barriers with low-code zDIH applications by using zDIH Developer Kit and Java.
- Delivers information in a more consumable manner through a flexible cache infrastructure.

For more information about zDIH and how you can start using it, see IBM 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 an event-driven architecture opens an exciting future. With it, businesses can be more competent and prepared in ever-changing competitive markets, and technical teams can be creative, innovative, and focused on business solutions by providing them with richer technical stacks and automating part of their workload. The beauty of a containerized system is that it provides a “lift and shift” approach. Depending on the use cases, we can shuffle these containers on-premises, on a private cloud, or a public cloud with the provider of the customer's choice, either by using the same provider or a mix of providers, making solutions truly hybrid.

5

# Modernizing Enterprise DevOps

After looking at many different use cases and solution examples for three groups of IBM Z application modernization patterns in this book, this chapter explores the Enterprise DevOps patterns, also known as the Enabler patterns for the other three pattern groups in the framework. These Enabler patterns define best practices and solution patterns for organizing the development teams' ways of working and 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, which are composed of cloud-native and z/OS application components that were presented as examples in the previous chapters.

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

1. Important best practices for DevOps and how they apply to hybrid development projects, which are described in 5.1, “Core practices of IBM Z DevOps for hybrid enterprise application development” on page 60.
2. Key tools and technologies that help enable development teams to realize these best practices, which are described in section 5.3, “IBM Z Cloud and Modernization Stack: A layered development tool architecture adding incremental capabilities” on page 70.
3. An end-to-end example project that uses these practices and technologies, which are described in 5.4, “A next-generation developer end-to-end development example” on page 88.

This chapter covers the following topics:

- Core practices of IBM Z DevOps for hybrid enterprise application development
- The vision for a cloud-native developer experience for z/OS enterprise applications
- IBM Z Cloud and Modernization Stack: A layered development tool architecture adding incremental capabilities
- A next-generation developer end-to-end development example
- IBM Wazi as a Service and IBM Z and Cloud Modernization Stack tutorial

## 5.1 Core practices of IBM Z DevOps for hybrid enterprise application development

When development organizations start with enterprise modernization and adopt DevOps, the first thing to review and reform are the development practices that are used in the organization. Every development team follows some kind of process. To create a hybrid development team that can work on hybrid applications, this process or the various processes of the previously siloed teams must be assessed and compared to industry best practices and technical solutions that enable them.

The practices that are described here were collected over the last few years by the IBM Z DevOps offerings group that is led by Rosalind Radcliffe. For more information, see the following resources:

- DevOps from APIs to IBM Z For Dummies
- *Enterprise Bug Busting: From Testing through CI/CD to Deliver Business Results*, by Radcliffe
- Enterprise DevOps pattern
- IBM Garage™ Method

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

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

### 5.1.1 Standardizing and automating your development setup

Before any CI/CD automation can take place, all developers in the hybrid team must have the same baseline for working on application changes, which includes access to artifacts such as source code and scripts to build and debug the application components that they are working on, and standardizing the setup and configuration of the 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 best practices for standardization 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 because they contribute significantly to a developer's job satisfaction.

However, standardization must happen on the artifact formats that are produced by these developer environments because they must be uniform and platform-independent. For example, if the team is composed 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 that is used for source code files (such as UTF-8 without any special characters allowed outside of strings), and 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 is to standardize platform-independent tools when developers work with different editors around source code formatting, which is essential for the 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 that are provided by different editors, then difference-views that are used for merging would show many changes that are based only on formatting and not actual code changes. A simple thing such as having trailing white spaces and tabs versus spaces can become a huge productivity killer for a development team. An editor-independent tool such as Prettier can be used to format code within many editors or outside of the editor with command-line interface (CLI) operation. Prettier can 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, so teams must find the right compromises to ensure such standards in different ways.

Another best practice is for a hybrid development team to provide a set of well-documented and proven standard configurations for a developer to choose, at least as a starting point. Here, it is 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 work with 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 that are used by the team, 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 following sections, in which we are exploring the IBM Z Cloud and Modernization Stack and the End-to-End Next Generation Developer Walkthrough, we describe some configuration and automation examples for hybrid development teams.

### 5.1.2 Maintaining a single source code management system

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

Because these applications are complex, and many people are involved with many different aspects of the application in parallel. All these changes must be continuously integrated and tested, so an important requirement is that a common source code management system enables a consistent baselining, branching, and retrieval of old versions in a consistent way. For a hybrid application, this technology must work on all participating platforms and in all development and automation environments. Using a special solution for z/OS comes with significant risk of breaking integrations and impeding parallel development.

One technology that addresses all these requirements and runs on z/OS is Git. For more information about Git, see Git.

For more information about open-source languages and tools for z/OS, see Rocket Open Source Solutions for z/OS.

For more information from the z/OS Open Source Tools community, see z/OS Open Tools Docs.

Surveys, such as from the Stack Overflow website, 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 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, and a centralized management server such as GitLab or GitHub that can be used for build orchestration.
- Store all your source code that is 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 one. For more information about the Git branching model and branch-based workflows, see the following websites:
  - A successful Git branching model
  - The essence of branch-based workflows

### 5.1.3 Incrementally building 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 your experience, skills, and careful planning and adjusting plans continuously. The end goal is to build and deploy an entire hybrid application end to end that is based on any change in any of its components, with the optimization to rebuild only the changed parts.

There can be many strategies to achieve these 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 is that the different components might have evolved out of siloed teams that chose different technologies to accomplish their goals, which might be hard to integrate. A top-down approach assumes that some plan or overall strategy is defined, for example, as part of an enterprise modernization initiative. The downside here is that all automation rarely can be reinvented from scratch, so in most cases, a mix between a bottom-up and top-down implementation is the result in most modernization projects.

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

- CICS transaction programs that are written in COBOL that must be compiled on z/OS by using a build toolkit such as IBM Dependency Based Build (IBM DBB); unit tested with IBM z/OS Automated Unit Testing Framework (zUnit) programs that run on the IBM Virtual Test Platform (IBM VTP); and deployed with scripts that are written for IBM DBB in Groovy or IBM UrbanCode® Deploy (IBM UCD).
- Java programs running on z/OS, such as an application back end, can be compiled anywhere by using Java automation frameworks such as Gradle or Maven, but if they use z/OS capabilities such as MVS, they must be tested on the platform, perhaps by using a testing framework such as Galasa.
- A web front end might be written in Node.js with TypeScript and built by using a package manager such as NPM or Yarn that runs unit tests and packages the application.

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

Realizing such an integrated pipeline requires top-down thinking and the selection of a pipeline orchestration technology that can perform all the various steps. In this chapter, we look at a few examples of such technologies. There are other publications that are available that explore such technologies in much more detail, such as the following websites:

- Develop Mainframe Software with Open-source 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 must 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 must be performed at many levels of the application, ranging from unit tests that test the functions 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 that is 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 must 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 that are collected through user stories. Developers run and rerun these tests while creating the implementation until they pass. The results are a formal specification of the requirements, but the downside is that you must maintain many tests, which slow down build times with potential redundancies with tests that are not useful at catching regressions. So, unit tests must be carefully curated and augmented with integration tests.

For impact analysis, IBM has an application suite that is 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 that are 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, and many more. With these tools, you can rapidly analyze an existing application and trace control or data flows to assess the impact that a potential change will have end to end. With this information, you can define the tests that you must create to ensure the correctness of the application before and after the changes.

For test writing, IBM has a framework that is called zUnit (originally based on the xUnit framework, but heavily modified since then) that is composed of a method for test writing and execution, and 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 run in the same environments. In combination with the IBM Z Virtual Test Platform, these tests can run as unit tests, and 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 also can be used for full integration tests running against a full infrastructure setup.

After the tests are written, you must be able to run automatically them as part of a build pipeline and on a component level, which is described in 5.1.5, “Every change that is pushed to the source code management system is automatically built and tested” on page 64.

## 5.1.5 Every change that is 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 run the automation. The short answer is *as often as possible*. Ideally, the full build pipeline should run when 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 *Enterprise Bug Busting: From Testing through CI/CD to Deliver Business Results*, by Radcliffe, the builds must 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 VTP in the Enterprise Application world. Then, development teams can run fast CI builds for their local components, rapidly reviewing, fixing (in the case of regression causing builds to break), and merging code changes, which then can be built in larger integration builds. Such builds can be nightly builds that then run the entire test suites, including CI/CD integration tests on real staging systems. This approach requires more work on maintaining tests for these different stages of builds.

## 5.1.6 Clearly defining your builds as a consistent set of artifacts

The result of an automated pipeline build should be a set of artifacts that is made up of the deployable application with all its installation and configuration components that also allows a full trace back to all the sources that were used for the build. The sources also include the log files and build artifacts that are produced with the build. Some examples of such artifacts are as follows:

- Log files for building individual application components and integration build activities.
- Dependency information linking specific component builds, for example, when not all components require a rebuild, then there must be a 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, and test results.
- Versioning baselines from the software configuration management (SCM) that define the version that is used for each file that is used in the build. For Git, this process can be as simple as using 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 committed IDs, some teams use Git Tags as a way to have more expressive baseline names and label special builds, such as end-of-iteration builds or release builds.
- Application binary files that are used and produced by the build.

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

- Branch off, for example, reproduce the release build three released versions back to fix a bug in an earlier version of the application that might be deployed at a specific site or customer.
- Compare any earlier build with another build to answer questions such as why did the application work back then, and why is it now broken? What changed since then? When was the problem introduced, and did the log files report anything unusual then?
- Perform a trend analysis of quality metrics that are captured by the builds. Restore the last release build, run a static code analysis against it, and compare it against the current state of the code.

Git provides all the capabilities that are needed for baselining source code. Moreover, there are many tools that integrate with Git for creating baselines, such as SonarQube. It can store many of the quality data types that are 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 that store these results for every build and branch.

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

For more information about 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 were performed for the change. These evidence lockers are often implemented as a set of versioned reports that are stored as baselined Git artifacts, and databases or general storage that maintains binary artifacts.

Finally, you must 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 the JFrog Artifactory product that you find referenced by many IBM DevOps offerings. This solution is attractive for hybrid development teams because it supports many artifact types from many different technologies that can be accessed in a technology native way. For example, it provides NPM repositories for storing JavaScript or TypeScript packages that the application might use for its front end, Maven repositories that can be used for Java libraries with Maven or Gradle, and generic repositories that can 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 Artifactory that uses IBM UrbanCode Deploy is shown at z/OS considerations for IBM UrbanCode Deploy.

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

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

IBM Wazi was chosen as a brand name for IBM products that embody the IBM Z DevOps vision that is outlined in this chapter. IBM 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.

IBM Wazi started out as a project to provide COBOL editing capabilities in a browser. 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 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 that is based on the Red Hat CodeReady Workspaces environment. A debugger also was added, and User Build capabilities that use the IBM DBB product.

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

We envision the hybrid development team as a group of people without major skill gaps or skill gaps that can easily be bridged if required, and that work with similar development tools and a common pipeline for hybrid applications that are made up of components running on z/OS and distributed or 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 another node in your pipeline, for example, integrating z/OS 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:

1. z/OS has been working to remove the differences that serve no purpose. z/OS has a set of qualities of service that provide value and allow you to build applications that leverage the reliable hardware, but that does not mean that the development and operations processes must 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 toward 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 IBM Z DevOps pipeline and development tools, let us review the audience, that is, the personas representing developers of a hybrid development project to guide our decision making, and the assumptions that we had about these personas’ provisioning and using our tools.

By using an IBM 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, Todd, and Zach.

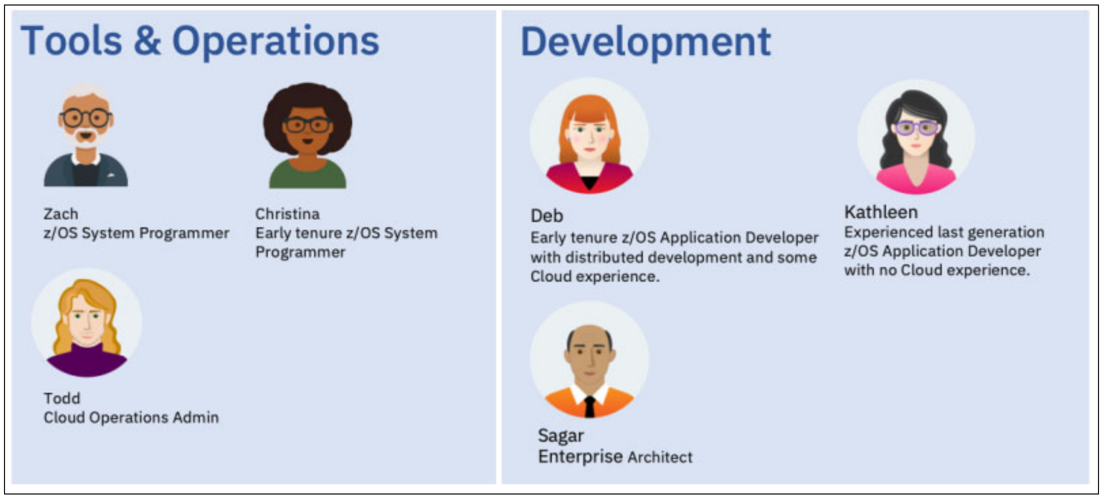

Tools & Operations
Zach
z/OS System Programmer
Christina
Early tenure z/OS System Programmer
Todd
Cloud Operations Admin
Development
Deb
Early tenure z/OS Application Developer with distributed development and some Cloud experience.
Kathleen
Experienced last generation z/OS Application Developer with no Cloud experience.
Sagar
Enterprise Architect

*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) that are used for traditional mainframe application development. Instead, she learned to develop on a modern IDE. Although her instructor in school might have suggested a specific IDE to use for her classes, Deb also tried out other IDEs, and she is used to being able to pick her IDE of choice for programming assignments.

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

## Kathleen's background

Kathleen is an experienced z/OS application developer. Although she spent most of her career coding on the traditional 3270 interface by 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 to continue learning and improving as a developer, and 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 IBM Z DevOps and its associated tools for enriching her team's development process with more 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.

Because developers that are 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 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 knowledge 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 Red Hat 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 that were written with various technologies, such as Java and Node.js. He also has a background in writing automation for Kubernetes by using Helm and Operators that are 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 primarily has been as a traditional z/OS developer that uses “green screen” only. Although he believes in z/OS as the best and most robust platform for security, reliability, and availability, he also feels the need to modernize to adapt and remain relevant in today's fast-paced technical industry. Thus, one of his goals is to automate processes that are 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 try to be as independent from Zach's services as they can. To help them, Zach creates standard z/OS development configurations that Deb's team can deploy themselves as VSIs and use with minimal setup.

### Assumptions for the hybrid cloud-native development team

For this chapter, we assume that there are development teams that are composed of representatives of the three main personas, with limited involvement from Zach, working on a hybrid application that matches one or more of the architectural patterns that are described in this book. The application under development is composed of code in several languages, such as COBOL code running on z/OS, access through REST application programming interfaces (APIs) from Red Hat OpenShift applications that implement their back end (perhaps in a language such as Java), and front-end applications (perhaps in TypeScript and Node.js).

The team decided that all developers should have full visibility into all the components of the application because they want to work as one development squad that uses a common agile, rapid-iteration development process for working on all application components. Although there are clear experts and owners of specific application components that match their backgrounds, all team members should be able to understand how the application works in general, and 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 Red Hat OpenShift.

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

In 5.4, "A next-generation developer end-to-end development example" on page 88, we pick up this scenario again, but remember these personas and their backgrounds when we walk you through the technical IBM 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 its IBM Wazi components. As the name implies with the word *stack*, it is a layered set of capabilities that provide the following functions:

- Give developers and their organizations options by providing layers with alternative implementations that use alternative technologies that best match their requirements or reuse the technologies that are deployed and in use. For example, you can use z/OS Management Facility (z/OSMF) or 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 that users want to implement in their organization or want to implement incrementally. Examples for such layers might be command-line interface (CLI) operations to access only z/OS resources from your development machine; use a graphical navigator in a local VS Code or Eclipse editor; or access z/OS resources from the cloud.
- Support different ways of working based on organizational policies that either centralize development resource management or empower developers to provision their own development environments. For example, developers can use only the z/OS system that is provided to them by their system programmers, but developers can deploy and configure their own z/OS systems as virtualized sandboxes that are deployed through Red Hat OpenShift, which serves as a central command center with automation for provisioning and configuring such systems. Another example is development teams that implement full end-to-end developer responsibilities by managing all build scripts with the source code, which is maintained by the developers versus all development scripts that 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 more sophisticated and complex capabilities. Each layer also is representative of a core architectural principle that we established for IBM 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. Because IBM no longer produces physical 3270 terminals, access through a “green screen” must 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 that they access a Linux server running a Java back-end server, we want to use protocols that allow such similar behavior. There are several to choose from, among which are the following ones:

- FTP: This protocol is focused on getting files on and off z/OS. Through extension, the z/OS FTP server supports access to subsystems such as z/OS UNIX System Services, MVS, and Job Entry Subsystem (JES).
- SSH: z/OS has an SSH implementation that is based on OpenSSH that provides encrypted login and command run facilities, and file transfer access to z/OS, which effectively provides a more secure FTP implementation. Plus, it provides login shells for interactive command runs and for running scripts.
- z/OSMF: 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, running of commands and jobs, and querying status such as for JES. All REST services are fully documented and are discoverable with Open API or Swagger interfaces.
- Remote Systems Explorer (RSE) REST API (RSE API): Provides capabilities like z/OSMF that were introduced with IBM z/OS Explorer and IBM Developer for z/OS (IDz) development tools. Although z/OSMF focuses on systems management, these services are optimized for development activities and maintained by the same group that develops the IBM Z DevOps development tools. The REST APIs are also discoverable with Open API or Swagger, plus there are Node.js and Java libraries that are available to use from development tools implementing these languages.

Development teams can use any of these technologies or combinations of them to access z/OS directly, for example, by using SSH or SFTP to download and upload program files for maintenance updates, and provide connectivity to z/OS for development environments. Teams can create their own tools that use these well-documented capabilities or use the tool sets that were built by them.

## 5.3.2 Layer 2: Building a foundational layer with client software development kits, open APIs, and command-line interfaces

Instead of asking developers to use the connectivity services directly, IBM and IBM Business Partners have invested in extra 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 straight-forward:

- Provide an API for every new capability.

  z/OSMF provides a proprietary web user interface and a REST API that is fully documented for every new capability.

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

  There is also a machine-readable specification that is provided in the industry-standard OpenAPI that is accessible through a Swagger interface.

- Create software development kits (SDKs) to use 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 use it. SDKs are available for Node, Java, Python, and several other programming languages.

- Implement alternative user experiences (UXs) with these SDKs.

  The Zowe SDKs are used for creating several UXs for interacting with z/OS and MVS Data Sets. The most well known of these UXs is the Zowe CLI, which provides a CLI that allows users to create a data set with a simple command, such as **`zowe zos-files create data-set NEW.DATASET --like EXISTING.DATASET`**, and Zowe Explorer, which provides a GUI in Microsoft VS Code editor that allows users to search and display data sets and create data sets with simple right-click operations, such as right-clicking an existing data set and performing an Allocate Like operation similar to the CLI operation.

This abstraction layer for providing SDKs can be combined with the previous layer to provide alternatives to data provider technologies because the SDKs support z/OSMF and other protocols such as FTP. Vendors such as IBM also provide extensions to these SDKs, which are called *plug-ins*, for their own technologies, such as the RSE. The ability to create such plug-ins for extensibility is another core principle that the design of SDKs follows. The Zowe project often makes the extensibility of its capabilities a core feature. For an example of the APIs that are provided by the Zowe Explorer project for contributing new data providers, see VS Code extensions for Zowe Explorer. The RSE API data provider is designed to plug in to Zowe CLI and Zowe Explorer to provide users with the choices for the UXs.

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

We described a flexible architecture for IBM Z DevOps that gives developers and their organizations choices for alternative technologies. We used the example of CLI versus a GUI to show how the same capabilities can be provided as different UXs.

Therefore, retrieving a COBOL program from MVS on a z/OS system, changing it in an editor, and then re-uploading it to z/OS for compiling and running the program can be done in either of these scenarios. Deb the developer (the persona that was introduced in 5.2.2, “Personas of the hybrid development team” on page 67 as a next-generation developer without prior enterprise development experience) can do the following tasks with a CLI 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 as VIM to edit the program.
3. Use another Zowe CLI command to upload the file and convert it back to EBCDIC.
4. Use a third Zowe CLI command to submit a JCL that compiles, links, and tests the programs.

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

1. Use Zowe Explorer's Data Sets view to search for the data set with her COBOL programs.
2. Open the file that was converted by Zowe Explorer to UTF-8 in the IBM Z Open Editor COBOL editor by clicking the data set member name and making changes in the editor.
3. 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 the same way as she found her program files and right-click to submit it, or use the IBM Z Open Editor user build function to upload, build, and run the program by right-clicking in the editor.

Either the CLI editor or a more integrated editor are both 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. One option provides more integrated capabilities than the other one, such as the language support of IBM Z Open Editor performing instant syntax checking and advanced editing capabilities, such as code completion, code formatting, and navigation along an outline view and code references.

However, both scenarios are not mutually exclusive, but rather are specific choices that a developer can make to support their preferred way of working. The tools that are mentioned in these examples can be configured in a way that the scenario can become more interchangeable. For example, VS Code has an integrated terminal and Deb can run any Zowe CLI command directly from VS Code. If the COBOL editing capability was implemented with the right abstractions, then it is feasible that the (almost) exact same UX 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 it is another open API that was created to support interchangeable layers of capabilities and capabilities that can be combined with one another in a plug-and-play 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 LSP, implementations are available for a many programming languages and different and diverse editors, including VIM and 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 20 years as part of its IDz offering, an API such as LSP is ideal because with refactoring, these capabilities can be offered as part of a language server in other editors, such as Microsoft VS Code and Eclipse Theia. However, language capabilities that were newly created for the IBM Z Open Editor for VS Code, such as IBM Z Open Editor's language server for the REXX language, also were immediately adopted and made available for IDz.

In conclusion, when it comes to development organizations providing choices to their developers, a family of editors that use the same language server implementations for specific languages is easier to support and maintain, and it allows developers to switch between editors, even editors that behave differently and offer fundamentally different features.

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

In 5.3.3, "Layer 3: Standardizing on next-generation editors and modern languages capabilities" on page 72, we described the LSP API, which provides pluggable language capabilities to editors. In addition to programming languages support, development teams also need integrations with other development tools and connectivity to platforms, which should be integrated with similar APIs to allow alternatives.

Continuing the example from 5.3.1, "Layer 1: Establishing connectivity to z/OS" on page 71, different development organizations want to standardize on different protocols on that list. If an organization used the Eclipse-based z/OS Explorer or IDz before, then they already deployed the 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 is likely that they want to use the same technology to avoid double-maintenance. Another organization that is not using RSE might prefer using z/OSMF because it is part of the z/OS operation system.

When designing connectivity tools such as Zowe CLI and Zowe Explorer, which are key underlying technologies of the IBM Wazi offerings, there must be support for APIs that support multiple protocols. Zowe CLI provides a CLI plug-in that allows users to install extra capabilities. Such new capabilities can be the implementation of commands that interact with z/OS by using a different protocol. Because Zowe CLI provides z/OSMF that is ready to use, IBM created a plug-in for it that can run all its commands by using the RSE API instead of z/OSMF and SSH.

Zowe CLI and its plug-ins provide CLI operations and can be used as an SDK for other Node.js applications to use the implementation of the CLI operations. The graphical Zowe Explorer that adds visual tree views to VS Code showing hierarchies of MVS data sets, z/OS UNIX System Services folders, and JES jobs was created with these Zowe CLI SDKs. When using the graphical tree view to, for example, list all the data sets for a search pattern, the Zowe Explorer calls the respective API method that is provided by the Zowe CLI SDK.

The RSE API plug-in for the Zowe CLI provides API calls to run the exact same commands against RSE API. It implements the same SDK method for listing data sets for a search pattern. Zowe Explorer was designed to extend its tree views to use alternative plug-ins by using a simple Adapter Design Pattern[1]. So, the combination of APIs from Zowe Explorer and Zowe CLI allowed a 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 by using the Zowe CLI plug-in for FTP with the Zowe Explorer extensibility APIs.

[1] *Design Patterns: Elements of Reusable Object-Oriented Software*, by Gamma, et al.

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

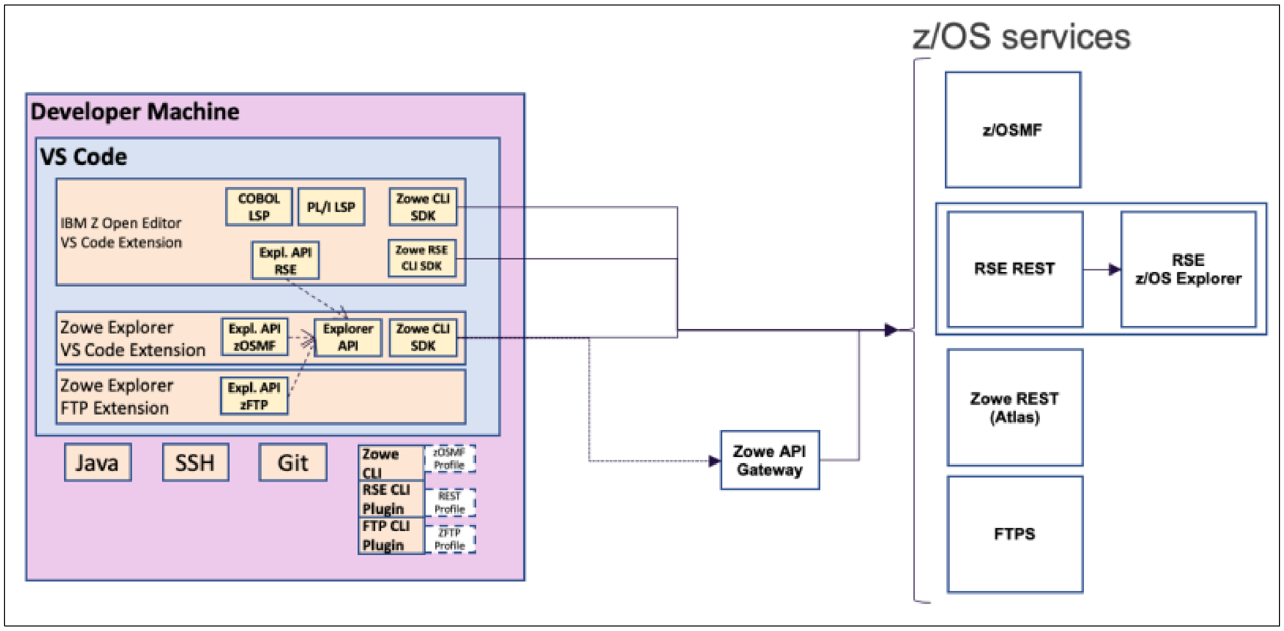


*Figure 5-2 An overview of various extensibility use cases for IBM Z Open Editor and Zowe Explorer*

On the left side of Figure 5-2, you can see three extensions for VS Code: IBM 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 extra capabilities for the Zowe Explorer. IBM Z Open Editor does that task by providing an implementation for the RSE API that uses the RSE API CLI SDK.

