International Journal on Science and Technology (IJSAT)

E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Java on IBM z/OS: Deployment Models and Technical Considerations

Chandra mouli Yalamanchili

chandu85@gmail.com

Abstract

Java on z/OS has matured into a robust and strategically valuable platform for running enterprise-grade applications and modernizing legacy applications to enable seamless integration with the surrounding applications built using modern technologies.

This paper explores the technical foundations and practical deployment models for Java on z/OS, including standalone batch execution, COBOL interoperability, CICS integration, and modern service-oriented architectures using z/OS Connect and Liberty. This paper also outlines hybrid execution options using Linux-based technologies on IBM Z, such as zVM, zKVM, zCX, and zLinux.

This paper also provides brief comparisons with distributed and cloud platforms and performance and cost efficiency metrics, highlighting Java's strengths on z/OS, such as zIIP offloading and co-located data processing. Real-world configuration examples, coding practices, and architectural insights are included to help developers and architects maximize value on IBM Z.

Keywords: Java, z/OS, batch processing, JZOS, CICS, Liberty, z/OS Connect, zCX, z/VM, zKVM, zLinux, USS, Language Environment, SMF, RACF, DevOps, microservices, hybrid workloads, zIIP, mainframe modernization

Introduction

Mainframe systems have been the cornerstone of enterprise computing, and with growing modernization efforts, integrating and executing Java applications on z/OS has become inevitable. Java on z/OS enables organizations to combine existing COBOL, HLASM, and PL/I programs with modern Java services.

This paper explores the core features of Java on z/OS and outlines different options for running Java workloads on z/OS along with use case examples. This paper also presents the technical considerations for developing, deploying, and optimizing Java applications in this environment.

History of Java on z/OS

Java was first made available for IBM mainframe in 1997 [11], initially focusing on supporting emerging enterprise applications that required platform independence and a more flexible programming model. IBM's Java Virtual Machine (JVM) implementation for z/OS has evolved significantly since then. Early support focused on enabling Java in the USS (UNIX System Services) environment. Over time, deeper integration was achieved into several other areas on z/OS through JZOS, CICS JVM server, and Liberty Profile. The introduction of z/OS Connect, WebSphere Compute Grid, and zCX further expanded Java's utility, positioning it as a first-class citizen in mainframe environments. [1][2][3]

Benefits of Running Java on z/OS

Running Java directly on z/OS provides several strategic and operational benefits.

- Java workloads are eligible to run on IBM z Integrated Information Processors (zIIPs), allowing offload from general-purpose CPs to specialized, cost-efficient engines. This CPU offload provides significant software cost savings, especially for compute-intensive or long-running Java applications. [12]
- Java has matured as a trusted and stable programming model on the mainframe, with over two decades of integration and optimization across the z/OS ecosystem, enabling reliable and portable enterprise applications. [11]
- Java on IBM z/OS gets the same benefits of high availability and reliability by leveraging the robust architecture of IBM Z mainframe hardware.
- Direct access to mainframe-resident data sources such as VSAM, DB2, and IMS without requiring external data replication or transfer.
- Seamless integration with security frameworks such as RACF allows centralized authentication and authorization.
- Workload Management (WLM) capabilities and functionality are available to prioritize and control Java workload behavior alongside traditional workloads.
- Detailed logging, performance statistics, and operational audit trails are available through the System Monitoring Facility (SMF) integration.
- Physical proximity to core enterprise data and subsystems provides low-latency processing.
- Support for hybrid application models where Java services can be invoked by or invoke COBOL/PL/I logic, enabling gradual modernization strategies. [2][3]
- Improved interoperability with COBOL 6.4 allows easier Java and COBOL integration with enhanced data structure mapping and method invocation compatibility, improving hybrid processing efficiency. [10]
- Java Native Interface (JNI) enables the reuse of optimized z/OS-resident modules written in C or assembler.
- JNI support facilitates access to z/OS system-level functions and APIs that may not be exposed directly via standard Java classes. This integration allows building performance-sensitive components while still leveraging Java's flexibility. [9]

Foundational components for Java on z/OS

All Java workloads on z/OS fundamentally rely on core z/OS components such as UNIX System Services (USS) and the Language Environment (LE):