The RSE API CLI SDK is packaged with IBM Z Open Editor and fully reused from the Zowe CLI plug-in for RSE API. By using it, Zowe Explorer with RSE API capabilities is independent from the Zowe CLI plug-in that is installed locally. Figure 5-2 illustrates that scenario by showing various CLI plug-ins at the bottom that are installed directly 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 with these extending VS Code extensions, Zowe Explorer can 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 that are used in the project. Here is a selection of extensions that we recommend for an IBM Wazi setup.

- IBM Z Open Debug: Provides a Debugger UI in VS Code for COBOL, PL/I, and high-level assembler language. It is part of the IBM Z Cloud and Modernization Stack.
- Another open-source extension that uses the extensibility APIs of the Zowe Explorer is the Zowe Explorer for IBM CICS VS Code 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 has many features, which almost replace the need for a CLI.

- IBM Z Open Editor uses YAML for configuration files. Red Hat has a YAML extension that reads the JSON and YAML schema and provides code completion and validation.
- If you are doing automation with Ansible and IBM Red Hat Ansible Certified Content for IBM Z, which we use later in this chapter, then you want the Red Hat Ansible VS Code extension, which provides code completion and validation.

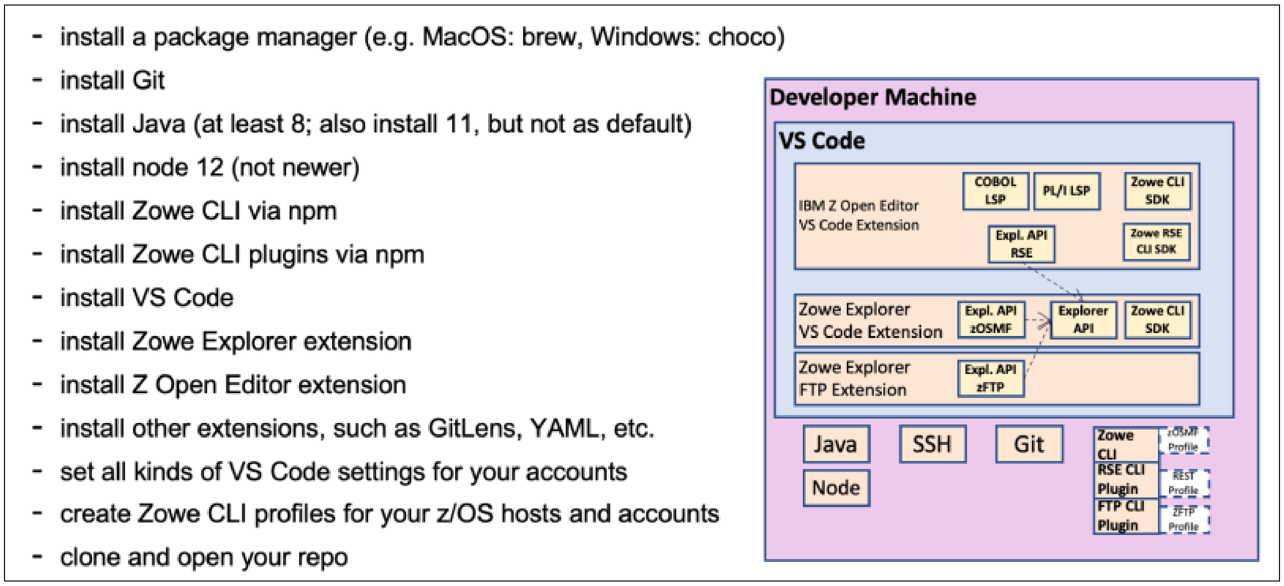
### 5.3.5 Layer 5: Adopting containerization for deploying development tools with Red Hat OpenShift and Dev Spaces

In this chapter, we showed examples of many technologies that can be used 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 are more tools from which the teams can choose.

The IBM Wazi tool set is centered around VS Code so that developers do not have to change tools for different technology combinations. We want to support the most commonly used environments, with the most flexible extensibility, and the largest portfolio of extensions that is 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 is reimplemented even by other editors, including the many implementations of the LSP. The VS Code Marketplace provides tens of thousands of extensions that were 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. At the time of writing, The IBM Z Open Editor was installed 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 that for hybrid development teams that are staffed with personas like Deb, VS Code is the ideal editor for IBM Wazi because 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 because she worked on a traditional green screen emulator and perhaps a more integrated IDE such as Eclipse and IDz that come prepackaged with more capabilities.

Figure 5-3 on page 77 lists some of the steps that Kathleen must perform to set 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 to set up IBM Z Open Editor and Zowe Explorer*

Many development teams manage a wiki (or online web page) that documents all the setup steps that are required for a developer joining a project. For a developer that is experienced with these tools, such as Deb, they are straight-forward. However, even for her there are potential points of errors. For example, if a version of a language run time such as Node.js or Java is required, it is possible that the wrong version is being used, particularly for developers that work on more than one project, which can lead to an accidental usage of incompatible SDKs, build errors, and other items.

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 them on the client and host. When you use Ansible on z/OS, there are more potential incompatibilities to consider with specific versions of IBM Z Open Automation Utilities (ZOAU) 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. If some team members choose to use a different VS Code "compatible" editor such as Eclipse Theia, they cannot use the latest of each version that is available because the Theia implementation of the VS Code API is not up to date. For Windows users, the setup is even more complicated because Ansible does not support Windows as a client (for example, the control node), and users must install a Linux virtualization such as Windows Subsystem for Linux (WSL).

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

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

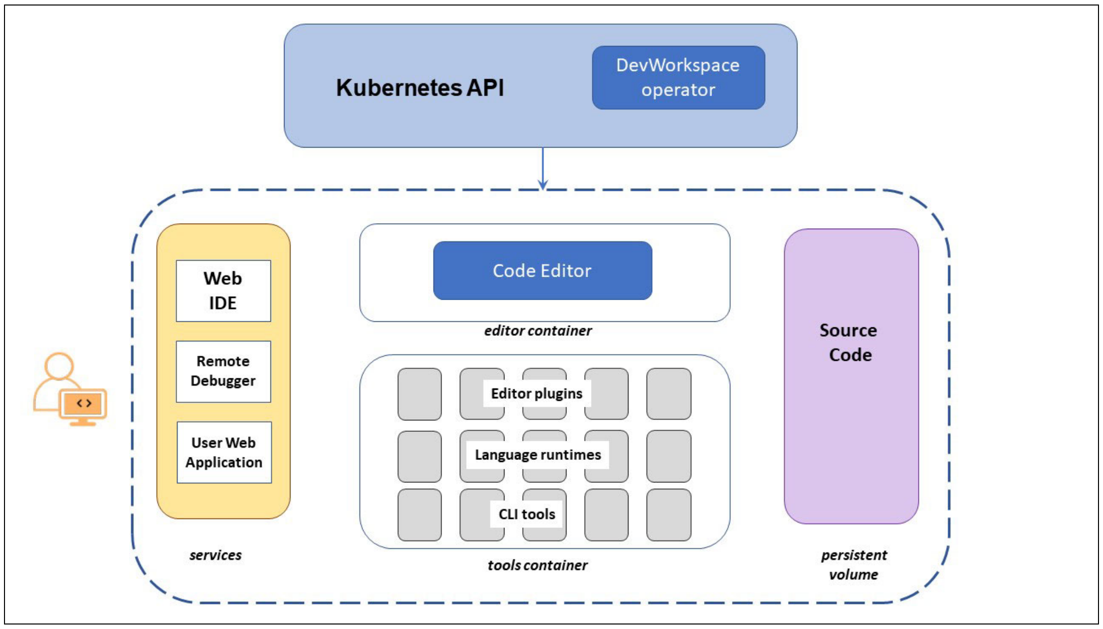


*Figure 5-4 Overview of a developer's personal Eclipse Che workspace*

In Eclipse Che, development teams can choose from a large stack of technology configurations that are provided as container images that are based on open-source Docker files and an open specification framework that is called *devfiles* that you can use to specify development stacks as combinations of multiple images, VS Code extensions, storage volumes, settings, environment variables, command short cuts, Git repositories, and others. A developer then uses a web-based dashboard to select a predefined stack from Eclipse 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 are instantiated into containers, and VS Code extensions are loaded in to a web-based editor. The developer does not need to install anything themselves because everything can be predefined. The stacks that are provided by the Eclipse 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 Eclipse Che system. Then, every other developer can instantiate the stack as their personal workspace with a single click. All the prerequisites and VS Code extensions are 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.

Because 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 that define the container images can be managed by the team and republished to an image registry that is 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 as needed.

Eclipse Che provides a large set of images and devfiles as open source items that are donated by many different developers and organizations. Red Hat took a major subset of them and curated them in a similar fashion as they curate their Linux distributions. With an emphasis on stability, reliability, and security, Red Hat maintains each image by basing them on the Red Hat Universal Base Image and certifying each image. Red Hat published this variant of Eclipse Che as a branded distribution that is called Red Hat OpenShift Dev Spaces, formerly known as Red Hat CodeReady Workspaces, which 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 must license Red Hat OpenShift and the Universal Base Images.

For enterprise and hybrid application development support, IBM took the Red Hat distribution of Red Hat OpenShift Dev Spaces and customized it by using the IBM Wazi for Dev Spaces offering. IBM added capabilities and integrations for z/OS development by providing an extra set of custom images and devfiles on top of what Red Hat provided. The images are fully IBM Cloud certified, which adds extra requirements to the Red Hat certification.

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

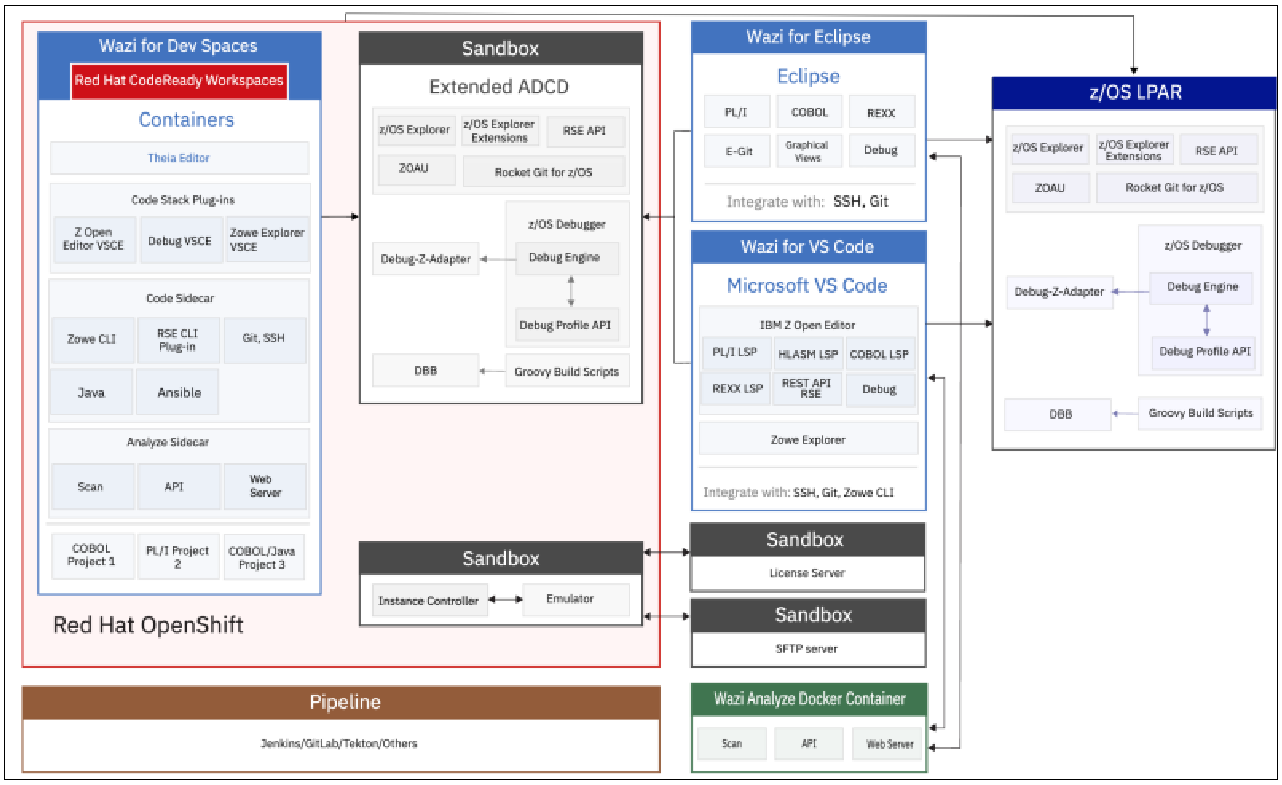


*Figure 5-5 IBM Wazi for Dev Spaces and its integrations*

IBM 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 Plug-in for Zowe CLI that were described in 5.3.2, “Layer 2: Building a foundational layer with client software development kits, open APIs, and command-line interfaces” on page 72.

Together with the Zowe Explorer VS Code extension that is listed in the Plug-ins box in Figure 5-5 on page 79, these capabilities provide tools to access z/OS systems from the editor. Another important set of productivity tools that is pre-configured in the IBM Wazi stack are tools for Ansible and Ansible collections for z/OS, which are installed on that image with the Ansible run time and its prerequisites, such as Python, and editing tools such as the Ansible Linter and the VS Code extension for Ansible. Developers can immediately start working on Ansible playbooks in the editor with 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.

Another sidecar that is available for IBM Wazi for Dev Spaces is the Analyze Sidecar, which provides static code analysis of your applications that are written in COBOL, PL/I, assembler language, JCL, or Java. You can run an analysis scan of the code that you are editing and review the results in a web-based, graph-browser presentation directly from the IBM Wazi for Dev Spaces editor.

As shown in Figure 5-5 on page 79, IBM Wazi provides alternative choices. Although the IBM 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 IBM Wazi package. IBM Wazi comes with the IBM Wazi Sandbox, which provides developers with their own personalized z/OS environment that is emulated on x86 hardware and deployed and managed from Red Hat OpenShift.

**Note**: If you want to try the IBM Z Open Editor running in Red Hat Dev Spaces running VS Code, Red Hat offers a 30-day trial that is named the Developer Sandbox, which can be found at Developer Sandbox for Red Hat OpenShift.

After signing up for the trial, log on and load a workspace in the cloud by running IBM Z Open Editor and Zowe Explorer in VS Code in your browser at Red Hat IDP.

The trial loads a Git repository with some sample COBOL, PL/I, HLASM, and REXX programs that you can use with the editor for testing. For more information, see GitHub.

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

In 5.2, “The vision for a cloud-native developer experience for z/OS enterprise applications” on page 66, we outlined our vision for cloud-native hybrid application development. We described 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 use the same methods, editors, and tools for developing enterprise applications and cloud applications. In Figure 5-5 on page 79, we showed connectivity tools and editors such as IBM Wazi for Dev Spaces that can run in the cloud and deployed into an Red Hat OpenShift namespace.

In this section, we describe what else we can move to the cloud to empower Deb and Kathleen to work in virtual environments that can be rapidly created and re-created and run close to 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, and other automation.

We start 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 these offerings. They come with a preconfigured z/OS image that has been curated for IBM Z DevOps scenarios, which provides many of the IBM software components that are needed, such as compilers for all enterprise languages and modern languages such as Java or Python, and connectivity servers such as RSE API and z/OSMF, Debug, IBM DBB, and 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 by using the IBM Wazi Image Builder. Coming back to our personas from Figure 5-1 on page 68, the system programmer Zach assembles the image for the team and then works 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 by using x86 hardware, but it can be deployed in the cloud in what is known as VPC environments. Todd prepares the environment on Red Hat OpenShift so that Deb and Kathleen can easily create personal instances by using the Red Hat OpenShift developer user interface by pasting and editing YAML files. The v instances can be used as the development platform with other Modernization Stack components, such as IBM Wazi for Dev Spaces, z/OS Cloud Broker, and z/OS Connect.
- IBM Virtual Dev and Test for z/OS (ZVDT) is similar to the IBM Wazi Sandbox with the difference that it is targeted to all IBM Z development shops. The VSIs run on S390x hardware running Linux on IBM Z. The z/OS system is emulated on the real target hardware, which offers significant performance advantages. ZVDT can eliminate typical development bottlenecks by enabling flexible, horizontal scaling of early development, test, and education activities.
- IBM WaaS is a pure cloud-based offering that also runs on real IBM Z hardware in IBM Cloud data centers in many regions throughout the world. Developers such as Deb and Kathleen can self-provision z/OS VSIs into a secure virtual network that is managed by Todd in a VPC. As with the other two solutions, they can create instances by using the IBM default development and test image or use a custom image that is created by Zach and uploaded by Todd into secure IBM Cloud Object Storage. By using the regular cloud interfaces and automation utilities, they can add extra volumes for test databases and other requirements.

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

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

When we described the usage scenarios for IBM Sandbox in 5.3.6, "Layer 6: Moving z/OS development into the cloud" on page 80, we briefly mentioned the role of operators in Red Hat OpenShift playing a role for deploying the IBM Wazi Sandbox and developers using the user interfaces in Red Hat OpenShift that are provided by this operator to create IBM Sandbox instances.

*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 that is part of the Kubernetes specification, and they are the preferred method for deploying and managing applications on Red Hat OpenShift.

This approach of using operators in Red Hat OpenShift also is part of a larger strategy for the IBM approach to hybrid cloud-native z/OS development. Assuming that hybrid development teams want to create their hybrid applications for and with Red Hat OpenShift, then you should use it as a general control platform for managing the development tool infrastructure and 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 CLI.

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

Moreover, z/OS Cloud Broker allows development teams to create their own custom operators by using the popular Ansible framework with IBM Ansible collections for z/OS to support direct z/OS interactions. In 5.3.5, “Layer 5: Adopting containerization for deploying development tools with Red Hat OpenShift and Dev Spaces” on page 76, we already described how VS Code and IBM Wazi for Dev Spaces can provide a highly productive development environment for Red Hat Ansible and how such operators empower the hybrid development teams to create and standardize on common z/OS and other systems automation tasks without involving system programmers.

## 5.3.8 Layer 8: Creating end-to-end automation with IBM DBB and 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 these systems is that development teams can quickly spin up and configure such systems themselves and use them for development and testing in any way they need. However, the potential disadvantage is that these systems, unless they were created with the IBM Wazi Image Builder specifically for a project, are generic and in many cases must be configured for the specific application under development. They require automation to configure the subsystems that are used, such as CICS, to build and to deploy the application.

### Automation for configuring z/OS

We covered several technologies, starting with connectivity APIs, that development teams can choose from to configure their freshly deployed IBM Wazi Sandbox or WaaS VSI. They can pick any of them based on the skill set that is available in the team or other preferences.

Here is a summary of the most important ones:

- Zowe CLI and its many plug-ins: In addition to the basic z/OS interactions that Zowe CLI provides, there is a large list of plug-ins that provide extensions for more capabilities, including z/OS subsystems such as CICS, IMS, IBM MQ, and IBM Db2, but also many third-party technologies. The Zowe Conformance Program website currently lists 18 fully Zowe project certified plug-ins that are available.
- ZOAU is part of the IBM Z and Cloud Modernization Stack. It was developed to serve partially as an alternative to JCL, but also to augment the usage 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 that is available for developers to use ZOAU services in Node.js.

  ZOAU is designed as a natural way for programmers that are familiar with Linux and Unix to use the z/OS UNIX System Services environment directly to access traditional MVS resources such as data sets directly without using 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. Where specific language libraries do not exist, ZOAU provides a shared library interface that is written in C that can be invoked by any programming language by using the appropriate C language bindings.[2]
- Red Hat Ansible and Red Hat Ansible z/OS collections are partially built on top of the ZOAU utility, which is a prerequisite for them, but ZOAU targets Red Hat Ansible developers that are 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 also can integrate with and orchestrate existing tools and automation that are provided by the various technologies that are described 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 through custom operators, can provide a higher level of automation than the script-based frameworks that are in this list.
- IBM DBB and Groovy are intended as a pure build automation framework. as described in “User build automation and debug” on page 83. IBM DBB provides a JVM-based API that can be used for configuration tasks. A user-built script that runs 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 a valid alternative because it can be used in combination with any of the frameworks. JCL can upload and run JCL directly from a remote system. Also, the Zowe Explorer can be used by developers to quickly find and run JCL.

## User build automation and debug

Because enterprise application editors such as IBM Z Open Editor or IBM Wazi for Dev Spaces run on remote clients like developer laptops and Red Hat OpenShift clusters running on x86 Linux or Linux on IBM Z, developers need real-time syntax checking that is provided by an editor, they need the capability to build, run, and debug their applications on z/OS.

[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

The IBM Z and Cloud Modernization Stack includes the IBM DBB, which provides build automation capabilities for build pipelines, and build capabilities for developers. You can run builds by using a CLI, by using SSH shells to z/OS UNIX System Services, or by using integrations to run builds for individual programs in the IBM editors (if needed), and save them with a simple right-click. Developers manage build configuration files (defining things such as which compiler to use and where to search for include files) that they can store with their source code in Git, and build scripts that were written in Groovy, which is a language that became popular because of the Gradle build framework, which makes it easy for hybrid development teams to understand and maintain.

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

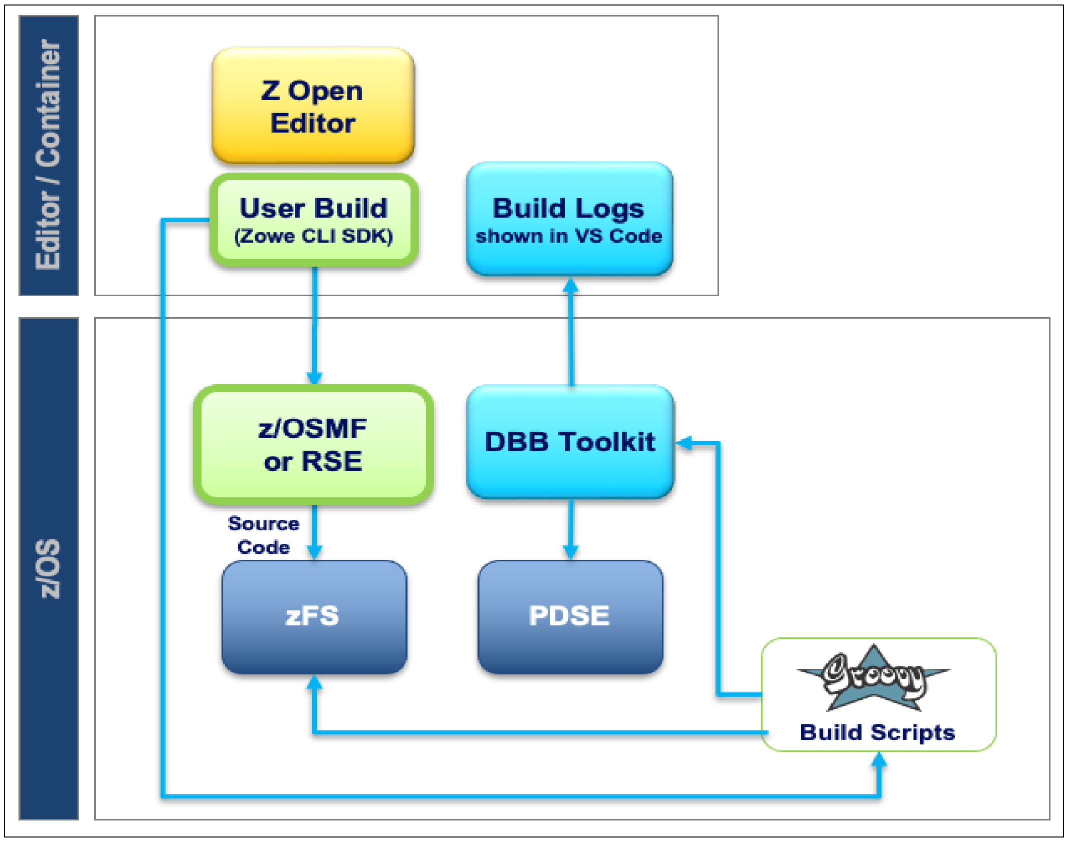


*Figure 5-6 IBM Z Open Editor user build with IBM Dependency Based Build*

Because IBM DBB requires that components are installed on z/OS, the development and test images that are available for IBM Wazi Sandbox, ZVDT, and WaaS are installed and configured so that they can be used directly from the IBM Wazi editors when the Deb and Kathleen 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 editing by right-clicking from the editor. This action uses the editor's language server capabilities to compute the list of local dependencies, and the copybooks in the COBOL cases, and uploads them with the program to zFS UNIX System Services through a Zowe Explorer profile by using the SDK capabilities of Zowe CLI, as described in 5.3.3, “Layer 3: Standardizing on next-generation editors and modern languages capabilities” on page 72.

Whether the user created profiles for z/OSMF or RSE API does not matter because the user build automatically picks the one that is available. On z/OS UNIX System Services, the user build remotely runs IBM DBB by using a Groovy build script. This build script performs all the operations that are required to build the application, create PDS data sets, copy files, run the compile and link commands, and other actions.