- **UNIX System Services (USS)** provides the POSIX-compliant environment that enables Java to function on z/OS. Whether launched via JCL, CICS, or Liberty, Java applications depend on USS to access the file system, shell execution, Java libraries, and runtime configuration artifacts. USS supports the storage of WAR, JAR, and configuration files and enables integration with DevOps tooling and Unix utilities. [1][2][3]

- **Language Environment (LE)** underpins all high-level language programs on z/OS, including Java, COBOL, and PL/I. LE facilitates inter-language calls, exception handling, and consistent runtime behavior. Java integrates with LE to enable seamless interoperability with COBOL and PL/I through Callable Services and to ensure that system-level functions (like I/O or error recovery) are uniformly managed.[1][2][3]

Together, USS and LE ensure that Java workloads can coexist and cooperate with traditional workloads on z/OS while providing the flexibility and portability of the Java platform.

1. Java in z/OS Batch

Java batch processing remains a basic use case for running Java workloads on IBM z/OS. It offers a familiar operational paradigm for mainframe practitioners while enabling Java's flexibility and modern capabilities. Organizations can leverage multiple batch implementation models to augment legacy JCL workflows or create new Java-based batch workloads. Java batch jobs can be scheduled, monitored, and tuned like traditional batch jobs and benefit from the integration with z/OS system features such as Workload Manager (WLM), System Management Facility (SMF), and Resource Access Control Facility (RACF) integrations.

Below are some of the benefits of running Java batch workloads on z/OS:

- Better resource isolation and JVM configuration flexibility, allowing fine-tuning of heap size, garbage collection policy, and processor optimization for each Java workload. [1][3]

- Ability to use UNIX file systems and DD allocation for mainframe file access, supporting traditional datasets and USS files via APIs like JZOS ZFile utility. [1][3]

- Easier monitoring through SMF records, job-level visibility through SDSF, and integration with workload management tools, enabling operational control and performance diagnostics similar to traditional batch jobs. [1][3]

- Integration with WLM allows fine-grained control of workload priorities, improving system responsiveness and meeting SLAs. [1][3]

- Co-location of business logic and data reduces I/O latency compared to off-platform execution models. [1][3]

- Java batch jobs can interoperate with COBOL and PL/I, enabling gradual modernization and reuse of legacy assets. [1][3]

1.1. Java workloads as standalone address space

Java workloads can be executed as standalone address spaces under z/OS or started tasks for long-running Java workloads, allowing developers to run jobs in a structured, schedulable, and observable environment.

Below are several common and supported methods for running batch workloads using Java:

- **BPXBATCH**: An IBM-provided shell or program execution utility is available as part of UNIX System Services (USS) on z/OS. It can be used to launch Java programs from a script. While simple to use, it doesn't offer full integration with z/OS batch infrastructure, but it is effective for launching Java via shell from a JCL wrapper. [15]
    - Example in JCL - "*EXEC PGM=BPXBATCH,PARM='SH program-name*'"
- **BPXATSL**: This utility enables launching shell scripts interactively or in batch environments. Like BPXBATCH, it offers shell-level access to Java applications; the difference compared to BPXATSL is that the Java application doesn't fork or spawn off a child task to work; the program is run in the local spawn. [15]
    - Example in JCL - "*EXEC PGM=BPXATSL,PARM='SH program-name*'"
- **JZOS Batch Launcher**: A high-performance and fully integrated option for running Java as part of z/OS batch jobs. It allows Java programs to be invoked through standard JCL procedures, supports access to DD statements, and provides system-level services like logging, SMF tracking, and DD I/O through Java APIs. The JZOS batch launcher is one of the most widely used and recommended options.[1][3]
- **WebSphere Compute Grid (WCG)**: IBM-provided framework founded on the WebSphere Application server designed to run Java batch workloads at scale. It provides features for job distribution, checkpoint/restart, and parallel execution. Although more complex, WCG is ideal for compute-intensive Java workloads and batch modernization. [3]
- **Liberty Profile**: Besides being a lightweight application server, Liberty has a built-in rich implementation for Java batch processing via JSR 352 (Java Batch API). It allows batch jobs to be defined using XML and annotations and run through the managed Liberty runtime. This feature makes it suitable for enterprise-grade batch workloads requiring REST APIs or modern programming models alongside batch execution.