Then, the build application can be deployed and started depending on the application. The developer can submit a JCL from Zowe Explorer or use any of the automation capabilities. They can 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 breakpoints, watching variable values, and stepping over and into calls.

### Deployment automation

IBM DBB is used for performing a user build from the editor, and 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 is to deploy the application it is building and run integration tests against it. Deployment automation frameworks such as IBM UCD can be used with IBM DBB.

IBM UCD provides a web-based UX for defining applications through conceptual models that are composed of components, resources, processes, and environments. IBM DBB performing a continuous integration (CI) build, such as for changes that are pushed to a Git server, can publish its build results to an artifact repository, such as Artifactory, and trigger a CD process in the IBM UCD server. Then, IBM UCD orchestrates the deployment of the Artifactory assets to a dedicated z/OS test system by using an agent that is installed there.

For more information and a detailed tutorial showing how to install the z/OS agent with the IBM UCD server to define an application process, set up a build pipeline for IBM DBB in Jenkins, and connect IBM UCD to the Artifactory, see API reference. This website also explores configuring IBM UCD deployment steps for CICS and Db2.

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

In 5.3.8, “Layer 8: Creating end-to-end automation with IBM DBB and Groovy and Ansible collections for z/OS” on page 82, we mentioned that the various automation components can be used in fully automated CI/CD pipelines. There is a 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 these pipeline technologies are cloud-native, that is, they can either run directly in the cloud or access resources and nodes in cloud, or can integrate z/OS as a compute node into the pipeline workflow.

For hybrid development teams, we envision pipelines that 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 describe representative example pipeline technologies here, but there are many more that can be used.

### GitLab CI/CD

GitLab Ultimate is tailored for hybrid IBM Z DevOps scenarios. Together with IBM DBB and a GitLab Runner (an agent, for example) that can run on Linux for IBM Z, it is a package that is tailored for hybrid teams that must build z/OS components. GitLab servers and agents can be deployed in the cloud and onsite and used in a fully distributed manner by 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.

Much like 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 that are stored on a GitLab server. Pipelines can be triggered based on a branch push, run on a schedule, or be manually started. The agent that runs the pipeline scripts can be running on a virtual, cloud-based, or hardware Linux or Linux for IBM Z node, and run build operations by using SSH against a z/OS build machine running IBM DBB. Build results can be pulled by the agents to populate build reports and artifact stores that are provided by GitLab or third parties such as Artifactory.

IBM published several tutorials and documentation for combining GitLab with many of these technologies. For more information, see GitHub.

Here are some important examples:

- Build a pipeline with GitLab CI, IBM DBB, and IBM UCD, which completes the automation that uses IBM DBB with IBM UCD, which is now orchestrated by GitLab CI/CD.
- Integrating IBM z/OS platform in CI/CD pipelines with GitLab, which is a detailed description into configuring GitLab for hybrid applications that are built on multiple platforms, and integrations with other IBM Z DevOps capabilities, such as IBM 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, and frequent release deliveries.

### IBM Cloud Toolchain

Although you can deploy GitLab securely in a VPC in VSIs or Kubernetes clusters, the most common deployment scenarios for GitLab are on-premises with possibly a hybrid mix of local and cloud-based nodes. If you want to run a purely cloud-native pipeline, you can use the IBM Cloud Continuous Delivery offering that offers a toolchain feature or graphical modeling of your entire DevOps configuration from Git repositories to links to cloud-based editors such as IBM Wazi for Dev Spaces to build pipelines, and artifact and build report management and insights tools.

When you create your toolchain by using the graphical editors in IBM Cloud, there is a toolchain template that you can pick that is tailored for building z/OS applications. For more information about detailed walk-through of the wizard that is used by the template and all the options that are available, see Create a toolchain to secure z/OS application development in IBM Cloud.

The resulting pipeline implements many of the practices that we described in 5.1, “Core practices of IBM Z DevOps for hybrid enterprise application development” on page 60, such as 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 were performed for the change. These evidence lockers often are implemented as a set of versioned reports that are stored as baselined Git artifacts and databases or general storage that maintain binary artifacts.

The pipeline runs as a Tekton pipeline that can run build steps against the target system. Developers manage their Tekton build scripts in a dedicated Git repository that is configured for the pipeline. The developers can define custom container images to use for these builds that provide all the tools that are required to run a remote build against z/OS, and then use any of the connectivity solutions 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 to create and run Tekton pipelines. Tekton is an open-source framework for creating Kubernetes native CI/CD systems. It was created for the Kubernetes project. For a hybrid development team that works on a Kubernetes front-end application and z/OS back-end applications, running the integrated pipeline on Kubernetes with Tekton can be an attractive alternative. If you run on a different Kubernetes platform such as Red Hat OpenShift, AWS, or Azure, and develop with on-premises Kubernetes clusters, using Tekton directly can be a valuable alternative.

Red Hat OpenShift is a potential central control center for the entire hybrid development project's development infrastructure that offers through Red Hat OpenShift Pipelines a great, almost platform-independent complement to the tools of the IBM Z and Cloud Modernization Stack. You can define the pipeline with the de facto, 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, Red Hat OpenShift Pipelines can be written as code, but it also integrates with the graphical Red Hat OpenShift UX in the same Developer Perspective as the z/OS Cloud Broker, IBM Wazi Sandbox, and IBM Wazi for Dev Spaces operators, which provide a single space for developers of the hybrid project.

Although Tekton was designed to build and deploy Kubernetes applications, it also can be used as an orchestrator for starting external subpipelines that run on other network nodes and wait for their completion. Tekton can be used to integrate other non Tekton models, such as the ones that are required for IBM DBB running on a z/OS node that builds and deploys 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 by using the Zowe CLI, found at Accessing z/OS from your Tekton Pipeline.

### 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 large and active community. It is easy to find developers with Jenkins skills, documentation and reusable solutions for most use cases, and a large portfolio of extending plug-ins. 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. For more information and a detailed tutorial about how to build a pipeline with Jenkins, IBM DBB, and IBM UCD, see Build a pipeline with Jenkins, Dependency Based Build, and UrbanCode Deploy. This tutorial describes how to set up a pipeline with Jenkins, IBM DBB, and IBM UCD. You can compare it to building a pipeline with GitLab CI, IBM DBB, and IBM UCD, comparing a Jenkins setup with GitLab to understand the differences in the UX, and decide which fits better in your hybrid development project.

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

This section describes some examples for using 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 look closer at Deb, who is our user persona who represents 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 typically follows to implement a change by using the practices and technologies with which she is familiar. Section 5.4.3, “Applying next-generation development strategies and tools to mainframe application development” on page 91 maps these steps in a detailed example of the 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 dedicated for her specific task. By switching to this new feature branch to make the code changes for her task, Deb can work on the task in isolation and in parallel with her team without having to worry about 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: After she makes 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 are correct, Deb commits her code, and then pushes her feature branch to the central Git repository.
7. Merge request and approval process: When 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 set up an automated pipeline to run builds of the code in the merge request, which also include tests and code scans.

   This point is when her team implements 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: When 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 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 continue working in this way with tools, technologies, and workflows that support the best practices that she was taught.

## 5.4.2 Deb's tools

Now, we describe the tools and technologies that Deb uses to achieve her development workflow from beginning to end. Within the CI/CD pipeline, these items can be broken into the following six main components:

1. SCM
2. IDE
3. Build component
4. Artifact repository
5. Deployment manager
6. Pipeline orchestrator

### Software configuration management

The SCM tool is what Deb and her team use to store and manage different versions of their source code files, application configuration files, test cases, and more. With this tool, Deb and her team can do parallel development.

Git is the de facto industry standard in source code version control. Popular Git providers such as GitLab and GitHub 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. The providers 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**: Because the source code for mainframe applications might be managed on z/OS, the following methods can facilitate the migration of a monolithic mainframe application into logically decoupled Git repositories:

- The IBM DBB migration tool.
- The IBM Software Configuration and Library Manager migration tool.
- Manual migration of source files in MVS to Git repositories is a possibility. The files are copied into z/OS UNIX System Services by using ISPF/TSO, which is the z/OS UNIX System Services CLI, or IDz, and then a Git repository is initiated from the destination folder in z/OS UNIX System Services.

IBM provides a no-charge and self-paced online course where you can learn more about the source code migration process and other IBM DBB fundamentals.

### Integrated development environment

Deb uses the IDE to check out and edit her code, check it back in to 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 that provided by their team through a web URL. The image contains the configured IDE and other development tools that are used by the team, which reduces the time and effort that is required to generate instances of the development environment and onboard new teammates.

### Build component

The build component understands and resolves dependencies, and then converts the source code to produce the runnable software artifacts. In this component, Deb and her team can integrate automated steps for unit testing and code quality inspection.

For z/OS application development, IBM 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 build only the changed programs and manage their impacts (similar to how mainframe application teams often manage builds), but IBM 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. IBM 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 wants to install or run during deployment. The package is uploaded and stored in the artifact repository, which also stores metadata to help trace the software artifacts back to the source code from which they originated. This process 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 tracks the inventory of runtime environments so that the team can know what each one is running.

IBM UCD is a popular deployment manager option, but it is 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 the IBM DBB public GitHub repository.

### Pipeline orchestrator

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

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

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

With next-generation tools, 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 that are favored by distributed development teams, which enable standardization of development tools across the enterprise, and can allow z/OS and distributed application teams to collaborate more easily by breaking down the silos between them and opening the way for enabling a multi-technology application architecture. There are many choices for integrating modern development tools and strategies into mainframe development processes. The scenario that is described in this section for Deb's team illustrates an example implementation.

#### CI/CD pipeline setup

Sharing the technologies that are 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 that is hosted by GitLab. GitLab CI/CD integrates the SCM with other pipeline tools such as IBM 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 set up a preconfigured developer workspace in IBM Wazi for Dev Spaces that contains all the necessary source code, z/OS access, and tools that are needed to be productive on the team. Deb and her teammates can access instances of this development environment with a single click after logging in to the team's IBM 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 so that she can work on her development tasks there by using IBM Wazi for VS Code or IBM Wazi for Eclipse. In this case, Deb chooses IBM Wazi for Dev Spaces as the most convenient option because it is set up for her and does not require any special configurations or tools on her local workstation.

#### Preparing the z/OS environment

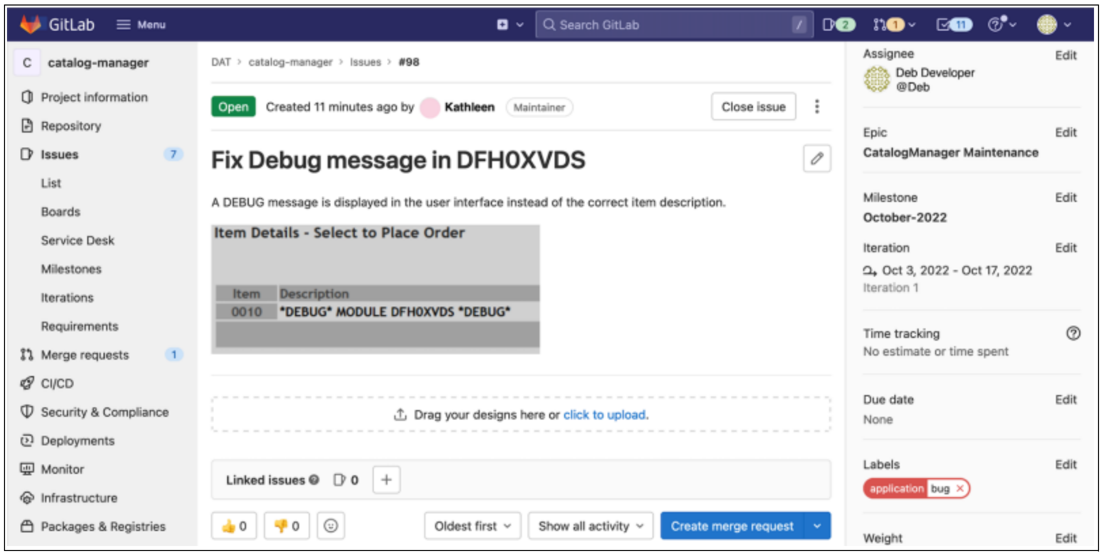
To easily create the z/OS development and test environments for Deb's team to work on, their system programmer, Zach, uses the IBM Wazi Image Builder to extract a custom image from their organization's on-premises IBM Z platform. This image contains the setup and configuration information for the z/OS system, and the organization's cloud administrator, Todd, uses it as the basis to create more development and test environments. Todd may use the IBM Cloud graphical web interface to create the WaaS target environments from the image. Although this graphical approach takes only a few clicks per environment instance, Todd 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, which makes the process more efficient and scalable. When Deb is ready to connect to the z/OS environment, she can self-provision her z/OS VSI on the VPC that is managed by Todd. Section 5.5, “IBM Wazi as a Service and IBM Z and Cloud Modernization Stack tutorial” on page 96 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 that 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 bug fixes that are identified as targets for the next milestone. They 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 example, Deb was assigned an issue to fix a display message in one of her team's COBOL programs, similar to Figure 5-7. She can view the details of her assigned issue on its dedicated GitLab issue page, and update her issue's description, status, comments, and other metadata there.



Open Created 11 minutes ago by Kathleen Maintainer

Fix Debug message in DFH0XVDS

A DEBUG message is displayed in the user interface instead of the correct item description.

```
Item Details - Select to Place Order

Item  Description
0010  *DEBUG* MODULE DFH0XVDS *DEBUG*
```

Assignee: Deb Developer @Deb

Epic: CatalogManager Maintenance

Milestone: October-2022

Iteration: Oct 3, 2022 - Oct 17, 2022 Iteration 1

Time tracking: No estimate or time spent

Due date: None

Labels: application bug

Weight

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

2. Get the latest code: Deb logs in to her team's IBM 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.

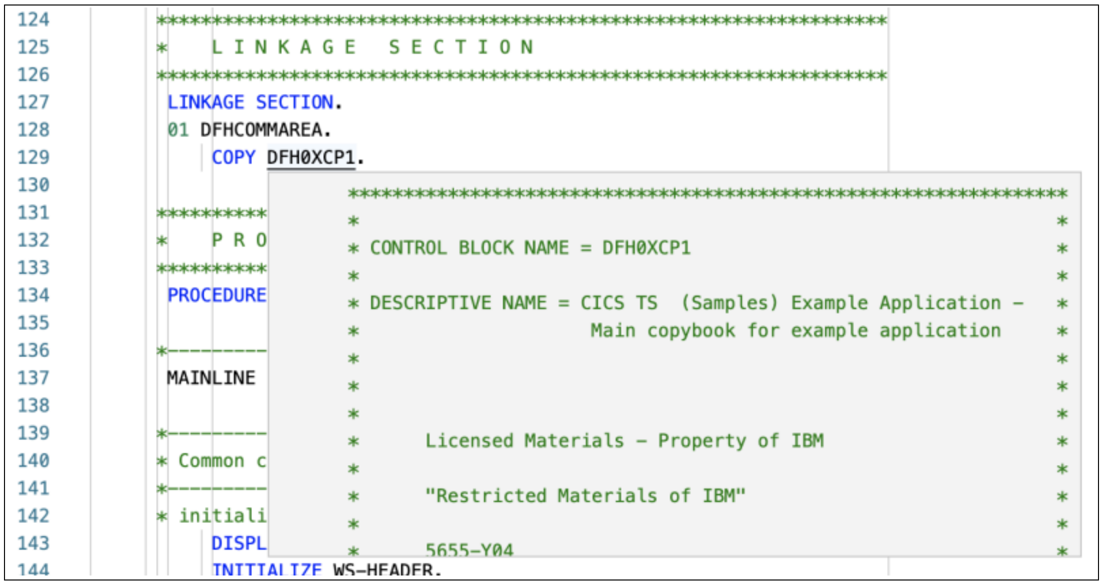
3. Switch to the feature branch: To begin working on her assigned issue, Deb uses the Git integration on her IDE to create a branch of the application code. This new branch is a copy of her team's shared development code, and she ensures that she switches to this new branch before beginning her coding work so that the branch is dedicated to her work on her assigned task.

4. Make the code changes: Using the IBM Z Open Editor and IBM Z Open Debug extensions to support her coding work, Deb makes the code changes to fix her assigned issue. She also can run unit tests that are related to the code she is modifying 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, z/OS UNIX System Services files, and jobs.

The IDE that is she is using provides her with the following capabilities, among others:

– Editing: In IBM Z Open Editor, Deb uses the Outline View to explore and navigate the code for the program that she is fixing. With the copybook resolution feature, she can preview the contents of copybooks that are referenced in the COBOL program by hovering her cursor over the copybook name, and she also can open the copybook itself in a separate editor by pressing Ctrl+click (Windows).

  While Deb is coding her fixes, IBM 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 leverages the hover declaration lookup feature 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 in which she is interested.



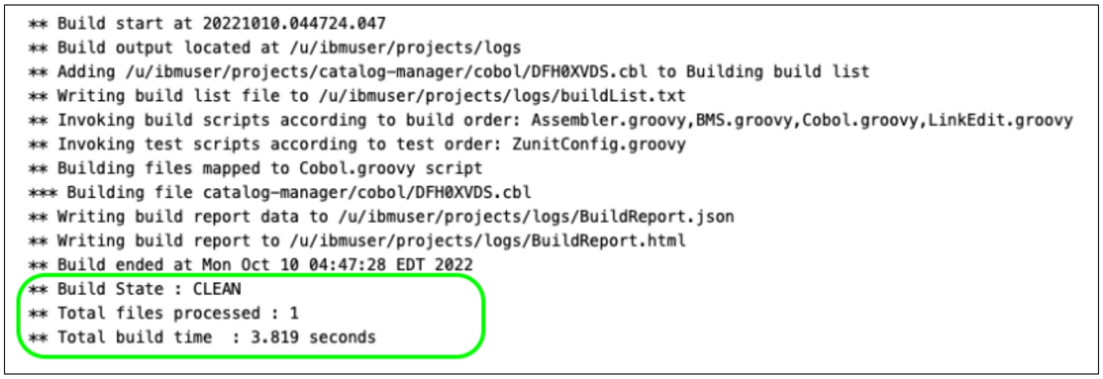
```
      *****************************************************************
      *    L I N K A G E   S E C T I O N
      *****************************************************************
       LINKAGE SECTION.
       01 DFHCOMMAREA.
           COPY DFH0XCP1.

      **********
      *    P R O
      **********
       PROCEDURE

      *---------
       MAINLINE

      *---------
      * Common c
      *---------
      * initiali
           DISPL
           INITIALIZE WS-HEADER.
```

```
      ****************************************************************
      *                                                              *
      * CONTROL BLOCK NAME = DFH0XCP1                                *
      *                                                              *
      * DESCRIPTIVE NAME = CICS TS  (Samples) Example Application -  *
      *                    Main copybook for example application     *
      *                                                              *
      *                                                              *
      *                                                              *
      *     Licensed Materials - Property of IBM                     *
      *                                                              *
      *     "Restricted Materials of IBM"                            *
      *                                                              *
      *     5655-Y04                                                 *
```

*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. By using this function, Deb can monitor the code's runtime path to analyze the root cause of the issue.

5. Run a personal build and test: After Deb makes her code changes that she is ready to test, she uses the IBM DBB 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.



```
** 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 IBM DBB User Build log with build results outlined in green*

Deb can install the generated build artifacts (such as load modules) into her self-provisioned z/OS instance and perform manual testing without worrying 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 extra and efficient way to check for regressions.

Deb and her teammates can use the zUnit within IDz to create *unit tests*, which can be stored as part of the same application repository containing the source code. Deb's team can script the running of these zUnit tests to occur automatically after a build completes.

6. Commit and push code changes: Having coded, built, and tested her fix for the assigned issue, Deb is satisfied 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 set up an automated GitLab CI/CD pipeline to run builds of the code for any changes to merge request code, which also includes 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.

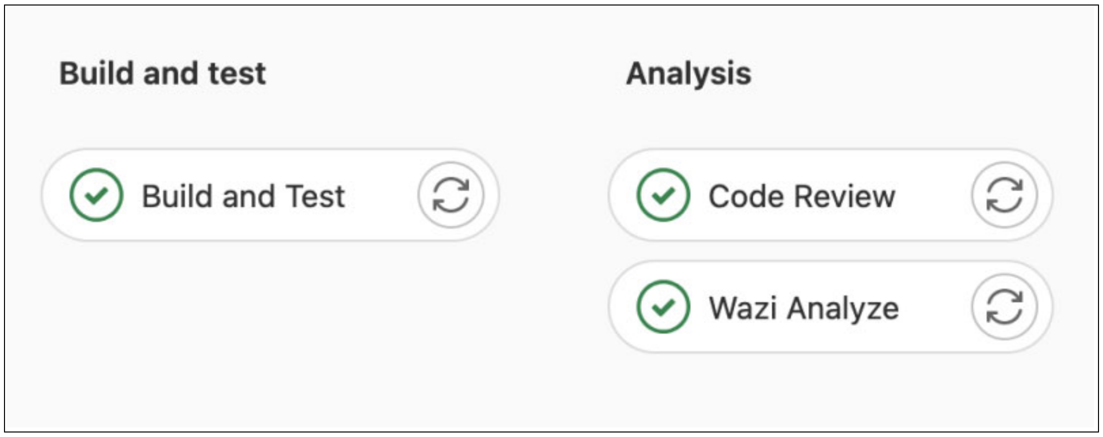


*Figure 5-10 The primary steps in a CI/CD pipeline for a merge request's feature branch*

8. Integrate code changes: After Deb’s request is approved and her fix is merged into the shared development branch, the feature branch that 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) with Deb's fix to move the changes forward or associate them with a release.

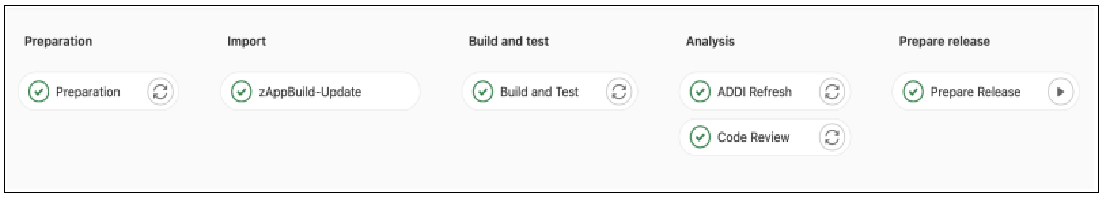


*Figure 5-11 An example of a CI/CD pipeline that can run when a merge request is merged into the team's shared development branch*

After Deb’s code changes are merged into the shared development branch, the artifacts and metadata that are 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 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, and 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 UCD or, as Deb's team has set up, by a deployment script.

Aftere Deb's team achieves their development milestone and their code 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, and create a release on their GitLab SCM. The release package is uploaded to the team's artifact repository, from where it can be downloaded and deployed to the various testing environments and, once ready, also deployed to the production environment.

For more information and guidance on CI/CD pipelines designs that can be implemented at different points in the release development process, see Implementing a Release-based Development Process for Mainframe Applications.

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

The following tutorial describes the steps that you perform to deploy a complete cloud-based development environment. You need a web browser (Firefox or Google Chrome). The system that we deploy is shown in Figure 5-12, which is a variant of Figure 5-5 on page 79 that replaces IBM Wazi Sandbox running in Red Hat OpenShift with WaaS running in the IBM Cloud on real IBM Z hardware as a VSI.

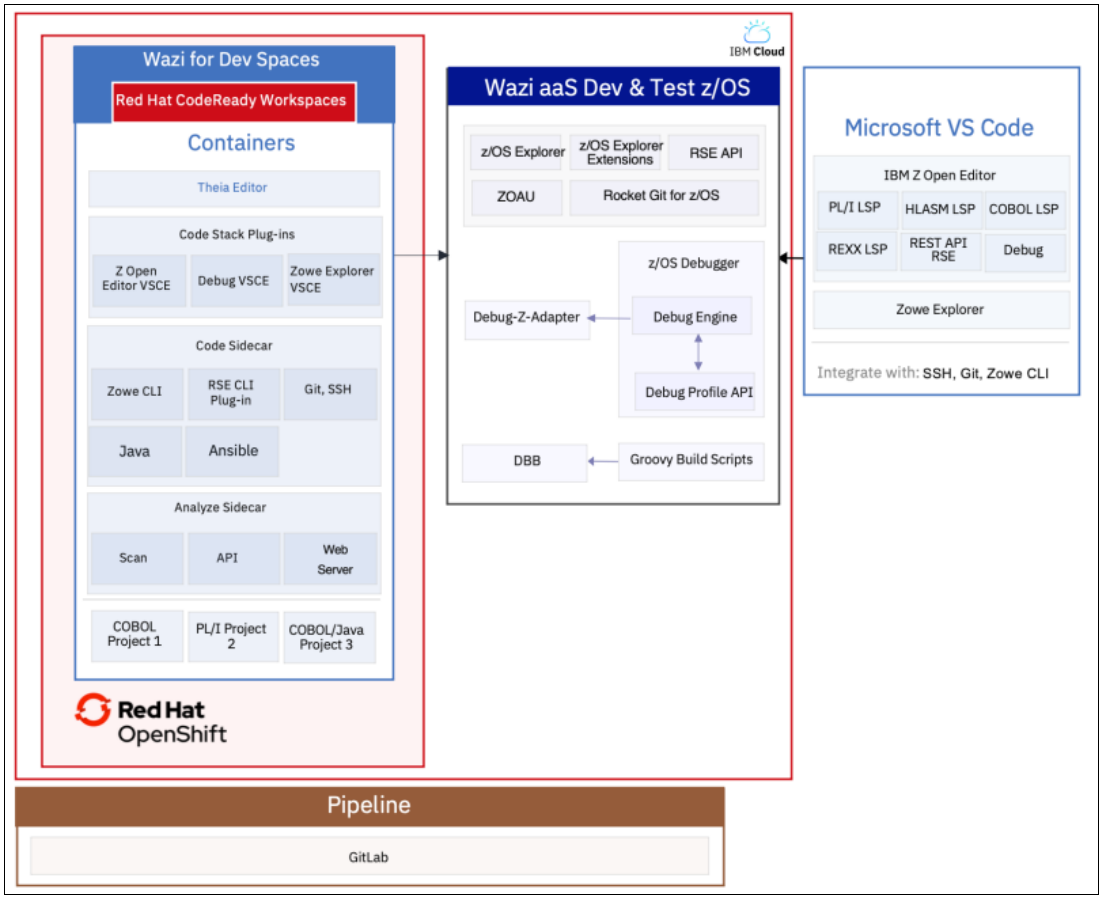


*Figure 5-12 Cloud deployment diagram of the tutorial system*

When the deployment is set up, you can use the cloud-based editor in IBM Wazi for Dev Spaces to edit, build, and debug a sample COBOL program in that environment. There are some optional steps that we do not cover in this tutorial to keep it 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 provide links to the documentation and recorded demo videos that show you how those parts work. With this system, you also can try other components from the IBM Z and Cloud Modernization Stack that we cannot cover here.

There is a fee that is associated with creating these systems. The costs are shown when you create the systems in the IBM Cloud creation dialog boxes. In most cases, the charges are per hour, so if you keep your tutorial walk-through brief and delete your resources afterward, then the charges should be reasonable.

## 5.5.1 Creating a virtual private cloud and z/OS virtual server instance

In this section, we assume the role of Todd, who is tasked with provisioning a cloud-based development infrastructure for the team. His plan is to provision a VPC for the team and deploy everything that they need there. To sketch out his plan and discuss it with the team, he creates the diagram in Figure 5-13 that shows all the components that he plans to deploy and configure.

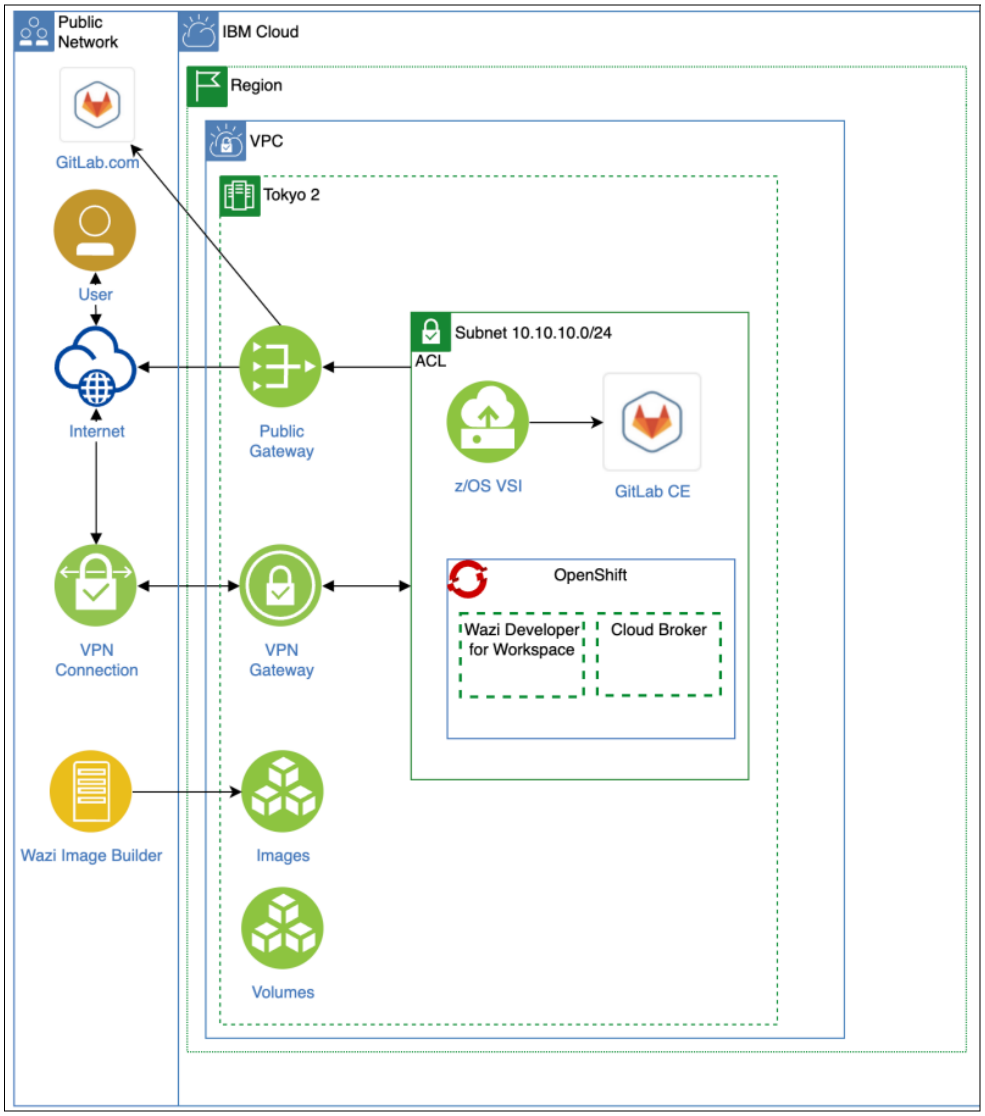
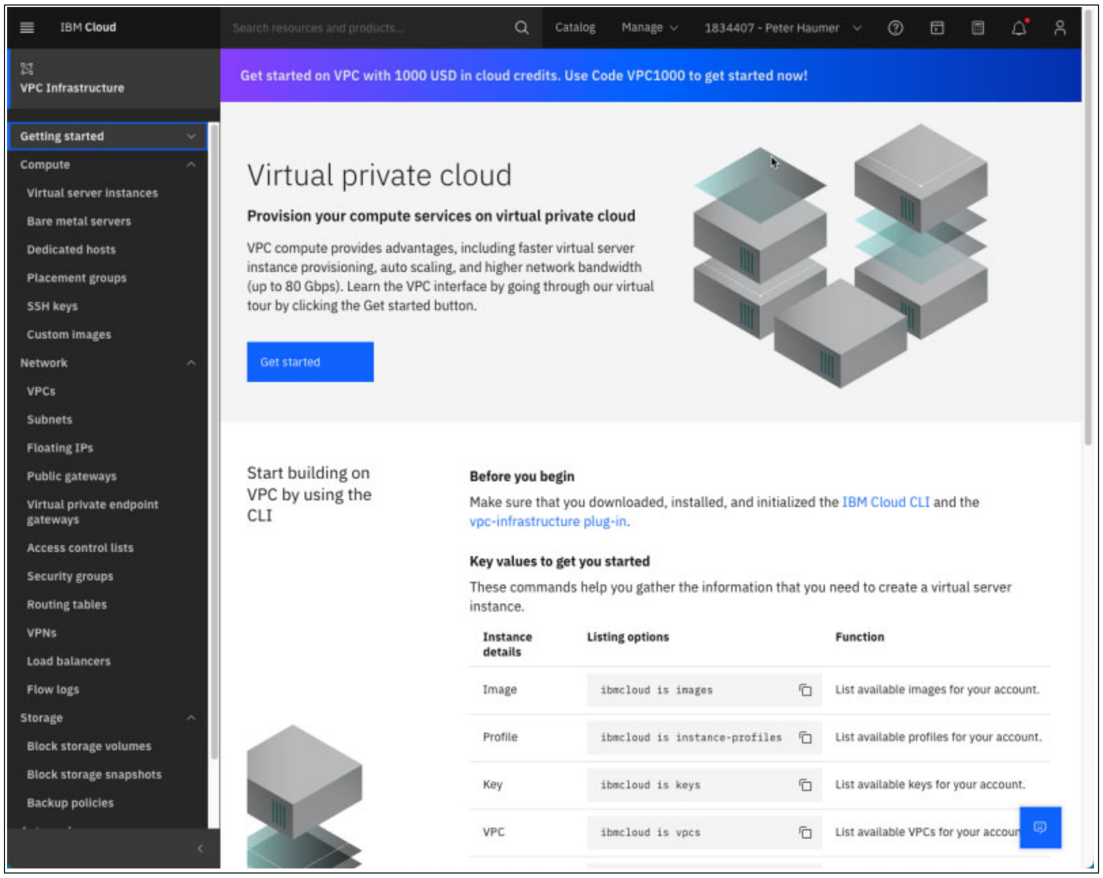


*Figure 5-13 Cloud deployment diagram of the tutorial system*

As an experienced cloud administrator, he deploys this infrastructure entirely with automation scripts that are written in Terraform and Ansible, which are based on reference scripts that are provided by IBM. However, this time he wants to teach his colleagues Deb and Kathleen the principles of this technology by using the IBM Cloud GUI to deploy each component one by one to explain each piece more visually. He performs the following steps:

1. He creates an account in IBM Cloud. He upgrades his new account to a Pay-As-You-Go account. For more information, see Setting up your IBM Cloud account.
2. He creates access groups and adds other cloud administrators to this account and gives them permissions. Deb and Kathleen do not need an account.

3. After he has access, he is ready to create a VPC and a z/OS virtual server instance (VSI). For more information, see Getting started with IBM Virtual Private Cloud (VPC).
4. He shows Deb and Kathleen the IBM Cloud VPC overview window, which in our example is `https://cloud.ibm.com/vpc-ext/overview` and shown in Figure 5-14. This window is the central hub for creating and configuring many of the components that he needs.



Get started on VPC with 1000 USD in cloud credits. Use Code VPC1000 to get started now!

Virtual private cloud

**Provision your compute services on virtual private cloud**

VPC compute provides advantages, including faster virtual server instance provisioning, auto scaling, and higher network bandwidth (up to 80 Gbps). Learn the VPC interface by going through our virtual tour by clicking the Get started button.

Start building on VPC by using the CLI

**Before you begin**

Make sure that you downloaded, installed, and initialized the IBM Cloud CLI and the vpc-infrastructure plug-in.

**Key values to get you started**

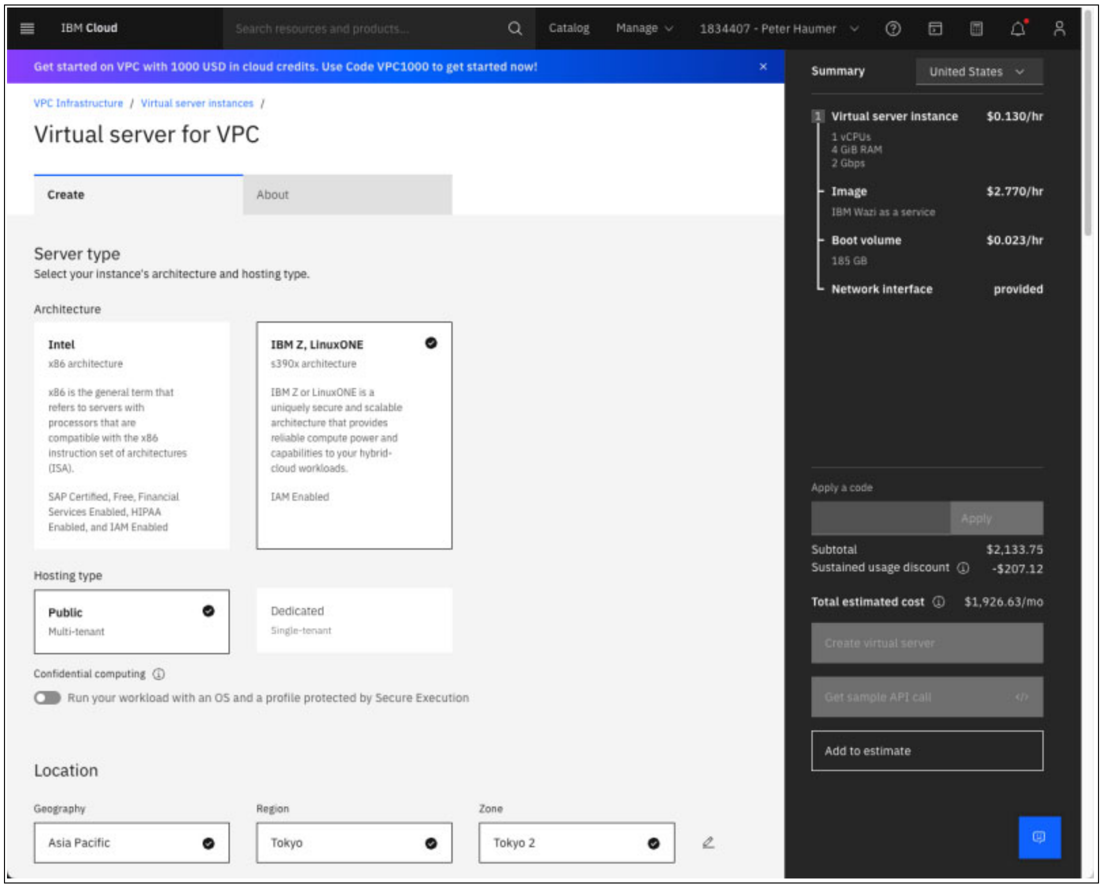
These commands help you gather the information that you need to create a virtual server instance.

| Instance details | Listing options | Function |
|---|---|---|
| Image | `ibmcloud is images` | List available images for your account. |
| Profile | `ibmcloud is instance-profiles` | List available profiles for your account. |
| Key | `ibmcloud is keys` | List available keys for your account. |
| VPC | `ibmcloud is vpcs` | List available VPCs for your accour |

*Figure 5-14 IBM Cloud Virtual Private Cloud*

5. From the table of contents on the left, he browses to various pages to create the resources that are needed, which include the following tasks:
   a. The SSH keys page provides a public key to be used later for connecting to the z/OS VSIs.
   b. He creates a VPC in his preferred region that also hosts z/OS VSIs.
   c. He creates subnets in three zones because he will use the same VPCs and subnets to also run his Red Hat OpenShift cluster later.
   d. He attaches a public gateway because he wants access to the public internet to download IBM Red Hat OpenShift Operators and images, such as IBM Wazi for Dev Spaces.
   e. He configures an access control list (ACL) to limit the subnet's inbound and outbound traffic.
6. He creates a VPN for clients to connect securely to the VPC from his organization. For more information, see Creating a client-to-site VPN server for VPC.

   With the VPC ready and configured, he can deploy his first z/OS VSI (Figure 5-15 on page 99).



Get started on VPC with 1000 USD in cloud credits. Use Code VPC1000 to get started now!

VPC Infrastructure / Virtual server instances /

Virtual server for VPC

Create | About

Server type

Select your instance's architecture and hosting type.

Architecture

**Intel**
x86 architecture
x86 is the general term that refers to servers with processors that are compatible with the x86 instruction set of architectures (ISA).
SAP Certified, Free, Financial Services Enabled, HIPAA Enabled, and IAM Enabled

**IBM Z, LinuxONE** ✓
s390x architecture
IBM Z or LinuxONE is a uniquely secure and scalable architecture that provides reliable compute power and capabilities to your hybrid-cloud workloads.
IAM Enabled

Hosting type

**Public** ✓
Multi-tenant

Dedicated
Single-tenant

Confidential computing
Run your workload with an OS and a profile protected by Secure Execution

Location

| Geography | Region | Zone |
|---|---|---|
| Asia Pacific ✓ | Tokyo ✓ | Tokyo 2 ✓ |

Summary — United States

| | |
|---|---|
| 1 Virtual server instance | $0.130/hr |
| 1 vCPUs<br>4 GiB RAM<br>2 Gbps | |
| Image | $2.770/hr |
| IBM Wazi as a service | |
| Boot volume | $0.023/hr |
| 185 GB | |
| Network interface | provided |

Apply a code

| | |
|---|---|
| Subtotal | $2,133.75 |
| Sustained usage discount | -$207.12 |
| Total estimated cost | $1,926.63/mo |

*Figure 5-15 Creating a virtual server instance for the IBM Z architecture*

7. He performs the following tasks:

   a. In the menu in the left pane, he selects **VPC Infrastructure** → **Virtual server instances**.

   b. He clicks **Create** on the Virtual server instances window, which opens the window that is shown in Figure 5-15, and selects the “IBM Z, LinuxONE” architecture template.

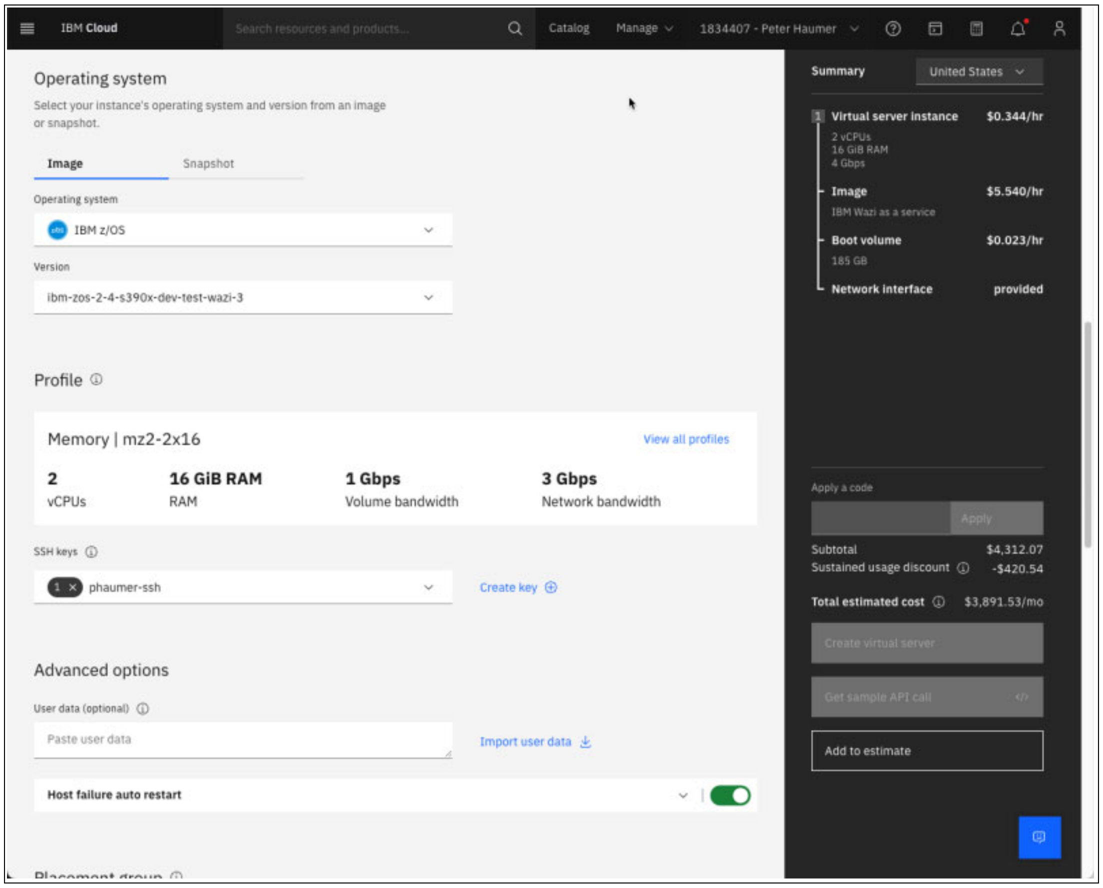
   c. He selects his VPC Geography, which automatically also selects the same Region and Zone (subnets) of the VPC.

   d. Moving down to the Details section, he enters a unique name for his VSI, and then selects the appropriate resource group.

e. He scrolls to the section that is 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`. (The actual number at the end of the image version might vary.)

f. He selects a profile with his vCPU and RAM requirements (`mz2-2x16`), which is a good choice for evaluation.

g. Under "SSH keys", he selects the SSH key that he created earlier to use for connecting to the VSI.



Operating system

Select your instance's operating system and version from an image or snapshot.

Image | Snapshot

Operating system: IBM z/OS

Version: ibm-zos-2-4-s390x-dev-test-wazi-3

Profile

Memory | mz2-2x16 — View all profiles

| 2 | 16 GiB RAM | 1 Gbps | 3 Gbps |
|---|---|---|---|
| vCPUs | RAM | Volume bandwidth | Network bandwidth |

SSH keys: phaumer-ssh — Create key

Advanced options

User data (optional): Paste user data — Import user data

Host failure auto restart

Placement group

Summary — United States

| | |
|---|---|
| Virtual server instance (2 vCPUs, 16 GiB RAM, 4 Gbps) | $0.344/hr |
| Image (IBM Wazi as a service) | $5.540/hr |
| Boot volume (185 GB) | $0.023/hr |
| Network interface | provided |

Apply a code — Apply

| | |
|---|---|
| Subtotal | $4,312.07 |
| Sustained usage discount | -$420.54 |
| Total estimated cost | $3,891.53/mo |

*Figure 5-16 Creating a virtual server instance with the z/OS Development and Test image*

8. Todd reviews the cost estimates for his configuration choices in the right pane and clicks **Create virtual server** to deploy his new VSI. The new VSI appears in his list of VSIs in the VPC management window.

9. To access the VSI without setting up a VPN, he temporarily creates a floating IP address by selecting his VSI from the list, going to the details window, and by using the Network Interfaces window to reserve and assign the VSI. He can try connecting to the VSI through SSH by using the following command:

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

10. After he is 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 IBM Wazi Dev Spaces in a VPC

After he creates a VPC and VSI in 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 IBM Wazi for Dev Spaces on Red Hat OpenShift are in public IBM and Red Hat catalogs that must be pulled from the internet. To do so, Todd must configure a public gateway that makes the internet reachable for his subnets for outgoing traffic by completing the following steps:

1. He uses the VPC management pages in IBM Cloud to create and assign a public gateway by logging in to IBM Cloud.
2. He clicks **Create**.
3. He uses the Subnets management page to go into each subnet and attaches the public gateway with the slider.
4. He also must order encrypted cloud storage, which is required by IBM Wazi for Dev Spaces to persist the developer's virtual workspaces. The quickest way to order this storage is to use the Search widget at the top of an IBM Cloud page to search for `Object Storage`, select **IBM Cloud** as the infrastructure, and select the **Lite** plan.
5. He provides a name for his Cloud Object Storage and creates it.

Todd is ready to deploy a Red Hat OpenShift cluster into his VPC, so he completes the following steps:

1. Todd uses the Search widget at the top of an 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. In the right pane, he reviews the cost estimates and clicks **Create**.

After the deployment finishes, he clicks **Red Hat OpenShift web console** on the cluster window to open the Red Hat OpenShift console.

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

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

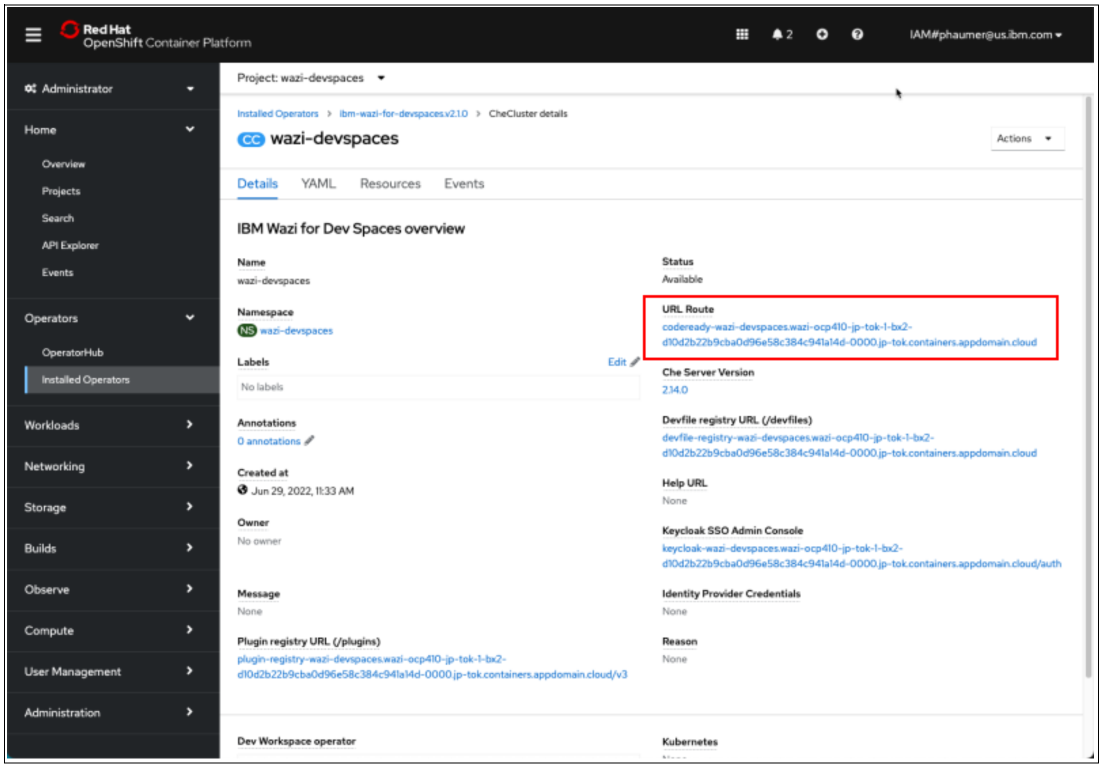
*Example 5-1 YAML to configure the IBM Catalog*

```
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. After the catalog is added, he selects **Operator** → **OperatorHub** and searches for `wazi` in the Filter by keyword field.
3. He clicks the IBM Wazi for Dev Spaces operator. A summary window opens.
4. He clicks **Install** and accepts all the default settings, and then clicks **Install** again.
5. The operator installation prompts him to create a license. He clicks the link, accepts the license terms, and installs the license.

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

1. In the Create Cluster page, he accepts all the defaults and clicks **Create**.
2. When the creation finishes and IBM Wazi for Dev Spaces is deployed, he can open the instance from the IBM Wazi for Dev Spaces list and find all the details that he needs, as shown in Figure 5-17.