- Example JCL using JZOS Batch Launcher:

```
//JAVAJOB  JOB (ACCT),'JZOS EXAMPLE',CLASS=A,MSGCLASS=X
//STEP1    EXEC PROC=JVMPRC80,
//            MAINCLASS='com.mycompany.FileReaderExample'
//STEPLIB  DD DSN=IBMJZOS.SAMPLE.LIBRARY,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//INPUT    DD DSN=MY.INPUT.SEQFILE,DISP=SHR
```

- Corresponding Java code using JZOS File Reader:

```
import com.ibm.jzos.ZFile;
import java.io.*;

public class FileReaderExample {
    public static void main(String[] args) throws IOException {
        String ddName = "INPUT";
ZFilezfile = new ZFile("dd:" + ddName, "rb,type=record,noseek");
BufferedReader reader = new BufferedReader(new
InputStreamReader(zfile.getInputStream()));
        String line;
        while ((line = reader.readLine()) != null) {
System.out.println(line);
        }
reader.close();
    }
}
```

1.2. Java workload in existing batch JCL step

Java can integrate directly into existing traditional batch job streams as a single job step or subroutine within an existing JCL flow. This option is commonly used when augmenting legacy workloads or gradually modernizing COBOL-centric applications.

- **Embedding Java steps in JCL**: Java classes or JARs can be executed using the IBM-provided JVMLDM86 program (JZOS batch launcher module for Java 8), allowing invocation of Java logic similar to traditional executables. This model supports parameter passing, output redirection, and SMF monitoring. [3]
- **Calling Java from COBOL using LE Callable Services**: COBOL programs can dynamically load and invoke Java methods via Language Environment (LE) interfaces. The language

environment support provides a powerful interoperability layer that enables extending legacy logic with Java features, such as XML/JSON parsing, encryption, or invoking external APIs. [3]

- **Considerations and best practices**:
    - Allocate sufficient memory for the Java heap using Xmx/Xms parameters.
    - Define the appropriate STEPLIB and CLASSPATH for locating Java libraries and classes.
    - Log output via STDOUT and STDERR to capture diagnostics in job logs.
- Example JCL step to run a Java JAR in batch:

```
//JAVAEXEC EXEC PGM=JVMLDM86,PARM='-jar mybatchapp.jar'
```

2. Java in CICS: OSGi and Liberty

CICS provides a powerful runtime environment for executing Java applications in a transactional context. Java in CICS supports modular OSGi-based applications and microservices deployed through the Liberty JVM server. This flexibility allows enterprises to run various Java workloads securely and efficiently within existing CICS regions.

- Deployment Options:
    - **CICS JVM Server (OSGi and Traditional)**: Java applications can be deployed in a dedicated JVM server configured in CICS. Using the JCICS API, applications can access CICS resources such as files, transient data (TD) queues, temporary storage (TS), and invoke other CICS programs. The OSGi model provides modularity, version control, and dynamic application lifecycle management. [2]
    - **CICS Liberty JVM Server**: Embeds a lightweight WebSphere Liberty server within CICS. It supports JEE features, RESTful services, and Java batch (JSR 352). This support for the enterprise-grade application server capabilities makes the Liberty JVM option suitable for modern microservices or hybrid workloads involving REST, JSON, and backend integration. [2][7]
- Use Cases:
    - Host REST APIs to expose CICS or DB2 data through JCICS API.
    - Invoke legacy programs from modern Java service layers through JCICS API.
    - Implementing Java batch applications using JSR 352 Java batch processing feature.
    - Make distributed vendor products available for legacy mainframe application integrations.

- Configuration Considerations:
  - Use a JVM profile to configure the JVM server's system properties, heap size, logging behavior, and class path.
  - Store the Java artifacts (JARs, WARs, configuration files) in the USS file system for easy access, version control, and integration with DevOps pipelines.
  - Monitor JVM behavior with Liberty's built-in tools and SMF 120 subtype 9 records for detailed performance and audit data.
- Best Practices:
  - Optimize heap size and thread pool settings to avoid high memory management load on the CICS regions.
  - Use JCICS API for all CICS interactions to ensure transactional consistency.
  - Separate the application logic and run configurations to provide better application maintenance.
- Example for Liberty:
  - Below is a sample server.xml file configuration for Liberty profile JVM server.

```
<server description="CICS Liberty Java Application Server">

<!-- Enable required features -->
<featureManager>
<feature>servlet-4.0</feature>
<feature>restfulWS-3.0</feature>
<feature>json-1.0</feature>
<feature>mpConfig-2.0</feature>
<feature>batch-1.0</feature>
</featureManager>

<!-- Deploy WAR application -->
<application location="cicsapp.war" type="war"/>

<!-- HTTP/HTTPS listener -->
<httpEndpoint host="*" httpPort="9080" httpsPort="9443"/>

<!-- Logging configuration -->
<logging traceSpecification="*=info" logDirectory="logs"/>

<!-- Configure batch work manager -->
<batchWorkManagermaxThreads="5" minThreads="1"/>
```

```
<!-- Define shared library and classloader -->
<classloadercommonLibraryRef="cicsSharedLib"/>

<library id="cicsSharedLib">
<filesetdir="/u/cicsusr/lib"/>
</library>

</server>
```