IBM Wazi for Dev Spaces overview

Name
wazi-devspaces

Namespace
NS wazi-devspaces

Labels
No labels

Annotations
0 annotations

Created at
Jun 29, 2022, 11:33 AM

Owner
No owner

Message
None

Plugin registry URL (/plugins)
plugin-registry-wazi-devspaces.wazi-ocp410-jp-tok-1-bx2-d10d2b22b9cba0d96e58c384c941a14d-0000.jp-tok.containers.appdomain.cloud/v3

Status
Available

URL Route
codeready-wazi-devspaces.wazi-ocp410-jp-tok-1-bx2-d10d2b22b9cba0d96e58c384c941a14d-0000.jp-tok.containers.appdomain.cloud

Che Server Version
2.14.0

Devfile registry URL (/devfiles)
devfile-registry-wazi-devspaces.wazi-ocp410-jp-tok-1-bx2-d10d2b22b9cba0d96e58c384c941a14d-0000.jp-tok.containers.appdomain.cloud

Help URL
None

Keycloak SSO Admin Console
keycloak-wazi-devspaces.wazi-ocp410-jp-tok-1-bx2-d10d2b22b9cba0d96e58c384c941a14d-0000.jp-tok.containers.appdomain.cloud/auth

Identity Provider Credentials
None

Reason
None

Dev Workspace operator

Kubernetes

*Figure 5-17 Deployed IBM Wazi for Dev Spaces instances*

Todd is using the Administrator menu in the left pane of the Red Hat OpenShift console to select **Workloads** → **Secrets** and open `che-identity-secret`. Here, he finds the password that is generated that is used for several administrative purposes by clicking **Reveal values**.

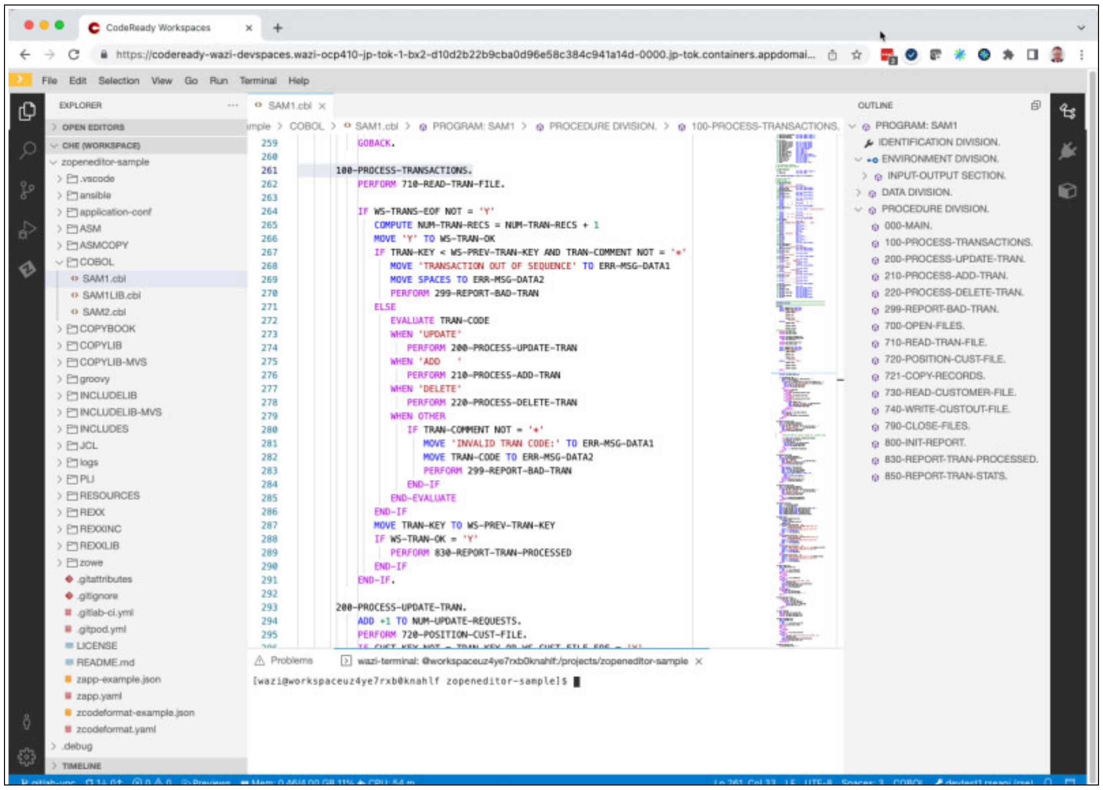
Back in the Dev Spaces instance window that is shown in Figure 5-17, he uses the link to the **Keycloak SSO Admin Console** with the password he looked up to create accounts for Deb, Kathleen, and himself. For more information about managing identities and authorizations, see Managing identities and authorizations.

IBM Wazi Dev Spaces is ready for Deb and Kathleen to use. Todd copies the links that are shown under URL Route (Figure 5-17 on page 102) and sends the links to them as the URL that they use to log on.

## 5.5.3 Creating and configuring a development workspace in IBM Wazi Dev Spaces

Now that Todd created an IBM Wazi for Dev Spaces deployment in IBM Cloud running in a VPC and created user accounts, Kathleen can use the URL that was 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. Because 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 and Zowe Explorer, and Ansible automation clients and sample applications for all the languages that she can use for experimentation.

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



```
                GOBACK.

            100-PROCESS-TRANSACTIONS.
                PERFORM 710-READ-TRAN-FILE.

                IF WS-TRANS-EOF NOT = 'Y'
                   COMPUTE NUM-TRAN-RECS = NUM-TRAN-RECS + 1
                   MOVE 'Y' TO WS-TRAN-OK
                   IF TRAN-KEY < WS-PREV-TRAN-KEY AND TRAN-COMMENT NOT = '*'
                      MOVE 'TRANSACTION OUT OF SEQUENCE' TO ERR-MSG-DATA1
                      MOVE SPACES TO ERR-MSG-DATA2
                      PERFORM 299-REPORT-BAD-TRAN
                   ELSE
                      EVALUATE TRAN-CODE
                      WHEN 'UPDATE'
                         PERFORM 200-PROCESS-UPDATE-TRAN
                      WHEN 'ADD   '
                         PERFORM 210-PROCESS-ADD-TRAN
                      WHEN 'DELETE'
                         PERFORM 220-PROCESS-DELETE-TRAN
                      WHEN OTHER
                         IF TRAN-COMMENT NOT = '*'
                            MOVE 'INVALID TRAN CODE:' TO ERR-MSG-DATA1
                            MOVE TRAN-CODE TO ERR-MSG-DATA2
                            PERFORM 299-REPORT-BAD-TRAN
                         END-IF
                      END-EVALUATE
                   END-IF
                   MOVE TRAN-KEY TO WS-PREV-TRAN-KEY
                   IF WS-TRAN-OK = 'Y'
                      PERFORM 830-REPORT-TRAN-PROCESSED
                   END-IF
                END-IF.

            200-PROCESS-UPDATE-TRAN.
                ADD +1 TO NUM-UPDATE-REQUESTS.
                PERFORM 720-POSITION-CUST-FILE.
```

```
wazi-terminal: @workspaceuz4ye7rxb0knahlf:/projects/zopeneditor-sample
[wazi@workspaceuz4ye7rxb0knahlf zopeneditor-sample]$
```

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

Kathleen opens her first COBOL program from the sample repository 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 cursor over a variable, she sees its definition, which she can right-click and go to that definition in the code or review other locations in the program where it is used. She also can write new lines by using code completion by pressing Ctrl-Spacebar to open choices for completing a command for the language in which she is working. She also discovers that the code completion menu provides many code templates from which she can choose. 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 make and build code changes on her team's WaaS VSI and start a Debug session from there too. For that task, the repository comes with sample automation scripts that are written in Ansible (for configuring the new z/OS VSI) and Groovy (for running builds on z/OS with IBM DBB).

To use the Ansible script, she edits an Ansible configuration file that contains all the specific host variables that are needed for the run time. The IBM Wazi for Dev Spaces sample repository includes configuration files that are prepared for the WaaS Development and Test image (and the IBM Wazi Sandbox image), so all she must do is provide the IP address for the VSI and the username that Todd enabled for her. For the tutorial, we did not set up a VPN, but a public IP address, so Todd must provide that address to the developers. He finds it in the VSI list of the VPC home page 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 listed there. She also replaces the username with `ibmuser`. She sees that editing the file also comes with advanced features because the development stack that she selected for her IBM 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 run. 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 by hovering her cursor in the editor, such as the documentation of most Ansible commands, syntax highlighting and errors in the Problems view, and code completion.

She opens `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 her for the following information:

1. For the password of the user that she specified in the configuration file, which generates several files.
2. For the version of Zowe CLI profiles that she wants to create. She confirms the default of `v7`.
3. For the editor that she is using. She enters `che` because she is in IBM Wazi for Dev Spaces.
4. Whether she wants to overwrite previously created files. She enters `yes`.

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

1. Finishing the Zowe CLI setup depends on how she answered the prompts. The script created a Zowe team configuration file for her host devtest1 and placed it in `~/.zowe/zowe.config.json`. To complete the setup, she runs the following command to complete the Zowe configuration:

   ```
   zowe config update-schemas
   ```

2. The script output informs her that the script created an IBM Z Open Debug launch for her that she must copy into her workspace's `.vscode` folder.
3. Finally, the script printed JSON values that she must add to her IBM Wazi for Dev Spaces user settings to run the debugger. To open the Preferences window, she selects **File** → **Settings** → **Open Preferences**, and then she switches to the JSON view and pastes the provided JSON at the end (before the final closing brace).

Because Todd has not contacted Zach to configure the debugger with his company's signed SSL certificates, Todd decided to use the debugger with self-signed certificates for now. He provides Kathleen and Deb with the instructions to run these certificates manually. He instructs them to open a new browser tab and go to `<https://vsi-ip-addres:8192/api/v1/remote-debug/config>` to accept the self-signed certificate into the browser so that IBM Wazi for Dev Spaces can use it for connecting to the debugger.

Now, Kathleen configured a connection to her WaaS VSI by using Zowe and configured the debugger to work with her VSI. She tests her configuration by switching to the Zowe Explorer view in the editor. She uses the z/OS UNIX System Services view to connect to the VSI and look at the `ibmuser` home directory as `/u/ibmuser`. For more information about using the Zowe Explorer, see Using the Zowe Explorer views.

In the `/u/ibmuser` home directory, Kathleen opens the file `.ssh/authorized_hosts` and pastes and saves her IBM Wazi for Dev Spaces Workspace's SSH key, which she retrieved from the command menu by pressing Ctrl-Shift-P (Windows) or Cmd-Shift-P (Mac), and then selecting **SSH: View public keys** → **Default**. For more information, see Managing Git configuration: identity.

Next, Kathleen wants to build and run a COBOL application. To do so, she must prepare the IBM DBB setup on the z/OS VSI. Its development and test image is preinstalled and preconfigured for IBM DBB, so she must configure only her user account. The sample repository has another Ansible playbook for the configuration task.

She runs the playbooks by using the following command:

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

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

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

Kathleen also sees that the script cloned a Git repository remotely on z/OS UNIX System Services into the folder `/u/ibmuser/projects` that contains the latest version of IBM dbb-zappbuild with the build scripts that are 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, and lists the data sets for the compiler, libraries, and ports. 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 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 focus on working on an application. She must build it, run debug sessions to trace through its functions and her changes, and run it with test data to ensure that 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 similar UX as though Deb were building, running, and debugging a Java or JavaScript application.

The team wants simple operations for the following tasks:

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 functions.
3. Quickly building the program that is being edited to ensure that it compiles.
4. Launching a debug session of the program that was built to step through its functions.

IBM Wazi for Dev Spaces and the sample repository have Ansible playbooks and editor launches for these operations. In our example, we complete the following steps:

1. Kathleen runs the playbook by using the following command to build the entire application. This playbook also generates and uploads 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 opened earlier. She performs a right-click operation from within the source code and selects **Run IBM User Build** (the first time she runs it, she must select **Run setup for IBM User Build**). This action starts a user build session. The editor opens an output view and shows the log for uploading the program and its local copybook dependencies and starting an IBM DBB build by using the `dbb-zappbuild` repository's scripts.
4. After the build succeeds, Kathleen can start a debug session. 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 and selects **Submit** to start the session. She switches to the editor's Debug view and selects the launch that was created by the initialized playbook that she ran at the beginning. She clicks **Play**, and the debug session starts in the editor. She can set breakpoints and examine the value of variables. The UX is the same as with debugging Java or TypeScript.

## 5.6 Summary

Our review of the Enterprise DevOps Enabler Patterns and the IBM technology portfolio around them is concluded. We described many important best practices for DevOps and how they apply to hybrid development projects. We explored key tools and technologies that help enable development teams to realize these best practices. We presented best practices as a set of layers that build on top of each other while also recounting the history of IBM Z DevOps, IBM Wazi, and IBM Z and Cloud Modernization Stack. We 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 more information that summarizes many of the core concepts of this chapter, and to see the end-to-end demonstration from the tutorial section in action, see this YouTube video.

6

# Managing your applications

This chapter describes the implications of application monitoring and management aspects in a hybrid cloud architecture. Additionally, it describes Red Hat OpenShift implementations on IBM Z.

This chapter covers the following topics:

- Monitoring, logging, and metering introduction
- Components of the Red Hat OpenShift monitoring stack
- Observability on z/OS
- Logging
- Metering

## 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.

These goals can be achieved by applying the following different techniques:

- Monitoring: Composed 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 administrators and operations teams use to gain insights into the overall system's health.
- Metering: Refers to the ability of collecting metrics that administrators, operations teams, and users use to understand the 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 components:

- 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. These components provide 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 on page 111.
- Components for monitoring user-defined projects: After optionally enabling monitoring for user-defined projects, more monitoring components are installed in the openshift-user-workload-monitoring project, which provides monitoring for user-defined projects. These components are illustrated in the "User" section of Figure 6-1 on page 111.

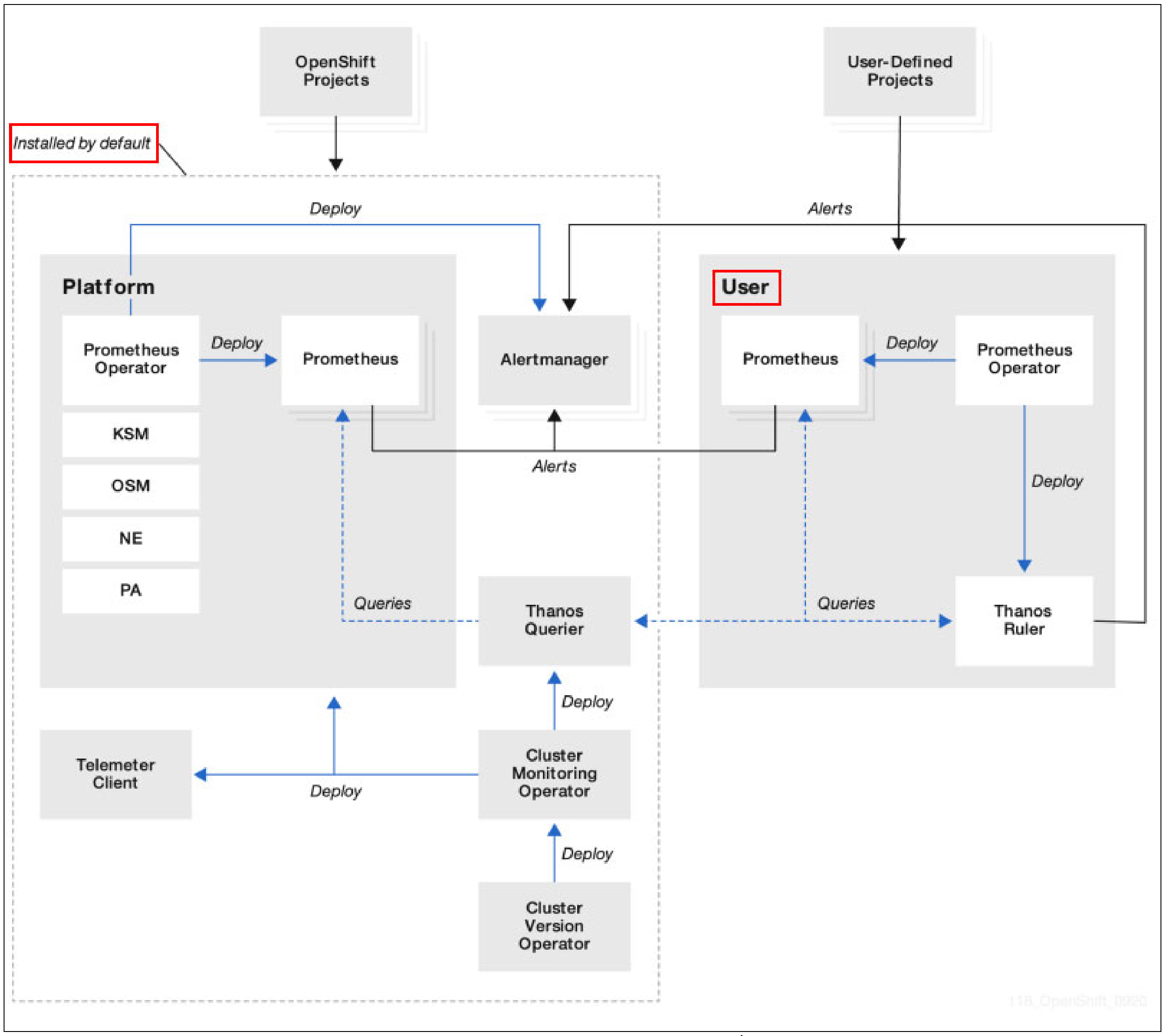


*Figure 6-1 Red Hat OpenShift Container Platform monitoring stack diagram*[1]

Ideally, you should configure persistent storage for your running Red Hat OpenShift cluster to store that data into a persistent volume (PV) so that it can survive a pod restart or re-creation. 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 are provisioned dynamically, for example, Red Hat Data Foundation provides that capability.

Monitoring user-defined workloads is beyond of the scope of this publication, but to contrast user-defined projects versus standard monitoring of components of a project or namespace, consider the following example: A PostrgeSQL database is deployed in a project that is named Database, and uses the standard monitoring capabilities of the built-in monitoring stack, so you can check the amount of CPU, memory, network usage, and others.

1 Source: https://docs.openshift.com/container-platform/4.11/monitoring/monitoring-overview.html

The user-defined monitoring capability that is described by Red Hat documentation means that extended monitoring capabilities can be added, such as the number of connections to the database, which extends the monitoring capabilities beyond the standard capabilities. For more information about enabling the monitoring of user-defined projects, see Configuring and using the monitoring stack in OpenShift Container Platform.

### 6.2.1 Monitoring the Red Hat OpenShift Container Platform infrastructure by using Prometheus

This section covers the built-in monitoring stack that comes with Red Hat OpenShift Container Platform.

Red Hat OpenShift Container Platform provides preconfigured and self-updating monitoring as standard. You can monitor the platform's core components and user-defined projects. By using the user-defined monitoring, the cluster administrators, developers, and other types of users can specify how services and pods are monitored for each project (also known as a *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 that are covered by the standard monitoring stack.

This section does not replace the official Red Hat OpenShift Platform documentation. For more information, see Monitoring overview.

In summary, the Red Hat OpenShift Container Platform web console provides a way to view and manage metrics and alerts, and to access the monitoring dashboards.

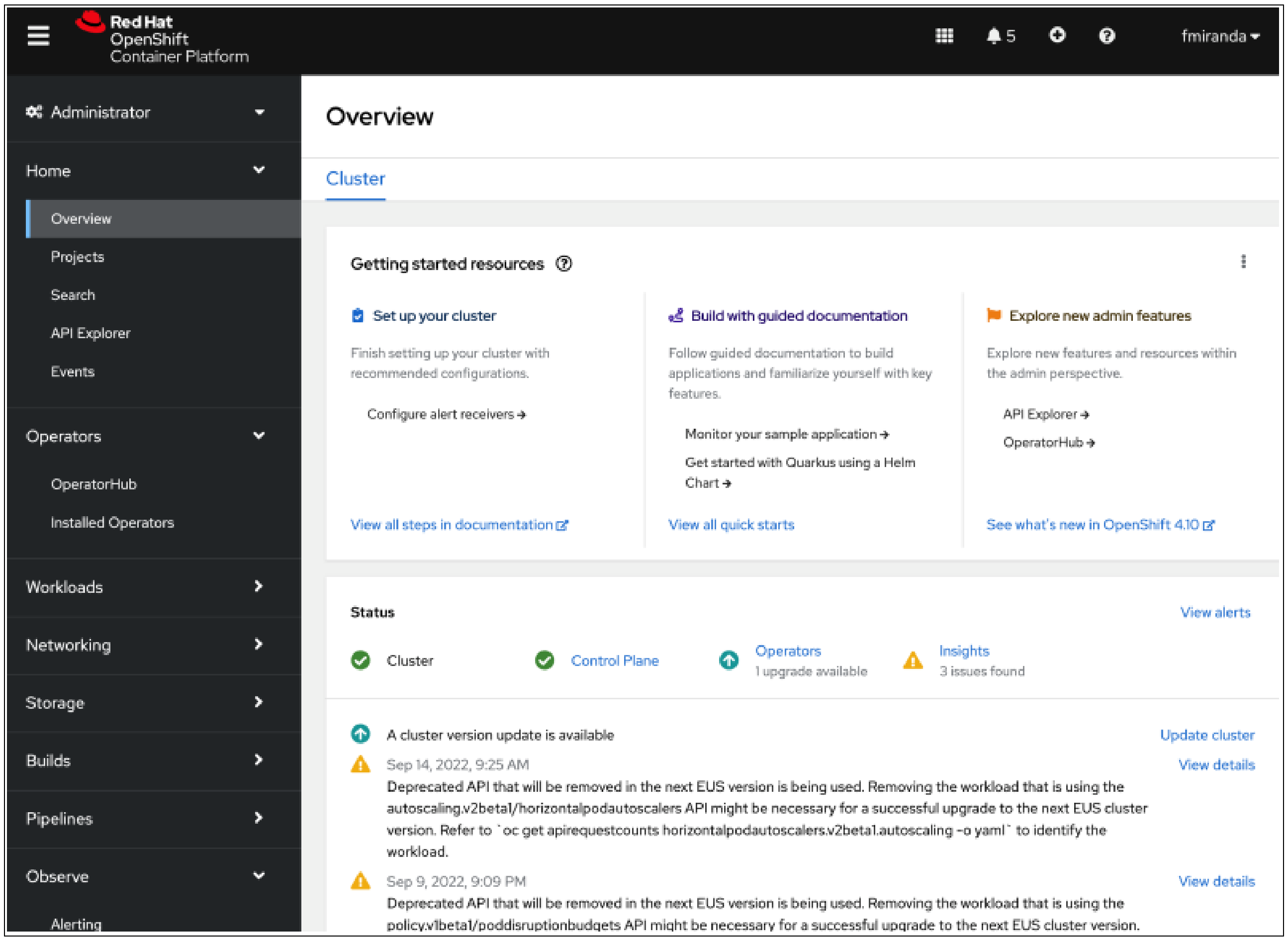
Besides the built-in monitoring dashboards, Red Hat OpenShift Container Platform also offers support for third-party interfaces such as Prometheus, Alertmanager, and Grafana.

### 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 dashboard as the cluster administrator

The Overview dashboard (shown in Figure 6-2 on page 113) provides information such as alerts; the status of the cluster and control plane nodes; the status of operators; and whether your cluster is deployed as a connected cluster. You can leverage the Insights capability to prevent issues on your environment by tapping into a huge database of information from Red Hat and leveraging the artificial intelligence (AI) capability to create alerts of potential issues in your environment.



Overview

Cluster

Getting started resources

Set up your cluster

Finish setting up your cluster with recommended configurations.

Configure alert receivers →

View all steps in documentation

Build with guided documentation

Follow guided documentation to build applications and familiarize yourself with key features.

Monitor your sample application →

Get started with Quarkus using a Helm Chart →

View all quick starts

Explore new admin features

Explore new features and resources within the admin perspective.

API Explorer →

OperatorHub →

See what's new in OpenShift 4.10

Status | View alerts

Cluster | Control Plane | Operators 1 upgrade available | Insights 3 issues found

A cluster version update is available | Update cluster

Sep 14, 2022, 9:25 AM | View details

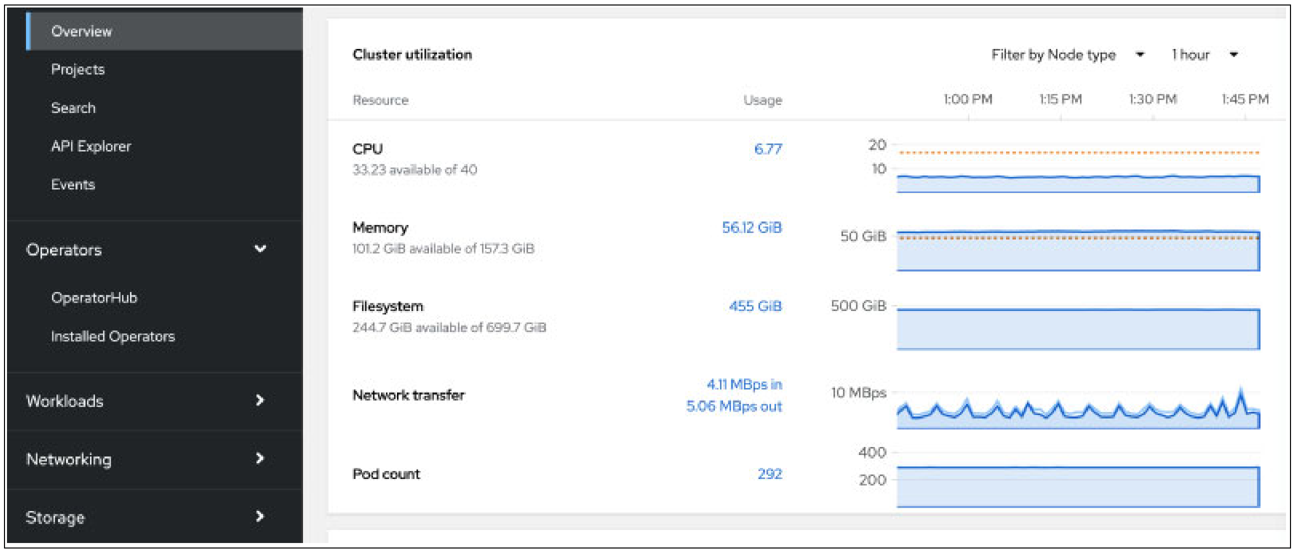
Deprecated API that will be removed in the next EUS version is being used. Removing the workload that is using the autoscaling.v2beta1/horizontalpodautoscalers API might be necessary for a successful upgrade to the next EUS cluster version. Refer to `oc get apirequestcounts horizontalpodautoscalers.v2beta1.autoscaling -o yaml` to identify the workload.

Sep 9, 2022, 9:09 PM | View details

Deprecated API that will be removed in the next EUS version is being used. Removing the workload that is using the policy.v1beta1/poddisruptionbudgets API might be necessary for a successful upgrade to the next EUS cluster version.

*Figure 6-2 Red Hat OpenShift dashboard web console overview*

Scrolling down through the Overview dashboard window, you see real-time monitoring of your Red Hat OpenShift cluster of total utilization of CPU, Memory, Filesystem, Network, and Pod count, as shown in Figure 6-3.



Cluster utilization

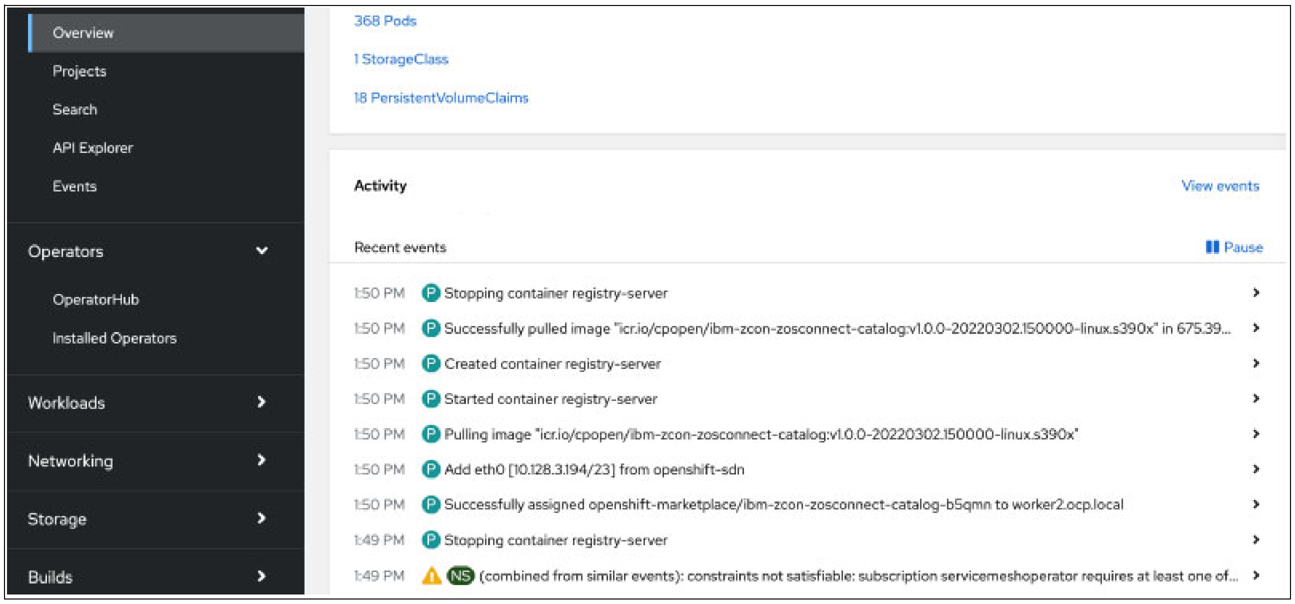
| Resource | Usage |
|---|---|
| CPU<br>33.23 available of 40 | 6.77 |
| Memory<br>101.2 GiB available of 157.3 GiB | 56.12 GiB |
| Filesystem<br>244.7 GiB available of 699.7 GiB | 455 GiB |
| Network transfer | 4.11 MBps in<br>5.06 MBps out |
| Pod count | 292 |

*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 PVs are used for the monitoring stack, data is saved in the local node where the stack is running. Data is lost if the pod is lost or if the node is unavailable, so as a best practice, add a persistent storage layer to your Red Hat OpenShift Cluster so that no historical data is lost.

To change the amount of time that historic monitoring data is kept, see Modifying the retention time and size for Prometheus metrics data.

Further down on the Overview window, you can check the activity of the cluster, where it explores the most recent events from this cluster, as shown in Figure 6-4.



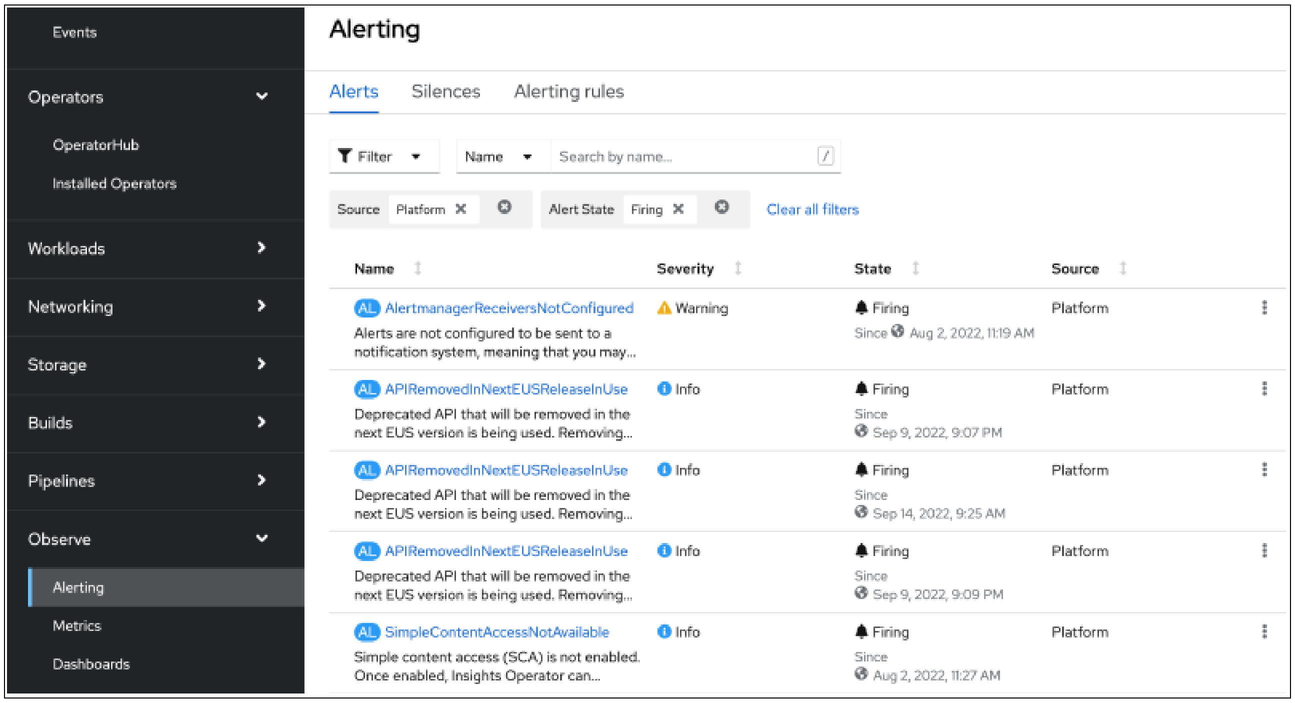
```
1:50 PM  P Stopping container registry-server
1:50 PM  P Successfully pulled image "icr.io/cpopen/ibm-zcon-zosconnect-catalog:v1.0.0-20220302.150000-linux.s390x" in 675.39...
1:50 PM  P Created container registry-server
1:50 PM  P Started container registry-server
1:50 PM  P Pulling image "icr.io/cpopen/ibm-zcon-zosconnect-catalog:v1.0.0-20220302.150000-linux.s390x"
1:50 PM  P Add eth0 [10.128.3.194/23] from openshift-sdn
1:50 PM  P Successfully assigned openshift-marketplace/ibm-zcon-zosconnect-catalog-b5qmn to worker2.ocp.local
1:49 PM  P Stopping container registry-server
1:49 PM  ⚠ NS (combined from similar events): constraints not satisfiable: subscription servicemeshoperator requires at least one of...
```

*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 provides access to the Alerting feature of Red Hat OpenShift and other dashboards that show resources that are monitored.

Figure 6-5 on page 115 shows the Alerting feature in Red Hat OpenShift.

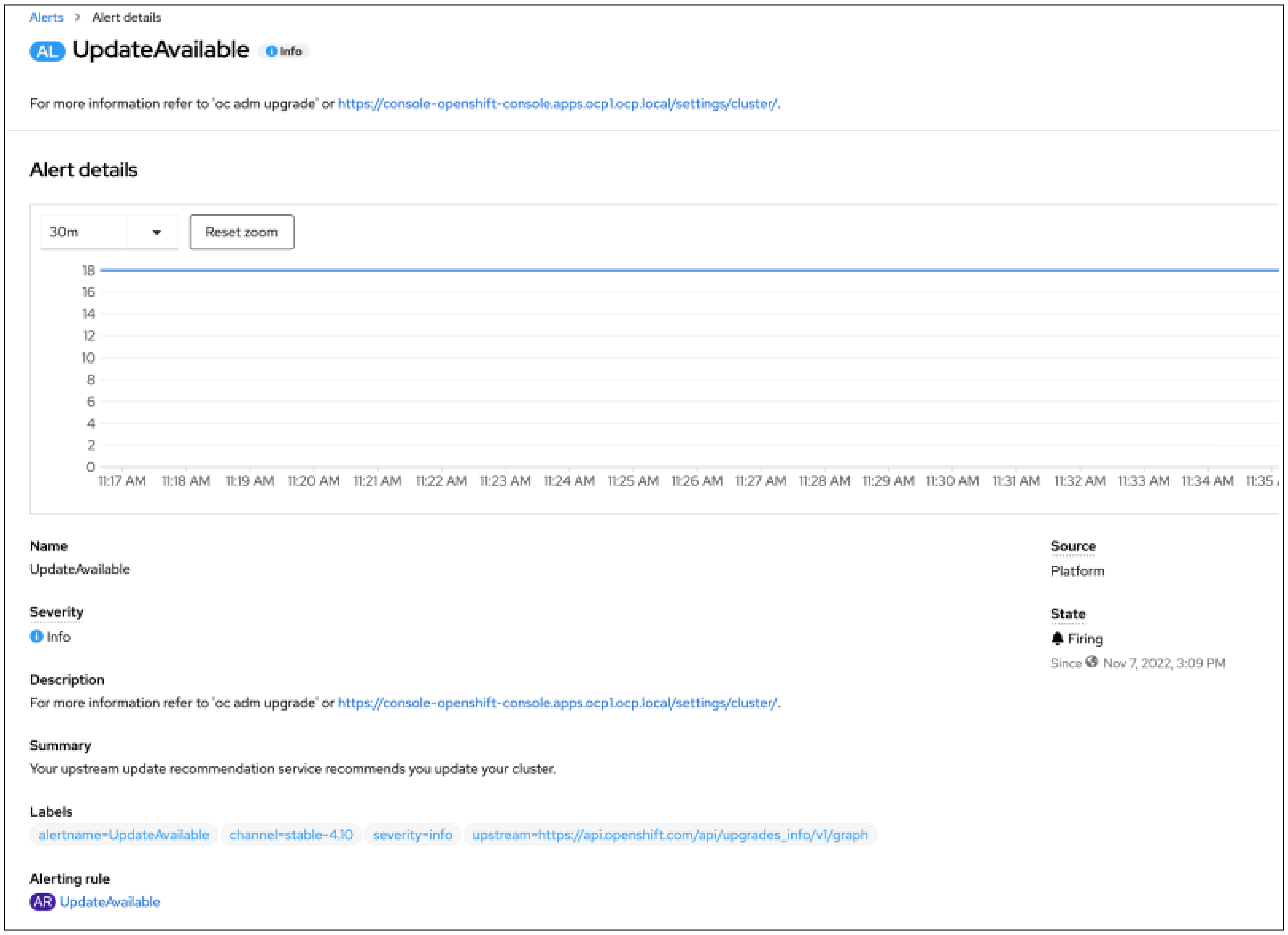


| Name | Severity | State | Source |
|---|---|---|---|
| AL AlertmanagerReceiversNotConfigured<br>Alerts are not configured to be sent to a notification system, meaning that you may... | Warning | Firing<br>Since Aug 2, 2022, 11:19 AM | Platform |
| AL APIRemovedInNextEUSReleaseInUse<br>Deprecated API that will be removed in the next EUS version is being used. Removing... | Info | Firing<br>Since Sep 9, 2022, 9:07 PM | Platform |
| AL APIRemovedInNextEUSReleaseInUse<br>Deprecated API that will be removed in the next EUS version is being used. Removing... | Info | Firing<br>Since Sep 14, 2022, 9:25 AM | Platform |
| AL APIRemovedInNextEUSReleaseInUse<br>Deprecated API that will be removed in the next EUS version is being used. Removing... | Info | Firing<br>Since Sep 9, 2022, 9:09 PM | Platform |
| AL SimpleContentAccessNotAvailable<br>Simple content access (SCA) is not enabled. Once enabled, Insights Operator can... | Info | Firing<br>Since Aug 2, 2022, 11:27 AM | Platform |

*Figure 6-5 Red Hat OpenShift Alerting feature*

Alerts can be used to monitor certain condition of the cluster and aspects of user-defined workloads. Configuring a new alert is beyond the scope of this publication, but we 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 that is available for the Red Hat OpenShift cluster. Regardless of whether the administrator is working connected or disconnected deployments of Red Hat OpenShift, they receive an alert whenever there is an update that is available to this particular cluster.



For more information refer to 'oc adm upgrade' or https://console-openshift-console.apps.ocp1.ocp.local/settings/cluster/.

Name
UpdateAvailable

Severity
Info

Description
For more information refer to 'oc adm upgrade' or https://console-openshift-console.apps.ocp1.ocp.local/settings/cluster/.

Summary
Your upstream update recommendation service recommends you update your cluster.

Labels
alertname=UpdateAvailable channel=stable-4.10 severity=info upstream=https://api.openshift.com/api/upgrades_info/v1/graph

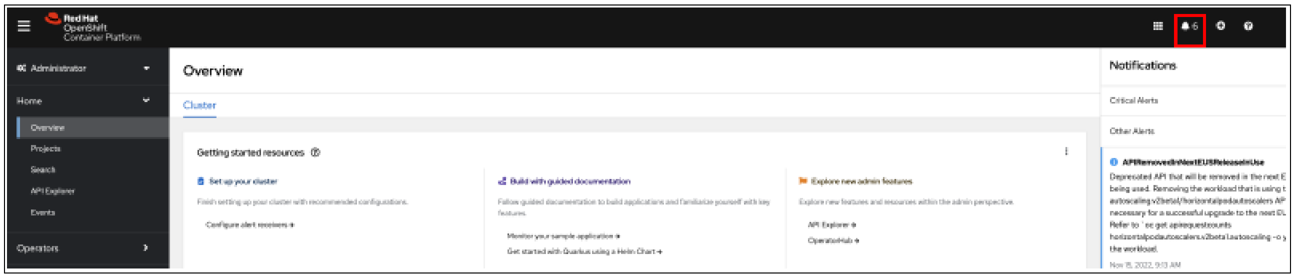
Alerting rule
AR UpdateAvailable

Source
Platform

State
Firing
Since Nov 7, 2022, 3:09 PM

Figure 6-6 Red Hat OpenShift Alerting example

In this cluster, as an example, this alert was triggered, and it shows in the Red Hat OpenShift web console in the bell icon at the upper right of the window (Figure 6-7).


Red Hat OpenShift Container Platform
6
Administrator
Home
Overview
Projects
Search
API Explorer
Events
Operators
Overview
Cluster
Getting started resources
Set up your cluster
Configure alert receivers
Build with guided documentation
Monitor your sample application
Get started with Quarkus using a Helm Chart
Explore new admin features
API Explorer
OperatorHub
Notifications
Critical Alerts
Other Alerts

Finish setting up your cluster with recommended configurations.

Follow guided documentation to build applications and familiarize yourself with key features.

Explore new features and resources within the admin perspective.

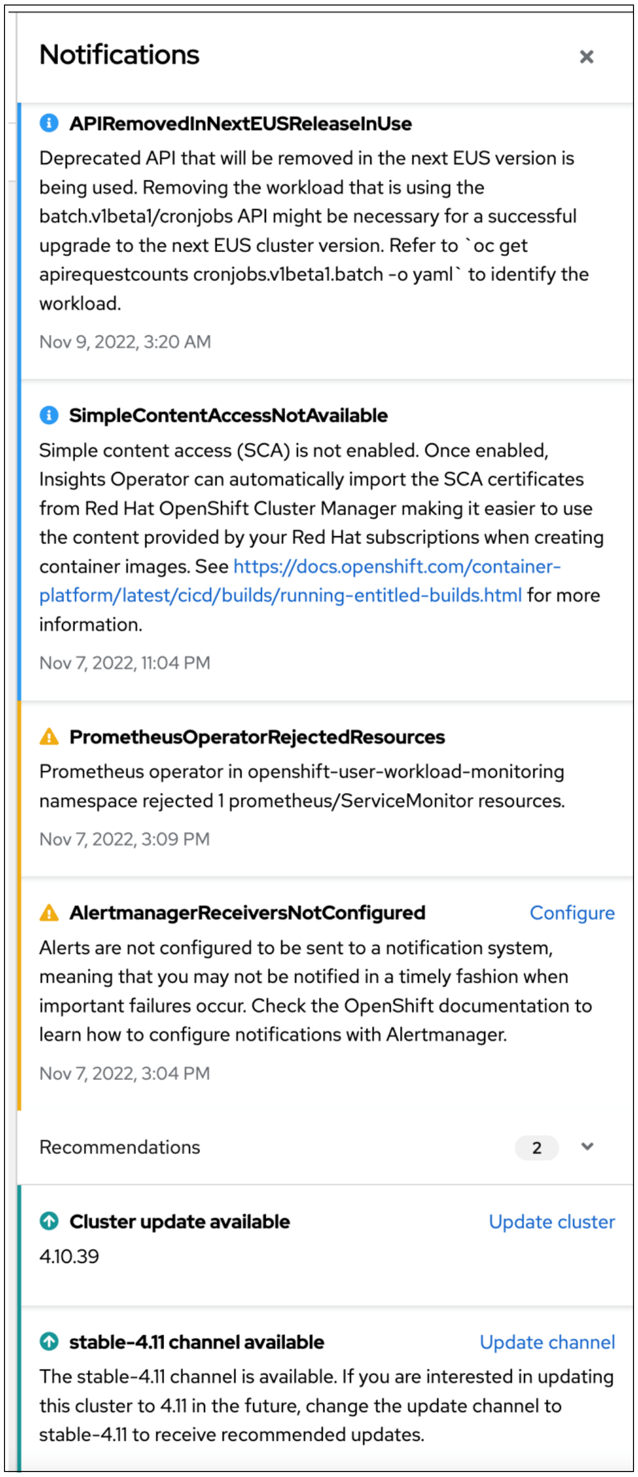
APIRemovedInNextEUSReleaseInUse

Deprecated API that will be removed in the next E being used. Removing the workload that is using t autoscaling/v2beta1/horizontalpodautoscalers AP necessary for a successful upgrade to the next EU Refer to `oc get apirequestcounts horizontalpodautoscalers.v2beta1.autoscaling -o y the workload.

Nov 15, 2022, 9:13 AM

Figure 6-7 Red Hat OpenShift Notification bell

In Figure 6-7 on page 116, there are six alerts that are triggered, and if we scroll down the list, we see that a cluster update is available, as shown in Figure 6-8.


Notifications

**APIRemovedInNextEUSReleaseInUse**
Deprecated API that will be removed in the next EUS version is being used. Removing the workload that is using the batch.v1beta1/cronjobs API might be necessary for a successful upgrade to the next EUS cluster version. Refer to `oc get apirequestcounts cronjobs.v1beta1.batch -o yaml` to identify the workload.
Nov 9, 2022, 3:20 AM

**SimpleContentAccessNotAvailable**
Simple content access (SCA) is not enabled. Once enabled, Insights Operator can automatically import the SCA certificates from Red Hat OpenShift Cluster Manager making it easier to use the content provided by your Red Hat subscriptions when creating container images. See https://docs.openshift.com/container-platform/latest/cicd/builds/running-entitled-builds.html for more information.
Nov 7, 2022, 11:04 PM

**PrometheusOperatorRejectedResources**
Prometheus operator in openshift-user-workload-monitoring namespace rejected 1 prometheus/ServiceMonitor resources.
Nov 7, 2022, 3:09 PM

**AlertmanagerReceiversNotConfigured** Configure
Alerts are not configured to be sent to a notification system, meaning that you may not be notified in a timely fashion when important failures occur. Check the OpenShift documentation to learn how to configure notifications with Alertmanager.
Nov 7, 2022, 3:04 PM

Recommendations 2

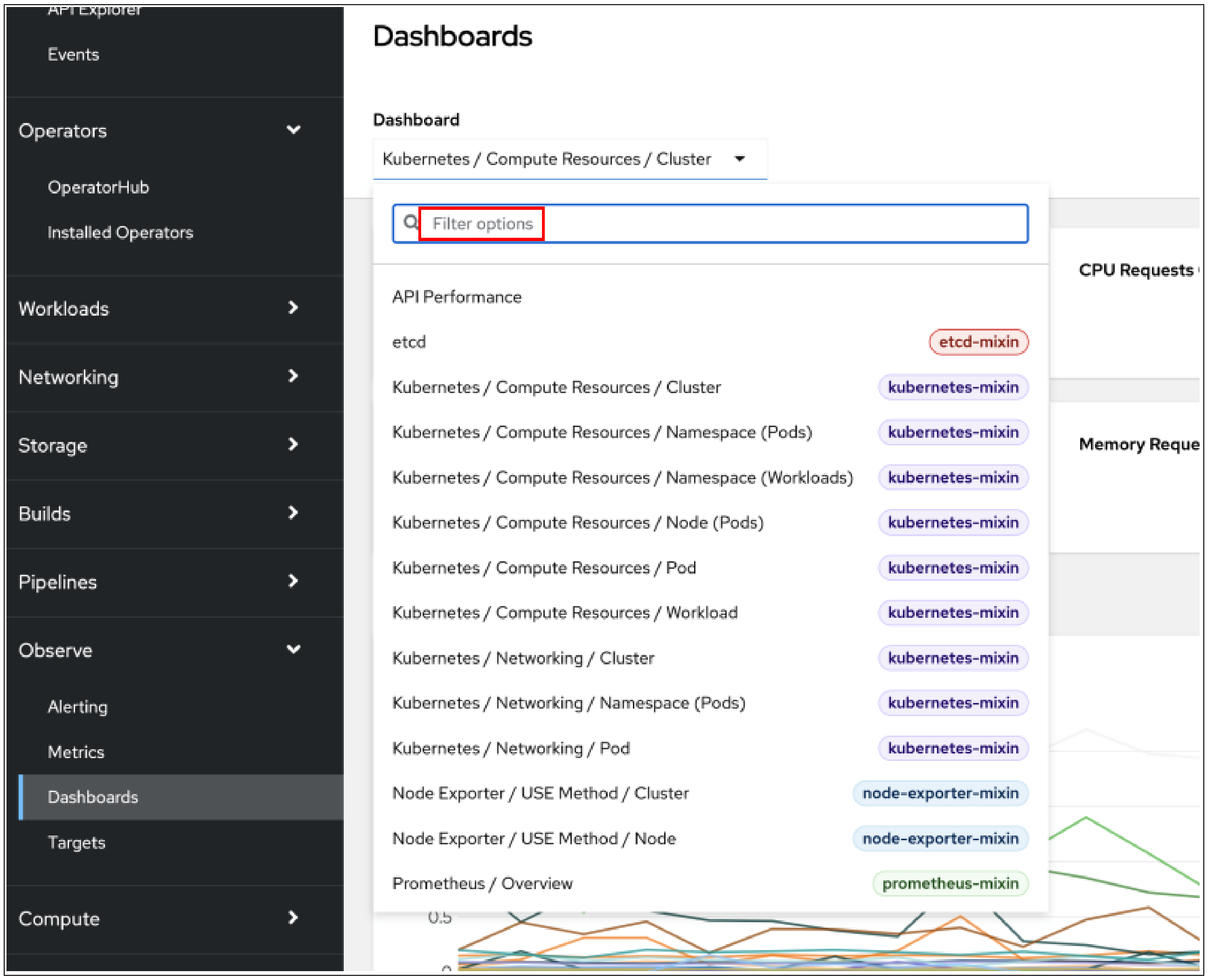
**Cluster update available** Update cluster
4.10.39

**stable-4.11 channel available** Update channel
The stable-4.11 channel is available. If you are interested in updating this cluster to 4.11 in the future, change the update channel to stable-4.11 to receive recommended updates.