3. z/OS Connect

z/OS Connect Enterprise Edition (EE) provides a runtime that enables RESTful APIs to access traditional mainframe assets such as CICS transactions, IMS transactions, and DB2 stored procedures. z/OS Connect is a gateway that allows external, cloud-native, or distributed applications to interact with z/OS systems through modern API protocols.

- **Features**:
    - Provide API interface for existing CICS, IMS, or DB2 services with minimal changes to the back end, making it easier to integrate with the legacy platforms. [4]
    - Secure API access using SAF and RACF for authentication and authorization. [4]
- **Use Cases**:
    - Enable mobile and web applications to consume mainframe data through modern integrations like APIs. [4]
    - Integrate z/OS services into hybrid cloud applications using RESTful interfaces. [4]
    - Modernize legacy interfaces by encapsulating 3270 screens as well as the business logic behind REST APIs. [4]
- **Operational Benefits**:
    - Reduces coupling between systems by using industry-standard interfaces.
    - Accelerates innovation by simplifying integration with third-party platforms.
    - Scalable architecture as this solution uses the Liberty server as the foundation.

4. Other Options to Run Java Applications

In addition to running Java natively on z/OS, IBM Z hardware offers several alternative environments that support Java execution indirectly via Linux-based virtualization or containerization. These are not z/OS-native Java environments, but they provide high-performance, secure, and scalable runtime options for Java applications on the same mainframe infrastructure.

- **z/VM**:
  - A full-featured hypervisor that supports running thousands of Linux virtual machines (VMs). [5]
  - Commonly used for workload isolation, development and testing, and Linux application hosting.
  - Java applications can run on Linux distributions (e.g., RHEL, SUSE) deployed as guests under z/VM. [5]
- **zKVM**:
  - zKVM is a lightweight, kernel-based virtual machine hypervisor designed to run Linux on Z. [5]
  - It is suitable for cloud-native and containerized workloads and allows tight integration with open-source tools. [5]
- **zCX (z/OS Container Extensions)**:
  - Enables Docker-compatible Linux containers to run directly within z/OS. [6]
  - Supports deploying Java microservices and other Linux-based tools (e.g., Jenkins, Ansible) alongside traditional mainframe workloads.
  - Managed through z/OSMF and supports integration with DevOps automation frameworks. [6]
- **zLinux (Linux on IBM Z)**:
  - zLinux is the standard Linux distribution running natively on IBM Z hardware. [8]
  - Java applications can leverage enterprise-class scalability, RAS (reliability, availability, and serviceability), and integrated security. [8]
  - zLinux is often used to run cloud workloads and container platforms like Red Hat OpenShift. [8]
- **Benefits Across These Options**:
  - These options leverage IBM Z's high hardware reliability, processing capabilities, and centralized security.
  - These options will enable enterprises to adopt a hybrid cloud and step towards modernization journeys without leaving the platform.
  - These options are ideal for scenarios where portability, open-source tooling, and Linux-native execution are required.

Additional Considerations that apply to all z/OS Java options

- **CI/CD and DevOps Integration:** IBM provides different tools to automate the build and deployment of Java applications to USS file systems, including UrbanCode Deploy, z/OSMF workflows, and Ansible. These tools support repeatable builds, scheduled rollouts, and integration into enterprise DevOps pipelines without modifying existing job streams or CICS configurations. [13][6] In addition to these IBM tools for z/OS, standard DevOps tools like Jenkins, RunDeck, etc., can be used to build and deploy the Java applications to the USS environment.

- **Observability and Logging**: Java workloads on z/OS can produce SMF 120 subtype 9 records for performance monitoring. Liberty JVM also provides built-in logging, diagnostic trace, and application monitoring and alerting support. These features offer insight into thread usage, garbage collection, and response times. [14]

- **Security Considerations**: Java on z/OS operates within the highly secure RACF/SAF ecosystem, like other applications running in z/OS. Java security policies (java.policy) work with z/OS access controls, enabling secure execution environments for Java applications. Liberty additionally supports TLS, JWT-based authentication, and SAF mapping for securing REST endpoints and batch APIs. [14]

Java deployment strategy visualization and comparison

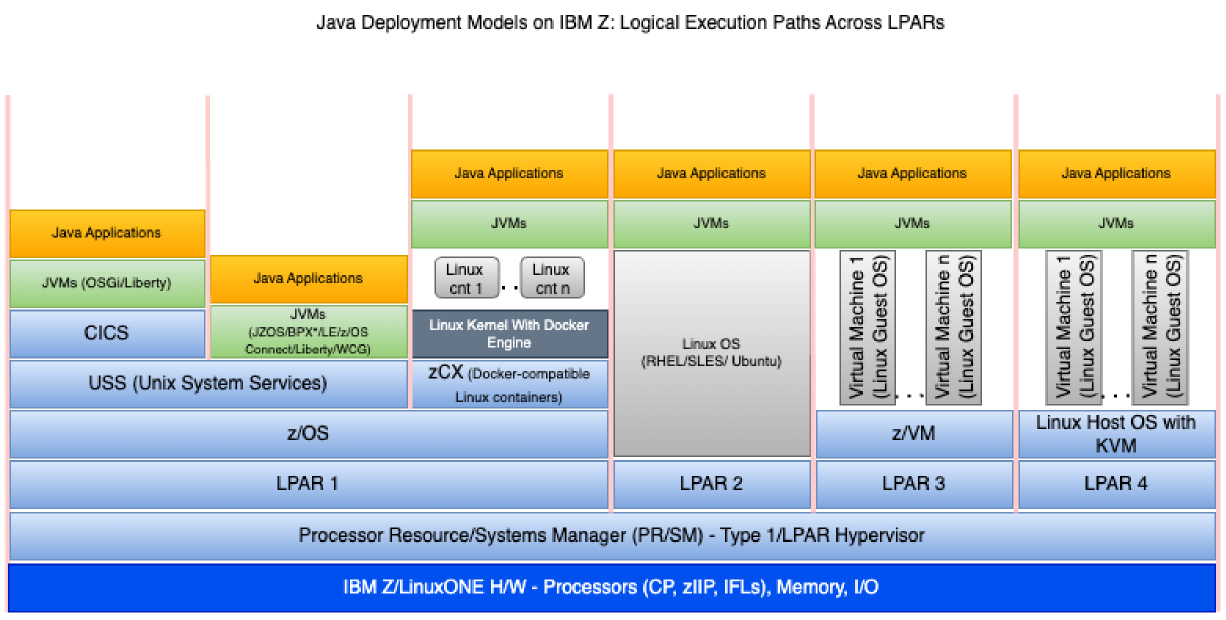
Java Deployment Models on IBM Z: Logical Execution Paths Across LPARs
Java Applications
JVMs (OSGi/Liberty)
CICS
Java Applications
JVMs (JZOS/BPX*/LE/z/OS Connect/Liberty/WCG)
USS (Unix System Services)
Java Applications
JVMs
Linux cnt 1
Linux cnt n
Linux Kernel With Docker Engine
zCX (Docker-compatible Linux containers)
z/OS
LPAR 1
Java Applications
JVMs
Linux OS (RHEL/SLES/ Ubuntu)
LPAR 2
Java Applications
JVMs
Virtual Machine 1 (Linux Guest OS)
Virtual Machine n (Linux Guest OS)
z/VM
LPAR 3
Java Applications
JVMs
Virtual Machine 1 (Linux Guest OS)
Virtual Machine n (Linux Guest OS)
Linux Host OS with KVM
LPAR 4
Processor Resource/Systems Manager (PR/SM) - Type 1/LPAR Hypervisor
IBM Z/LinuxONE H/W - Processors (CP, zIIP, IFLs), Memory, I/O