*Figure 6-8 Red Hat OpenShift notifications expanded*

### 6.2.4 Exploring cluster monitoring data from different sources, such as cluster nodes, projects, or pods

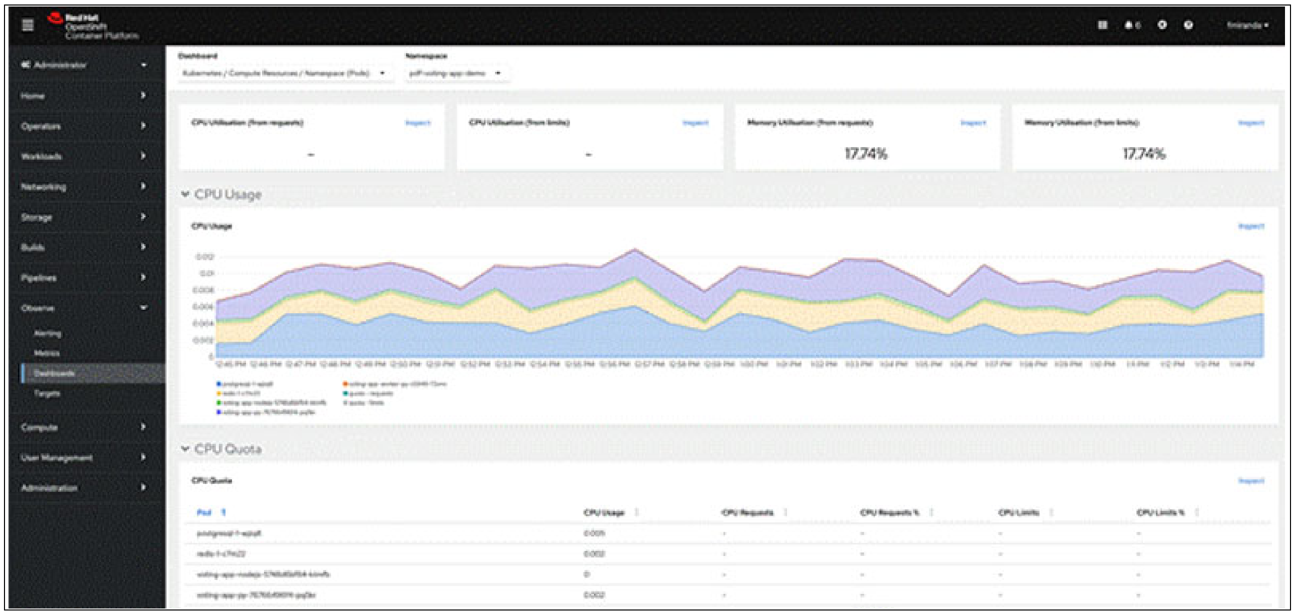
If you select **Observe** → **Dashboards** on the Red Hat OpenShift web console, a drop-down menu offers many options to show monitoring data about the Red Hat OpenShift cluster, as shown in Figure 6-9.



| | |
|---|---|
| API Performance | |
| etcd | etcd-mixin |
| Kubernetes / Compute Resources / Cluster | kubernetes-mixin |
| Kubernetes / Compute Resources / Namespace (Pods) | kubernetes-mixin |
| Kubernetes / Compute Resources / Namespace (Workloads) | kubernetes-mixin |
| Kubernetes / Compute Resources / Node (Pods) | kubernetes-mixin |
| Kubernetes / Compute Resources / Pod | kubernetes-mixin |
| Kubernetes / Compute Resources / Workload | kubernetes-mixin |
| Kubernetes / Networking / Cluster | kubernetes-mixin |
| Kubernetes / Networking / Namespace (Pods) | kubernetes-mixin |
| Kubernetes / Networking / Pod | kubernetes-mixin |
| Node Exporter / USE Method / Cluster | node-exporter-mixin |
| Node Exporter / USE Method / Node | node-exporter-mixin |
| Prometheus / Overview | prometheus-mixin |

*Figure 6-9 Red Hat OpenShift Observe dashboard*

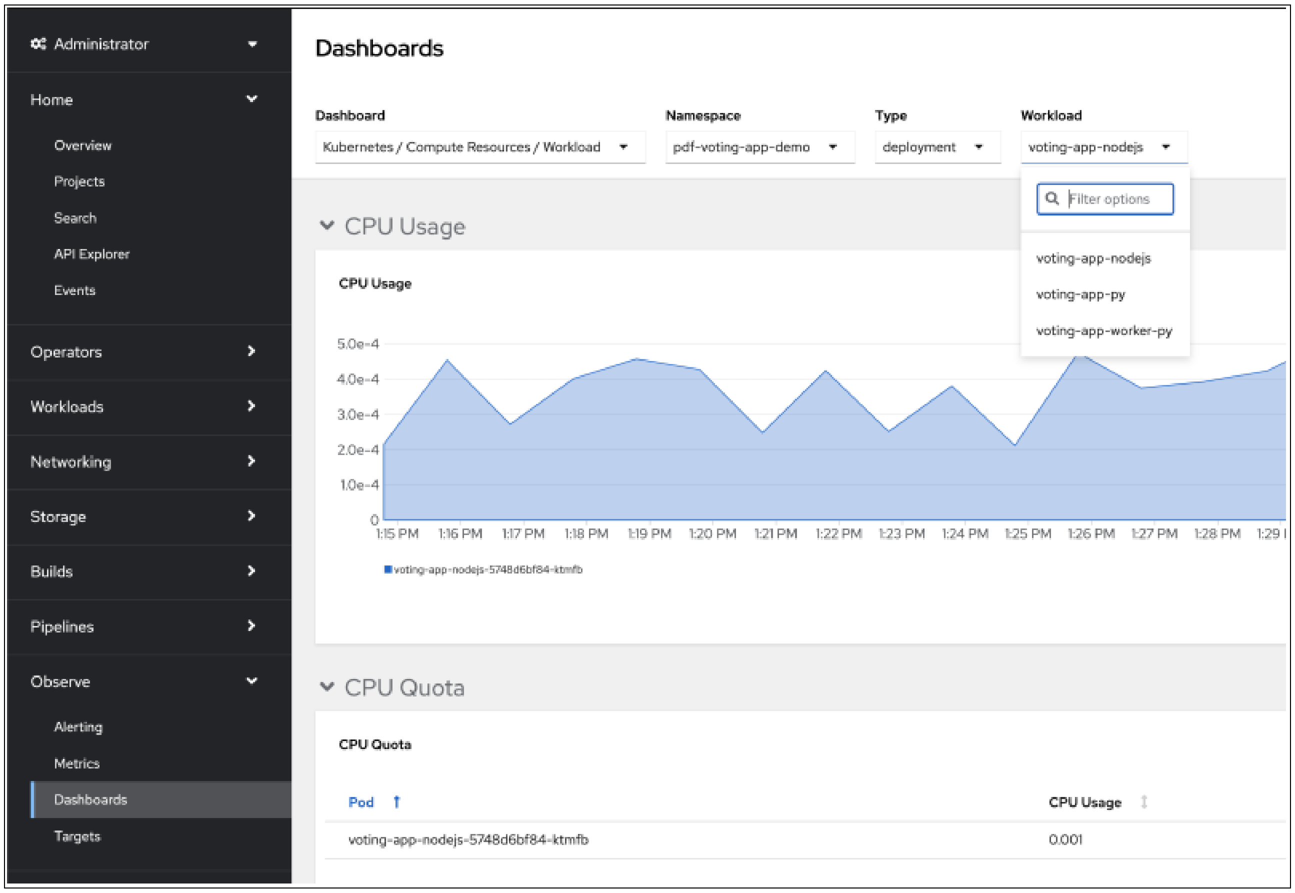
For demonstration purposes, a few options are selected from that list of items. Starting with **Kubernetes /Compute Resources / Namespaces (Pods)**, as shown in Figure 6-10. It shows monitoring information about a user workload (not user-defined monitoring information) about a specific namespace (project). 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 using, such as CPU, memory, and network usage.



| Pod | CPU Usage | CPU Requests | CPU Requests % | CPU Limits | CPU Limits % |
|---|---|---|---|---|---|
| postgresql-1-wjbqft | 0.005 | - | - | - | - |
| redis-1-c7m22 | 0.002 | - | - | - | - |
| voting-app-nodejs-5766d[illegible] | 0 | - | - | - | - |
| voting-app-py-7676[illegible] | 0.002 | - | - | - | - |

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 shown in Figure 6-11. By using this option, you can select the Namespace, the Type of deployment, and Workload (in this case, the different microservices of this application).



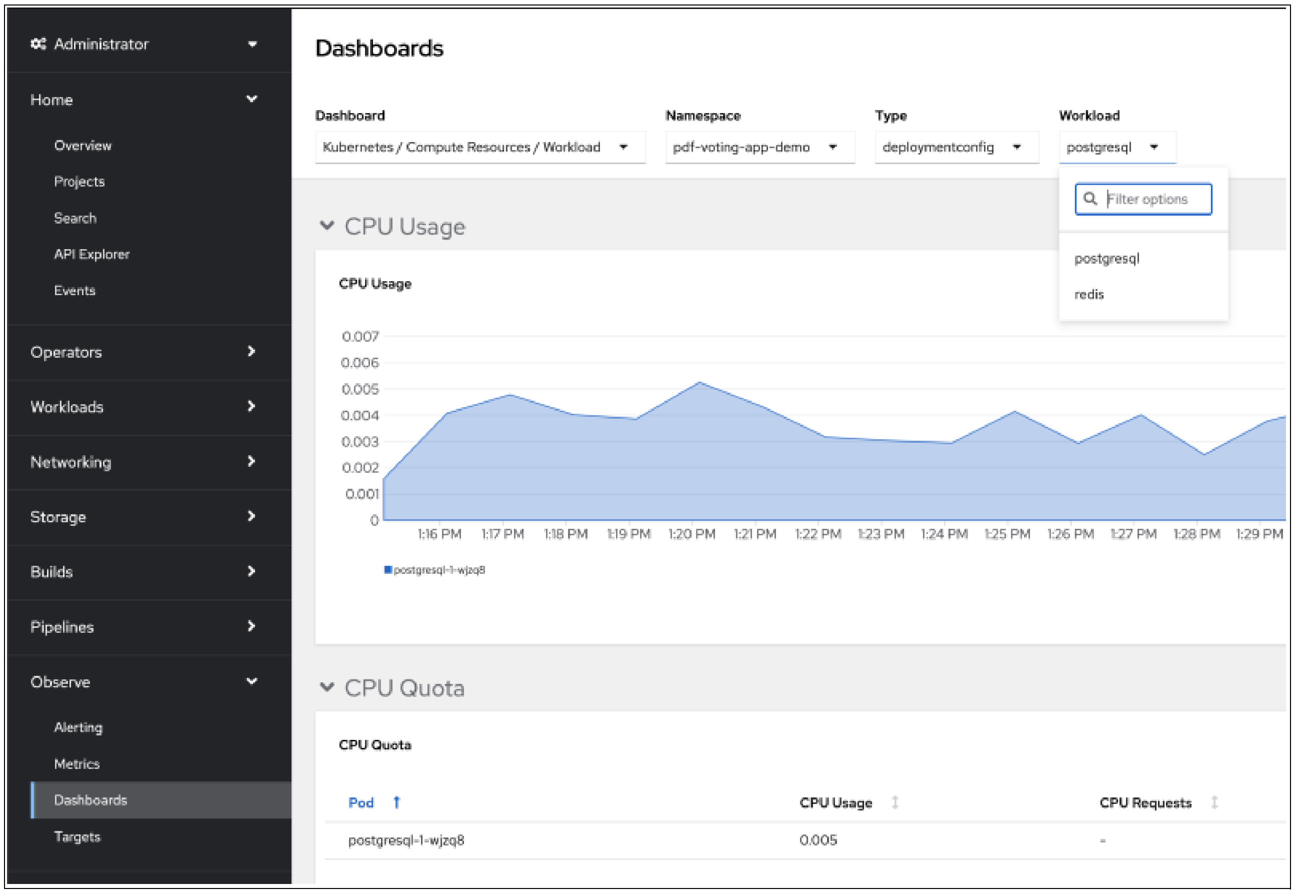
Dashboard: Kubernetes / Compute Resources / Workload
Namespace: pdf-voting-app-demo
Type: deployment
Workload: voting-app-nodejs

CPU Quota

| Pod | CPU Usage |
|---|---|
| voting-app-nodejs-5748d6bf84-ktmfb | 0.001 |

*Figure 6-11 Red Hat OpenShift dashboard namespace, deployment type, and workload monitoring*

Figure 6-12 on page 121 shows the same dashboard, but with a different deployment Type (this collection of microservices uses two types of deployments). Both Figure 6-10 on page 119 and Figure 6-12 on page 121 provide mode details about resource consumption per workload inside that specific namespace.



Dashboard: Kubernetes / Compute Resources / Workload
Namespace: pdf-voting-app-demo
Type: deploymentconfig
Workload: postgresql

| Pod | CPU Usage | CPU Requests |
|---|---|---|
| postgresql-1-wjzq8 | 0.005 | - |

*Figure 6-12 Red Hat OpenShift dashboard different deployment types example*

## 6.2.5 Using the oc client tool to monitor resources

The following set of examples describe how to use the command-line interface (CLI) to obtain the same information by using the **oc** client tool. For more information about this tool, see Getting started with the Red Hat OpenShift CLI.

After the **oc** client tool is installed on your client workstation, you can work remotely with your cluster.

Example 6-1 demonstrates how to monitor specific namespaces by using the **oc** client tool.

*Example 6-1 The 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 The 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 the resources that are used by nodes on the Red Hat OpenShift cluster with the **oc** client tool, as shown in Example 6-3.

*Example 6-3 Monitoring resources that are 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%
```

Our simplified overview of the Red Hat OpenShift Container Platform built-in monitoring stack is concluded. For more information, see About OpenShift Container Platform monitoring.

## 6.2.6 Using Resource Measurement Facility to monitor z/OS resources for Red Hat OpenShift Container Platform

In this 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 IBM z/OS Container Extensions (zCX) to run Red Hat OpenShift components and workloads, there are specific z/OS WLM definitions to be made as part of the Red Hat OpenShift implementation, such as service classes. For more information, guidance, and implementation steps to configure WLM policies for zCX Foundation for Red Hat OpenShift, see Workload management configurations.

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 that velocity goals are being achieved and CPU consumption for general-purpose processors and IBM Z Integrated Information Processor (zIIP) offload engines are within the needed levels.

For more information about monitoring zCX started tasks with RMF, see 7.5, “Monitoring with RMF on zCX instance level” of the *Getting started with z/OS Container Extensions and Docker*, SG24-8457.

## 6.3 Observability on z/OS

In a hybrid cloud architecture, you must monitor applications and all their involved infrastructure components from a single tool. Monitoring refers to assessing, based on logs, events and traces; whether an application or its components are available; healthy, malfunctioning, and heavily used; and others. However, when applications involve integration between several different layers and services that might be running in various platforms or clouds, it might become challenging to understand whether 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 and in each of its processing phases over time is known as *observability*.

*Observability* 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.

Due to existing requirements of availability, there is a shift to AI-driven observability, which brings specific metrics with it:

- Mean Time To Detect (MTTD) describes how long it takes to detect incidents or issues.
- Mean Time To Prevention (MTTP) describes how long it takes to automatically prevent incidents or issues from negatively impacting application performance and users.
- Mean Time To Notify (MTTN) refers to the amount of time that it takes to raise alerts to relevant teams about incidents or issues.
- Mean Time To Repair (MTTR) is used to describe the amount of time that is 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 that is available in the market and made available to cover z/OS components is IBM Instana®. It is an enterprise observability solution that automatically makes your applications and services visible, which provides context to that observed information. You can take intelligent action based on that information. Instana monitors and analyzes your applications, services, infrastructure, web browsers, mobile applications, and more.

With Instana, you instantly know whether any of your customers are impacted by performance or stability issues in your applications within a few seconds. Instana provides a GUI to guide you to the root cause with 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.
- It 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, which is the IBM mainframe.
- It does not replace traditional z/OS tools, but operations teams can form an application point of view, and quickly determine and point to exactly where there is an issue, such as unavailability or slowdown.
- It smooths operational visibility from all application intersections, including application dependency maps for every application, their flows, and calls.

**Note:** At the time of writing, the Instana back-end on-premises installation is supported only on certain Linux distributions running on x86/64bit processors, as documented in IBM Instana Observability documentation. For more information about this topic, see the IBM Statement of Direction IBM Observability by Instana intends to provide support for IBM Z application environments.

This IBM Statement of Direction is subject to change to support on-premises installation of Instana back-end servers running on Linux on IBM Z. For more information, contact your IBM Sales representative.

## 6.4 Logging

Red Hat OpenShift Logging aggregates all the logs from a 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 you can use a Kibana web console to visualize log data.

Red Hat OpenShift Logging aggregates the following types of logs:

- Application: Container logs that are generated by user applications running in the cluster, except infrastructure container applications.
- Infrastructure: Logs that are 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 that are 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 application programming interface (API) server and the Red Hat OpenShift apiserver.

## 6.5 Metering

Metering is a deprecated feature since Red Hat OpenShift 4.7. Its function is included in the Red Hat OpenShift Container Platform and continues to be supported, but it will be removed in a future release of this product, so you should not use it for new deployments.

7

# Deploying production applications

This chapter builds on the modernization patterns that were introduced in Chapter 1, “Introduction” on page 1. We focus on the delivery of an application by 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 are provided with a sample application, implementation details, and implementation best practices.

This chapter covers the following topics:

- Production deployment strategies
- Exposing on-premises applications through a public cloud

## 7.1 Production deployment strategies

Production deployment strategies involve delivering an application from development to the production environment, where it is used by users. There are various factors that inform the choice of deployment strategy, which include application architecture, business requirements, size of the change, and impact to application users.

Here are some common deployment strategies:

- Re-create deployment

  This deployment stops the current version of an application and creates a new version. This deployment type requires a maintenance window, which leads to downtime for 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 that are deployed in production. The current version and the new version are both functional, and 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. Two releases in production mean it is costlier because more resources are required.

- Rolling deployment

  This deployment involves the incremental rollout 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 deployment is used to expose a new version to a small subset of users. Using weights, traffic is partitioned to multiple versions of an application that are running concurrently in production. For example, a stable release and canary release are deployed into production with 90% traffic going to the stable release and 10% going to the canary release. With this deployment, you can test features in production with minimal impact.

  We use the canary deployment strategy in this chapter.

- A/B Testing deployment

  This deployment is similar to the canary deployment strategy but has fine-grained controls on the subset of users that can access a new feature. It uses statistical evidence from observing and monitoring user engagement to decide about whether a new feature is viable. This deployment has full control of traffic distribution, but it is a more complex implementation.

## 7.2 Exposing on-premises applications through a public cloud

In the section, we prepare an on-premises application that is deployed to Red Hat OpenShift on IBM Z to accept traffic from the internet through IBM Cloud. You learn how to connect your on-premises application to IBM Cloud in a secure fashion. This task is a building block of the *canary deployment* that was described in 7.1, “Production deployment strategies” on page 126.

## 7.2.1 Prerequisites

Here are the required components for our application deployment:

- A Red Hat OpenShift cluster and registry that are provisioned. The developer has sufficient access permissions (roles) on the cluster.
- If the cluster is on-premises or a private network, the user is connected to the network through a virtual private network (VPN).
- Podman is installed on the developer machine.
- Client tools to access the cluster from a command-line interface (CLI) are installed on the developer machine. For more information about installation instructions, see Getting started with the Red Hat OpenShift CLI.
- The server certificate is installed on the developer machine to enable a secured connection. You must obtain a server certificate from your cluster administrator.
- Access to GitHub to access some 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 IBM Cloud.

## 7.2.2 Current architecture

In our lab environment scenario, we have an application that is named *Supply Chain App* that is 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 by 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.

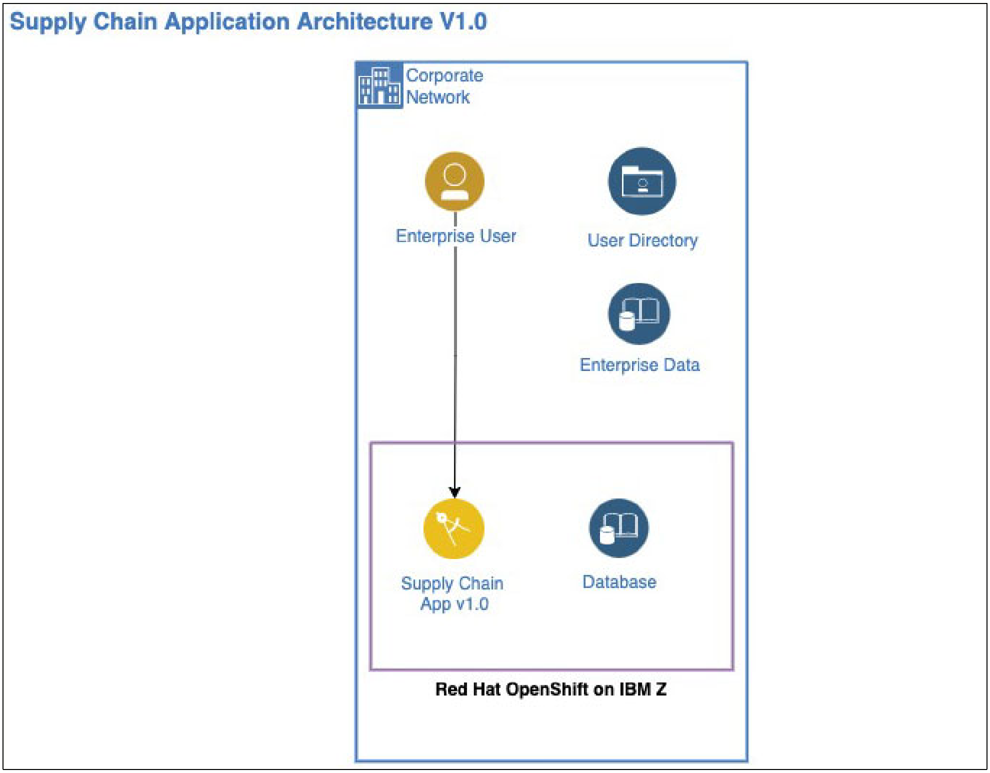


*Figure 7-1 Current architecture*

## 7.2.3 Target architecture

The business wants to allow access through the internet and enable business partners to access the application in a secure fashion. We use IBM Cloud to deploy the "front-door" components and perform network integration through a site-to-site VPN gateway and by using Internet Protocol Security (IPsec). An overview of our architecture is shown in Figure 7-2.

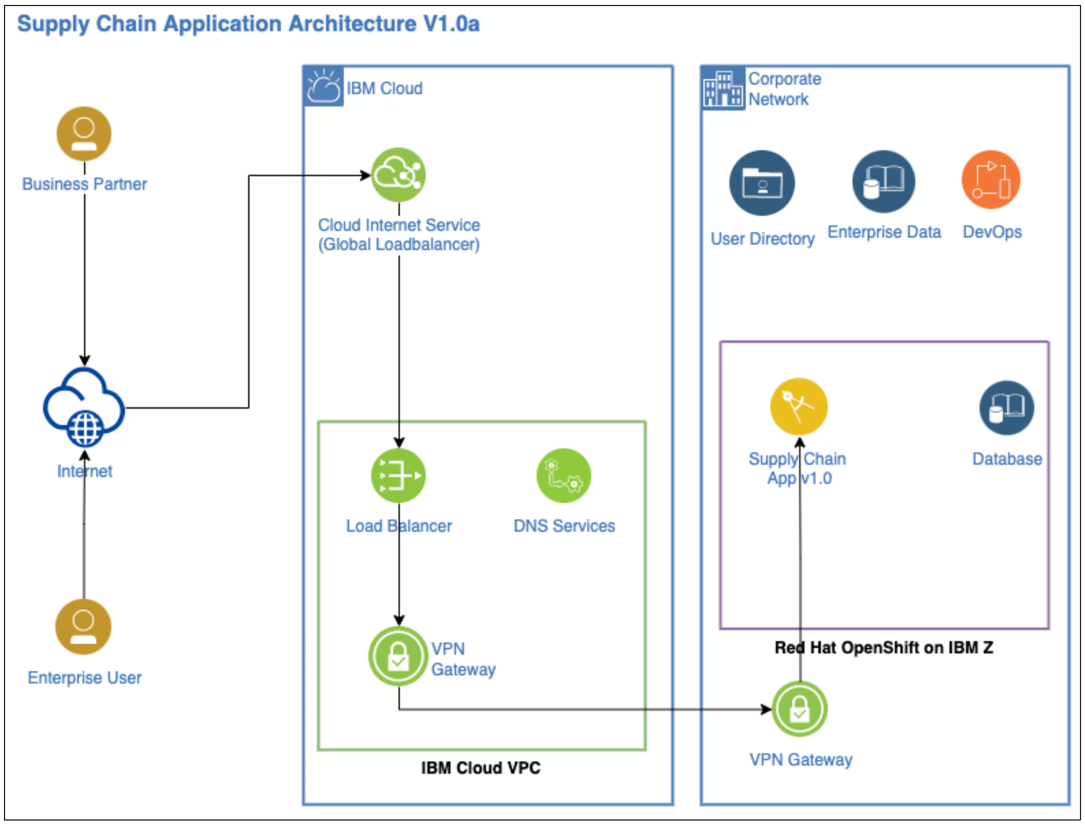


*Figure 7-2 Target architecture*

### Benefits

Some benefits that come from this target architecture are the following ones:

- Seamless application deployment. The user has no clue where the infrastructure is.
- Easy to implement multiple points of failures.
- Leveraging 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 data center and various IBM Cloud locations.

## 7.2.4 Current architecture implementation

We deploy a simple application that is written in Golang to Red Hat OpenShift on IBM Z. The application listens on port 8080 and provides a health endpoint.

### Building and deploying the Supply Chain App 1.0 from source code

Although the Red Hat OpenShift source to image (S2I) process was introduced in "Deployment instructions" on page 49, here we manually build the application by using Podman and uploading the image to the Red Hat OpenShift Container registry. For more information about the registry, see Registry.

To build and deploy an application, complete the following steps:

1. Download the source code from Git by using the command that is shown in Example 7-1.

*Example 7-1 Command to download the 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 Using 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 that is shown in Example 7-3.

*Example 7-3 Deploying the application*

```
oc create-project scapp
oc deploy bla
```

4. Check the application by using the command that is shown in Example 7-4.

*Example 7-4 Checking the application*

```
oc describe deployment
```

5. Test the application by using the command that is shown in Example 7-5.

*Example 7-5 Command to test the application*