Figure 1: Java Deployment Options on IBM Z — Including native z/OS-based models (Batch, CICS, Liberty, z/OS Connect) and hybrid Linux-based models (zCX, z/VM, zLinux). USS and LE serve as core foundational layers for Java execution on z/OS. [1][2][3]

Deployment Strategy	**Environment**	**Use Cases**	**zIIP Eligible**	**Best suited use case**
Java Batch (JZOS, BPXBATCH)	z/OS	Standalone Java apps, batch processing	Yes	Batch modernization with mainframe data access
Java in Existing JCL Step	z/OS	Add Java logic to existing COBOL JCL flows	Yes	Gradual enhancement of legacy batch jobs
Java in CICS (OSGi/Liberty)	CICS on z/OS	Transactional APIs, batch, REST services	Yes	Modernization via REST/CICS integration
z/OS Connect EE	Liberty on z/OS	Expose mainframe data as REST APIs	Yes	API-driven integration with external systems
Liberty Batch (JSR 352)	Liberty on z/OS	Enterprise Java batch workloads	Yes	JSR 352 batch orchestration
zCX (Containers)	Linux Container on z/OS	Microservices, DevOps tools, integration	CP / zIIP / IFL (configurable)	Hybrid container workloads on z/OS
z/VM / zKVM	Linux VM on IBM Z	Modern Java apps, cloud-native, hybrid	IFL only (No zIIP)	Running full Linux Java stacks on IBM Z
zLinux	Native Linux on IBM Z	Open-source Java frameworks, enterprise microservices	IFL only (No zIIP)	Running enterprise Java apps on native Linux with Z scalability

Table 1: Comparative summary of Java deployment models on IBM Z, covering environment scope, zIIP eligibility, and best-fit scenarios.[1][2][3]

Comparison with Other Platforms and Performance Metrics

As enterprises evaluate their modernization strategies, it is important to understand how Java on z/OS compares to other equivalent environments. Performance and cost efficiency are major decision drivers that further distinguish z/OS in enterprise IT landscapes.

- **Java on z/OS vs. Java on Linux**:
    - z/OS offers stronger I/O throughput, better access to legacy data, and enterprise-grade reliability. Java workloads can take advantage of zIIP processors, reducing general-purpose CPU usage by up to 60% [12].
    - Linux provides faster CI/CD workflows and broader open-source tool support but often lacks the same level of proximity to critical mainframe datasets.
- **Java on z/OS vs. Distributed WebSphere**:
    - z/OS ensures co-location with CICS, DB2, and RACF for performance and security, enabling microsecond-level access to enterprise data [12].
    - WebSphere on distributed platforms excels in horizontal scaling and container orchestration but may incur higher latency and operational complexity.
- **Java on z/OS vs. Public Cloud**:
    - z/OS delivers predictable performance, zero-trust security, compliance, and advanced workload management (WLM) integration.
    - Cloud offers elasticity and rapid provisioning but may introduce latency, unpredictable cost structures, egress fees, and governance concerns.
- **CPU Efficiency and Instruction Throughput**:
    - IBM z16 can process up to 40 trillion daily instructions, optimized for high-throughput transactional workloads [12].
- **Latency and Data Accessibility**:
    - On z/OS, Java applications can access co-located data in CICS, DB2, or VSAM with microsecond-level latency, avoiding network latency overheads seen in distributed/cloud setups [12].
- **Software Licensing Advantages**:
    - zIIP-executed Java code is exempt from traditional z/OS software license costs, and WLM enables prioritization and cost control for non-zIIP workloads [12].
- **Strategic Fit**:
    - Java on z/OS is ideal for critical transactional systems with strict SLA, audit, or data residency requirements.
    - Hybrid approaches can combine z/OS for backend logic with cloud for UX and analytics layers.

- **Comparison Summary**:
  - **z/OS**: Reliable performance, security, co-location with legacy systems, and cost optimization via zIIP.
  - **Distributed/Linux**: Flexibility, CI/CD speed, and tool diversity but incurs network latency and egress costs.
  - **Cloud**: Elastic scale and rapid provisioning but may introduce unpredictable cost structures and data governance overheads.

Conclusion

Java on z/OS presents a uniquely powerful environment that unites the reliability of mainframe computing with the flexibility of modern Java development. Its deep integration with core z/OS services like CICS, DB2, RACF, and WLM makes it ideal for transactional, secure, high-throughput, and mission-critical applications.