```
curl app url
```

   This command returns a response with the application version.

We built and deployed a simple application. This application is now deployed.

## 7.2.5 Target architecture implementation

This section involves the creation of the components that are representative of the architecture that is shown in Figure 7-1 on page 127. Most of the steps are conducted on the IBM Cloud web GUI. Here are the high-level steps:

1. Creating and configuring a load balancer in IBM Cloud
2. Creating the IBM Cloud internet service to manage the domain
3. Creating an IBM Secrets Manager to order public TLS certificates
4. Configuring IBM Cloud Load Balancer with TLS certificates

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 the applications are reachable on-premises at `https://<app>.app.rdbkvmocp.pbm.ihost.com`.

We named our external domain `zhybridcloud.centers.ihost.com`. The application that we deploy will be reachable in our lab environment at `https://scapp.zhybridcloud.centers.ihost.com` on the internet.

### Creating and configuring a load balancer in IBM Cloud

In this step, we create an application load balancer to receive traffic from the internet:

1. Create a resource group.

   Creating a resource group helps with grouping all the resources that are deployed for this example. You can accomplish this task by using the IBM Cloud command-line interface (CLI) or its GUI.

   For our example, we went to IBM Cloud Resource Groups to create a resource group that is named `redbook`.

2. Go to the VPC Infrastructure page on IBM Cloud.

   On the IBM Cloud dashboard, click the navigation menu at the upper left, and then click **VPC Infrastructure** (Figure 7-3), or go to IBM Cloud VPC.

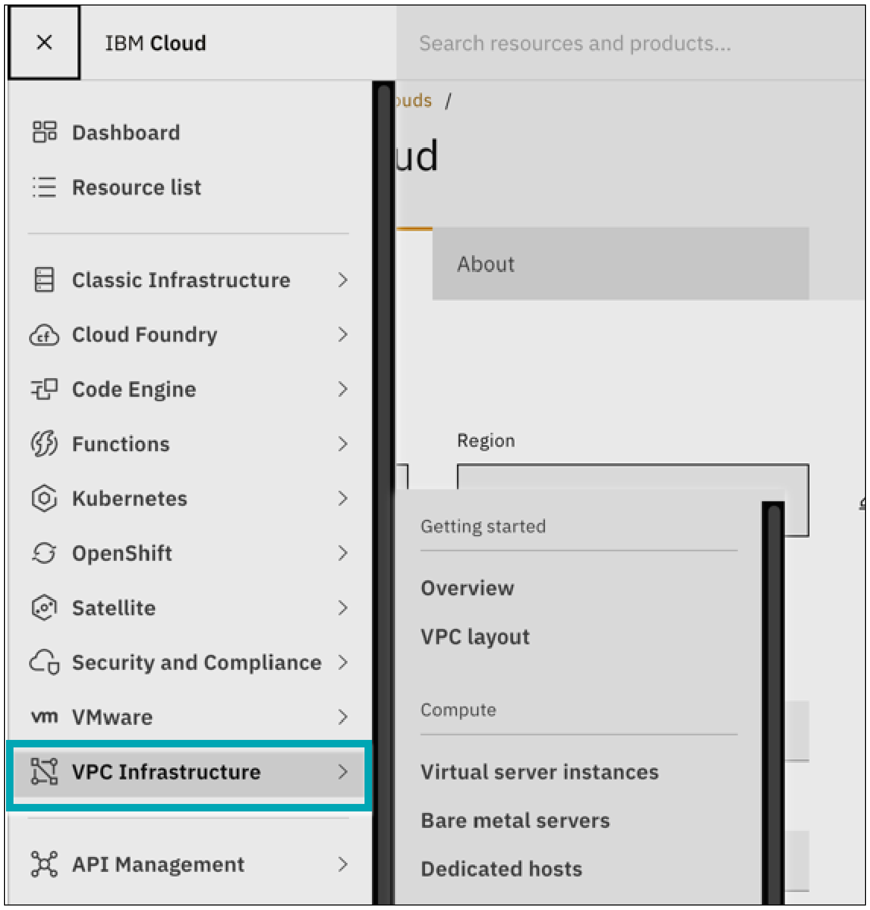
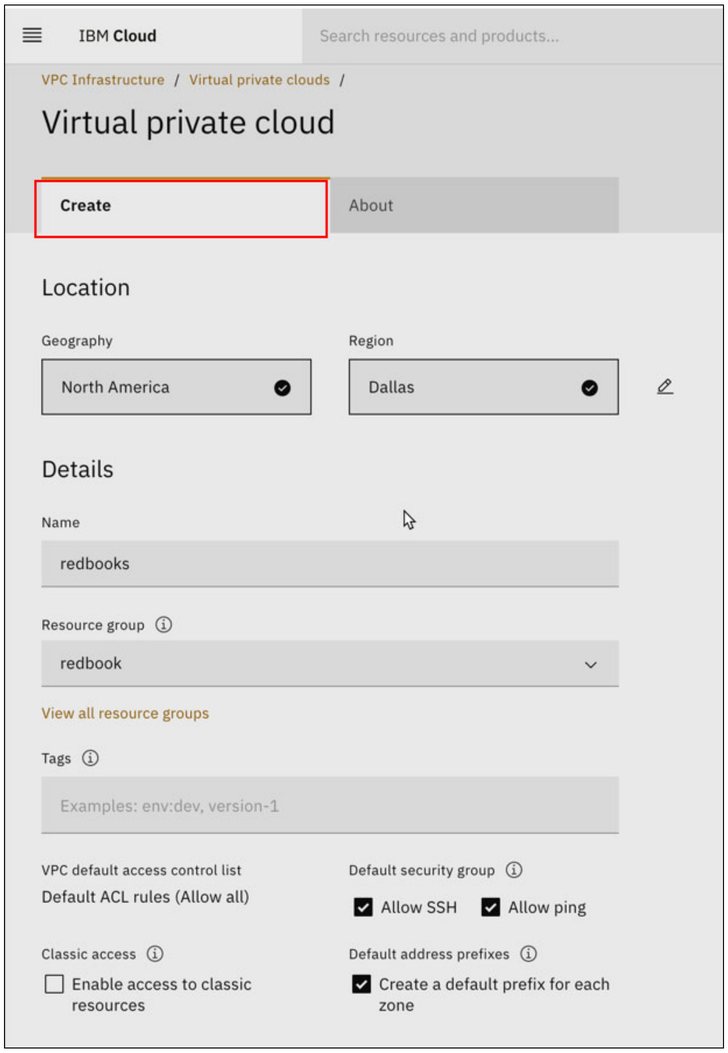


*Figure 7-3 VPC Infrastructure navigation*

3. Create a VPC.

   In the VPC Infrastructure window, scroll down the left pane to the Network section, click **VPC**, and then click **Create**. Figure 7-4 shows the window that opens.

   Alternatively, you can click **Catalog** at the upper right menu and search for `VPC`. We are deploying in the North America Geography and Dallas region. We input `redbook` as the Name and select `redbook` as the resource group. No changes are required for the other options.


IBM Cloud
Search resources and products...
VPC Infrastructure / Virtual private clouds /
Virtual private cloud
Create
About
Location

Geography: North America

Region: Dallas

Name: redbooks

Resource group: redbook

Tags: Examples: env:dev, version-1

VPC default access control list: Default ACL rules (Allow all)

Default security group: [✓ Allow SSH] [✓ Allow ping]

Classic access: [ ] Enable access to classic resources

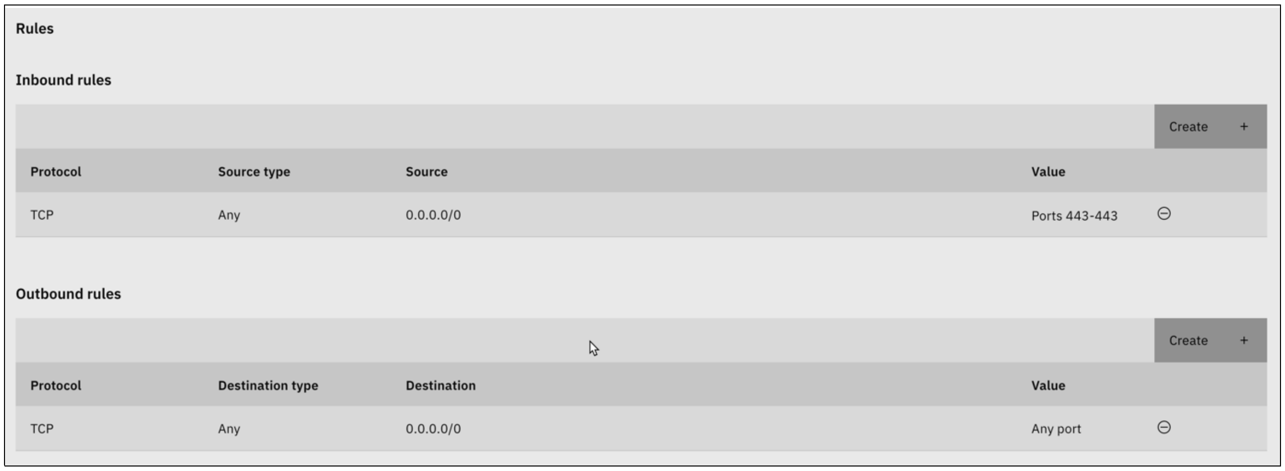
Default address prefixes: [✓ Create a default prefix for each zone]

*Figure 7-4 Creating a VPC*

   Click **Create virtual private cloud**.

4. Create the security group for the load balancer.

   This step is used to configure the traffic flow roles for the load balancer instance that we create. 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 on page 133.


Rules

Inbound rules

| Protocol | Source type | Source | Value |
|---|---|---|---|
| TCP | Any | 0.0.0.0/0 | Ports 443-443 |

Outbound rules

| Protocol | Destination type | Destination | Value |
|---|---|---|---|
| TCP | Any | 0.0.0.0/0 | Any port |

*Figure 7-5 Security group rules for the load balancer*

We entered the name `redbook-alb-sg`. Ensure that Location is the same as the VPC that you created and the resource group is the same as the one that you created. In our case, it is `redbook`. The VPC we selected is also `redbook`. Click **Create security group**.

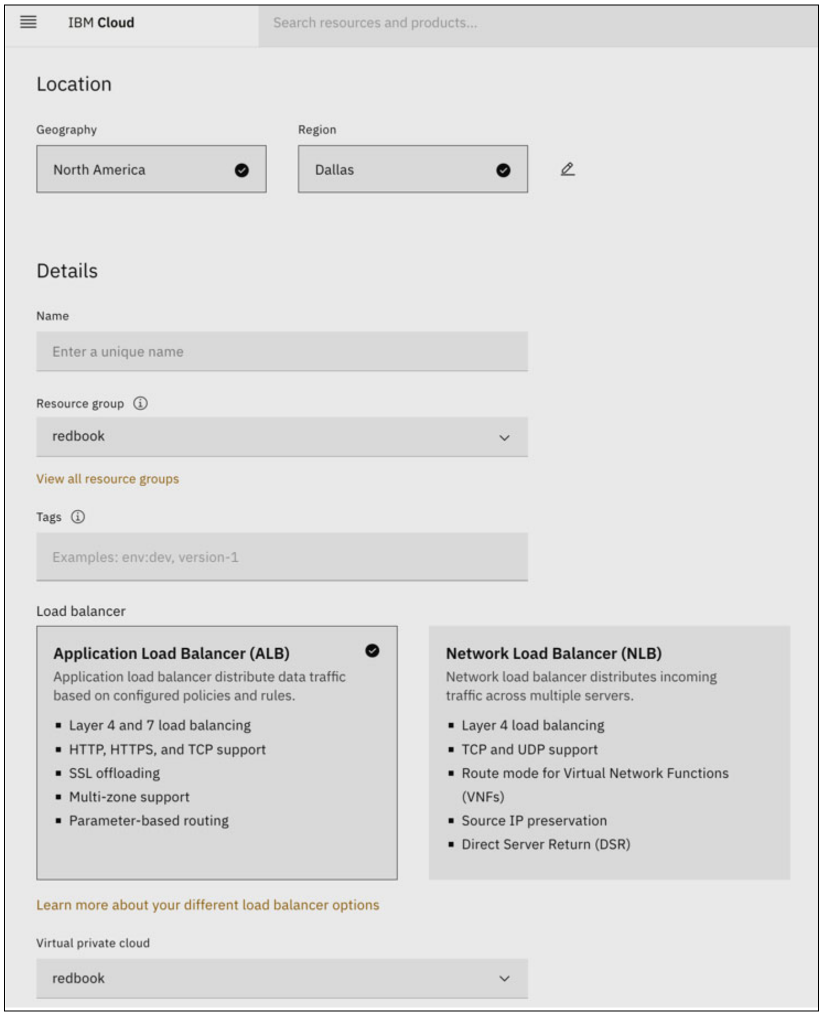
**Note:** In a real production environment, the outgoing traffic should be limited to only permitted protocols, ports, and IP addresses.

5. Create the load balancer.

   On the VPC Infrastructure window, scroll down the left pane to the Network section, click **Load balancers**, and then click **Create**. Alternatively, go to IBM Cloud Load Balancer, and then click **Create**.

   Enter `zhybridcloud-alb` as the name of the load balancer. Ensure that the Geography/Region is the same as in step 3 on page 132, the resource group is `redbook`, and the **Application Load Balancer** is selected. Select **Public** for the Type. Select all the subnets for Subnets. No back-end pool or listeners must be configured currently. For security groups, clear the default that is highlighted as VPC default, and select `redbook-alb-sg`.

Figure 7-6 shows the window where this information in entered.



Location

Geography: North America

Region: Dallas

Details

Name: Enter a unique name

Resource group: redbook

View all resource groups

Tags: Examples: env:dev, version-1

Load balancer

**Application Load Balancer (ALB)**
Application load balancer distribute data traffic based on configured policies and rules.
- Layer 4 and 7 load balancing
- HTTP, HTTPS, and TCP support
- SSL offloading
- Multi-zone support
- Parameter-based routing

**Network Load Balancer (NLB)**
Network load balancer distributes incoming traffic across multiple servers.
- Layer 4 load balancing
- TCP and UDP support
- Route mode for Virtual Network Functions (VNFs)
- Source IP preservation
- Direct Server Return (DSR)

Learn more about your different load balancer options

Virtual private cloud: redbook

*Figure 7-6 Creating a load balancer*

6. Click **Create load balancer**. Verify that the load balancer was properly created. You should have two public IP addresses that are allocated to the load balancer. Note the IP addresses because you reference them again.

## Creating the IBM Cloud internet service to manage the domain

In the next configuration tasks, we will be completing 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 the HTTPS front-end listener on the load balancer with a TLS certificate.

Complete the following steps:

1. Create an IBM Cloud Internet Services Instance.

   On the IBM Cloud dashboard, click **Catalog** at the upper right and search for `Internet Services`. Click the **Internet Services by IBM** tile. Select a pricing plan, enter `cis-redbook` as the service name, and select resource group `redbook`. Read the license terms and click **Create** if you agree.

   **Note:** At the time of writing, there is a no-charge 30-day trial plan that you can use for your test implementation.

2. Configure the IBM Cloud internet service.

   After you complete step 1, you should be redirected to the internet service instance. If that action does not occur, you can go to your resource list by clicking the upper-right IBM Cloud menu, clicking **Resource List**, and expanding the **Services and Software** section. Alternatively, you can go directly to IBM Cloud Resources.

   Add a domain by clicking **Add a domain**. Enter a domain name or subdomain that can be delegated, skip importing the DNS records, and go 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 records with the following addresses:

   – `ns001.name.cloud.ibm.com`

   – `ns096.name.cloud.ibm.com`

   After the configuration successfully completes, your domain should be active on the overview page. We return to this window in step 3 to create a DNS entry for a subdomain that is 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 window for `cis-redbook`, select **Reliability** → **DNS**. Scroll down to the DNS Records section, and click **Add**.

   Select `A` as the type, `Automatic` for TTL, and `scapp` for the name, and then enter one of the public IP addresses of the load balancer that we created. Click **Add**.

   To test that this configuration works, run the **`nslookup scapp.<domain>`** command. It should resolve to one of the public addresses of the load balancer, as shown in Example 7-6.

*Example 7-6 Results from nslookup*

```
Non-authoritative answer:
Name:scapp.zhybridcloud.centers.ihost.com
Address: 52.116.129.47
```

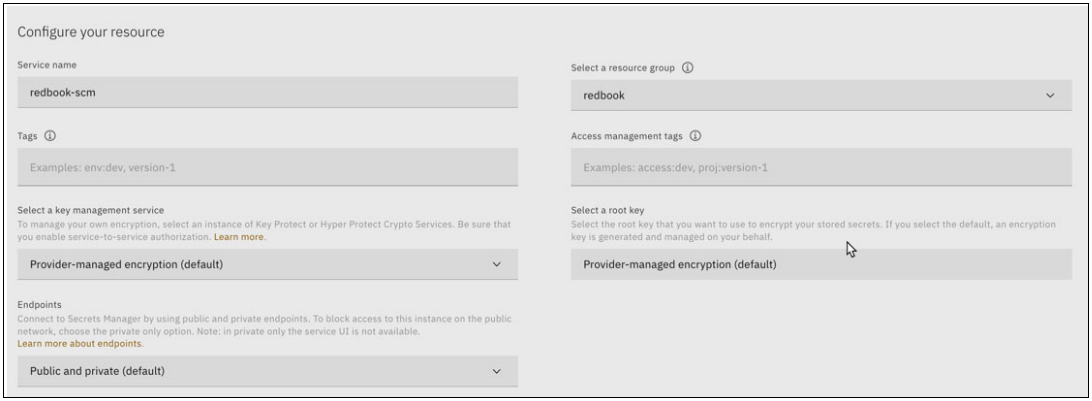
## Creating an IBM Secrets Manager to order public TLS certificates

To create your IBM Secrets Manager, complete the following steps:

1. Create a Secrets Manager instance.

   Go to the catalog and search for `Secrets Manager`. Click the **Secrets Manager by IBM** tile. Select the location (in our example, we selected **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 Figure 7-7.


Configure your resource

Service name: redbook-scm

Select a resource group: redbook

Tags: Examples: env:dev, version-1

Access management tags: Examples: access:dev, proj:version-1

Select a key management service
To manage your own encryption, select an instance of Key Protect or Hyper Protect Crypto Services. Be sure that you enable service-to-service authorization. Learn more.
Provider-managed encryption (default)

Select a root key
Select the root key that you want to use to encrypt your stored secrets. If you select the default, an encryption key is generated and managed on your behalf.
Provider-managed encryption (default)

Endpoints
Connect to Secrets Manager by using public and private endpoints. To block access to this instance on the public network, choose the private only option. Note: in private only the service UI is not available.
Learn more about endpoints.
Public and private (default)

*Figure 7-7 IBM Cloud Secrets Manager configuration*

   Read the license terms. If you agree, select the checkbox and click **Create**.

2. Create Identity Access & Management (IAM) access between Cloud Internet Services and Secrets Manager by using the instructions at IBM Cloud.
3. Configure the DNS provider in the Secrets Engine for Public Certificates:
   a. In the Secrets Manager UI, select **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. Click **Next**.

   If you have the IAM authorization properly configure, you should see the `redbook-cis` instance in the drop-down list in the **Authorization** tab. Select `redbook-cis` and click **Add**.
4. Create an Automatic Certificate Management Environment (ACME) account for use with the Let's Encrypt Public certificate authority (CA).

   With an account with the ACME protocol, you can install a certificate management agent on your web server. To create your account, go to GitHub. You need your account credentials for step 5.
5. Configure the CA in the Secrets Engine for Public Certificates by completing the following steps:
   a. In the Secrets Manager UI, select **Secrets engine** → **Public Certificates**.
   b. Click **Add** in the Certificate Authorities section, enter `redbook-letsencrypt` for Name, and select **Let's Encrypt** for Certificate Authority. Click **Next**.
   c. Click the **Enter value** tab, and then copy and paste your private key from the ACME account that you created in step 4.
6. Request s TLS certificate for the subdomain. In our example, we request a TLS certificate for `scapp.zhybridcloud.centers.ihost.com`.

### Configuring IBM Cloud Load Balancer with TLS certificates

Create IAM access between the load balancer and the Secrets Manager (for more information, see Managing IAM access for VPC Infrastructure Services). To access the TLS certificates from the load balancer, you must have IAM access (for more information, see Configuring the IAM credentials engine).

A

# Voting app changes to support an IBM Db2 database

In 2.3, “Sample application architecture” on page 16, we deployed an open-source, lightweight, and microservices-based application that is called the “Voting app”. You can find the public repository with the application source code at GitHub.

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 that we made to the Voting app to support IBM Db2 by noting them as follows:

```
---text in red: Original source code
+++text in blue: Updated source code
```

- The `voting-app/worker-python/app.py` program, as shown in Example A-1.

*Example: A-1 The voting-app/worker-python/app.py program*

```
#!/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()
```

- The `voting-app/result/server.js` program, as shown in Example A-2.

*Example: A-2 The voting-app/result/server.js program*

```
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 that is 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` | Main repository. |
| `kafka-demo-consumer` | A sample application that is a proof of concept for the proposed architecture in "Proposed architecture: Phase 2" on page 46. |
| `kafka-demo-producer` | A sample application that is a proof of concept for the proposed architecture in "Proposed architecture: Phase 2" on page 46. |

### Additional requirements

The web material requires the following system requirements to build and deploy this code on any machine:

- Java: An object-oriented programming language.
- Maven: A build automation tool that is used primarily for Java projects.

# Abbreviations and acronyms

| | |
|---|---|
| **ACH** | Automated Clearing House |
| **ACL** | access control list |
| **ACME** | Automatic Certificate Management Environment |
| **AI** | artificial intelligence |
| **API** | application programming interface |
| **BAU** | business as usual |
| **BIAN** | Banking Industry Architecture Network |
| **BPM** | business process management |
| **BYOL** | bring your own language |
| **CA** | certificate authority |
| **CD** | continuous deployment |
| **CI** | continuous integration |
| **CI/CD** | continuous integration and continuous deployment |
| **CLI** | command-line interface |
| **CQRS** | Command Query Response Separation |
| **DDA** | demand deposit |
| **ELT** | extract, load, and transform |
| **ETL** | extract, transform, and load |
| **FaaS** | function-as-a-service |
| **HTAP** | hybrid transaction/analytical processing |
| **IaaS** | infrastructure-as-a-service |
| **IAM** | Identity Access & Management |
| **IBM** | International Business Machines Corporation |
| **IBM DBB** | IBM Dependency Based Build |
| **IBM UCD** | IBM UrbanCode Deploy |
| **IDE** | integrated development environment |
| **IDz** | IBM Developer for z/OS |
| **IPsec** | Internet Protocol Security |
| **ISA** | Instruction Set Architecture |
| **JDBC** | Java Database Connectivity |
| **JES** | Job Entry Subsystem |
| **LSP** | Language Server Protocol |
| **ML** | machine learning |
| **MTTD** | Mean Time To Detect |
| **MTTN** | Mean Time To Notify |
| **MTTP** | Mean Time To Prevention |
| **MTTR** | Mean Time To Repair |
| **ODBC** | Open Database Connectivity |
| **ODS** | operational data store |
| **PaaS** | platform-as-a-service |
| **PV** | persistent volume |
| **RMF** | Resource Measurement Facility |
| **ROI** | return-on-investment |
| **RSE** | Remote Systems Explorer |
| **S2I** | source to image |
| **SaaS** | software-as-a-service |
| **SCM** | software configuration management |
| **SDK** | software development kit |
| **SLA** | service-level agreement |
| **SOA** | service-oriented architecture |
| **SOR** | system of record |
| **TDA** | timed deposit |
| **UX** | user experience |
| **VPC** | virtual private cloud |
| **VPN** | virtual private network |
| **VSI** | virtual server instance |
| **VTP** | Virtual Test Platform |
| **WaaS** | IBM Wazi as a Service |
| **WLM** | Workload Manager |
| **WSL** | Windows Subsystem for Linux |
| **z/OSMF** | z/OS Management Facility |
| **ZAT** | IBM zAcceleration Team |
| **zCX** | IBM z/OS Container Extensions |
| **zDIH** | IBM Z Digital Integration Hub |
| **zIIP** | IBM Z Integrated Information Processor |
| **ZOAU** | IBM Z Open Automation Utilities |
| **zUnit** | IBM z/OS Automated Unit Testing Framework |
| **ZVDT** | IBM Virtual Dev and Test for z/OS |

# Related publications

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

## IBM Redbooks

The following IBM Redbooks publications provide more information about the topics in this document. Some publications that are referenced in this list might be available in softcopy only.

- *Accelerate Mainframe Application Modernization with Hybrid Cloud*, REDP-5705
- *Getting started with z/OS Container Extensions and Docker*, SG24-8457
- *IBM Data Virtualization Manager for z/OS*, SG24-8514
- *Red Hat OpenShift on IBM Z Installation Guide*, REDP-5605
- *Why IBM Hybrid Cloud for Your Journey to the Cloud?*, REDP-5653

You can search for, view, download, or order these documents and other Redbooks, Redpapers, web docs, drafts, and additional materials, at the following website:

**ibm.com**/redbooks

## Other publications

These publications also are relevant as further information sources:

- Gamma, et al, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1994, ISBN
- Radcliffe, *Enterprise Bug Busting: From Testing through CI/CD to Deliver Business Results*, Accelerated Strategies Press, July 2021, ISBN 9781098381493

## Online resources

These websites also are relevant as further information sources:

- The Cloud Adoption Playbook

  https://www.ibm.com/cloud/architecture/adoption/the-cloud-adoption-playbook/
- *DevOps from APIs to IBM Z For Dummies*

  http://www.recarta.co.uk/wp-content/uploads/2017/05/DevOpsforDummies-ilovepdf-compressed.pdf
- Git repository for `kafka-demo-consume`

  https://github.com/IBMRedbooks/SG248532-zos-hybrid-cloud-examples/tree/main/Chapter04/kafka-demo-consumer

- Git repository for `kafka-demo-producer`

  https://github.com/IBMRedbooks/SG248532-zos-hybrid-cloud-examples/tree/main/Chapter04/kafka-demo-producer
- IBM Wazi as a Service: Bringing your own image with IBM Wazi Image Builder

  https://www.ibm.com/docs/en/wazi-aas/1.0.0?topic=bringing-your-own-image-wazi-image-builder

## Help from IBM

IBM Support and downloads

**ibm.com**/support

IBM Global Services

**ibm.com**/services

Redbooks

Mainframe Application Modernization Patterns for Hybrid Cloud

(0.2"spine)
0.17"<->0.473"
90<->249 pages

SG24-8532-00

ISBN 0738461016