The ability to run Java as part of batch jobs, in CICS regions, or as APIs via Liberty or z/OS Connect enables diverse deployment strategies while maintaining governance, auditability, or operational control that z/OS excels at.

Furthermore, the availability of cost-effective execution via zIIPs, high system throughput, and microsecond-latency access to mainframe data creates a strong value proposition compared to distributed or public cloud platforms. Other runtime options on IBM Z—such as z/VM, zKVM, zLinux, and zCX—allow Java applications to participate in hybrid and container-based ecosystems.

Java on z/OS is no longer just an option for modernization—it's a mature and strategic foundation for the next generation of mission-critical applications.

References

[1] IBM, "JZOS User's Guide and Reference", IBM Documentation, 2022. [Online]. Available: https://public.dhe.ibm.com/software/Java/Java80/JZOS/jzos_users_guide_v8.pdf. [Accessed: June 2023].

[2] IBM, "IBM CICS and the JVM Server: Developing and Deploying Java Applications", IBM Redbooks, July 2013. [Online]. Available: https://www.redbooks.ibm.com/abstracts/sg248038.html. [Accessed: June 2023].

[3] IBM, "Batch Modernization on z/OS", IBM Redbooks, July 2012. [Online]. Available: https://www.redbooks.ibm.com/abstracts/sg247779.html. [Accessed: June 2023].

[4] IBM, "IBM z Systems: The Heart of the Mobile and API Economy", IBM Redbooks, December 2015. [Online]. Available: https://www.redbooks.ibm.com/abstracts/redp5296.html. [Accessed: June 2023].

[5] IBM, "Set up Linux on IBM System z for Production", IBM Redbooks, November 2013. [Online]. Available: https://www.redbooks.ibm.com/abstracts/sg248137.html. [Accessed: June 2023].

[6] IBM, "IBM z/OS Container Extensions (zCX) Use Cases", IBM Redbooks, July 2021. [Online]. Available: https://www.redbooks.ibm.com/abstracts/sg248471.html. [Accessed: June 2023].

[7] IBM, "Liberty in IBM CICS: Deploying and Managing Java EE Applications", IBM Redbooks, January 2018. [Online]. Available: https://www.redbooks.ibm.com/abstracts/sg248418.html. [Accessed: June 2023].

[8] IBM, "Linux on IBM Z Documentation", IBM Documentation, February 2023. [Online]. Available: https://www.ibm.com/docs/en/linux-on-systems?topic=linux-z-linuxone. [Accessed: June 2023].

[9] IBM, "Java Native Interface (JNI)", IBM Basic z/OS Skills. [Online]. Available: https://www.ibm.com/docs/en/zos-basic-skills?topic=zos-java-native-interface-jni. [Accessed: June 2023].

[10] IBM, "What's new in COBOL for z/OS 6.4", IBM Documentation, August 2022. [Online]. Available: https://www.ibm.com/docs/en/SS6SG3_6.4.0/pdf/new.pdf. [Accessed: June 2023].

[11] B. S. Savenkoff, "Java at 25: Celebrating Milestones, Mainframe Interoperability and More", TechChannel, November 2022. [Online]. Available: https://techchannel.com/cobol/java-at-25-celebrating-milestones-mainframe-interoperability-and-more. [Accessed: June 2023].

[12] IBM, "IBM z16 Technical Introduction", IBM Redbooks, April 2023. [Online]. Available: https://www.redbooks.ibm.com/redbooks/pdfs/sg248950.pdf. [Accessed: June 2023].

[13] IBM, "UrbanCode Deploy and z/OSMF Integration", IBM Documentation, May 2023. [Online]. Available: https://www.ibm.com/docs/en/urbancode-deploy/7.3.1?topic=enabling-urbancode-deploy-zosmf. [Accessed: June 2023].

[14] IBM, "Monitoring Open Liberty with OpenTelemetry", IBM Documentation. [Online]. Available: https://www.ibm.com/docs/en/was-liberty/base?topic=tracing-monitoring-open-liberty-opentelemetry. [Accessed: June 2023].

[15] IBM, "z/OS V2R4 Documentation", IBM Documentation, January 2023. [Online]. Available: https://www.ibm.com/docs/en/zos/2.4.0. [Accessed: June 2023]