CICS TS V4.2 uses the IBM® 64-bit SDK for z/OS, Java Technology Edition, Version 6.0.1, in both pooled JVMs and JVM server configurations.

**NOTE:** There is no support for using a 31-bit JVM with CICS TS V4.2.

- Java 6.0.1 includes enhancements to JIT optimization and exploitation of z196 hardware through the use of new hardware instructions and out-of-order pipeline execution.

Using a 64-bit JVM provides significant virtual storage constraint relief by moving most of the memory used by both pooled JVMs and JVM server above the 64-bit bar as shown in the illustration.

This allows a single CICS TS V4.2 region to support more JVMs than earlier versions of CICS and each JVM can have much larger heaps. With CICS TS V4.2 it is quite possible to host traditional and Java workloads in a single CICS region in CICS TS V4.2 without encountering ‘short-on-storage’ conditions.

A potential disadvantage to using a 64-bit JVM is the associated increase in the size of Java objects needed to accommodate 64-bit object references. It is most likely that Java applications using 1-bit bar with CICS TS V4.1 and Java 6.0 will consume a larger amount of 64-bit memory if migrated to CICS TS V4.2 and Java 6.0. However, this increase can be moderated by using the “compressed reference” feature of the IBM JVM that reduces the size of 64-bit Java objects.

There are a number of GC policies supported by IBM Java 6.0.1. These GC policies can be specified in the WLM profile used by both pooled JVM and JVM server in both CICS TS V4.1 and V4.2 but the default policies and CICS-supplied settings differ between CICS TS V4.1 and V4.2.

- CICS TS V4.1 supports Java 6.0 which has a default gcpolicy of opthrun. This treats the Java heap as a single space. This policy is optimized to deliver high throughput but it can produce occasional long pauses. These long pauses are usually avoided by CICS TS V4.1, which forces garbage collection by starting a CJCS transaction in each pooled JVM when it detects that the Java heap reaches a heap occupancy exceeding CICS JAVA_HEAPOccupationThreshold.
- CICS TS V4.2 supports Java 6.0.1 which has a default gcpolicy of gencol. This treats the Java heap as separate garbage collection areas. This policy is optimized to deliver lower pause times while maintaining high throughput. The support of concurrent threads by a JVM server makes the forced GC mechanism used for a pooled JVM undesirable since it may lead to a forced GC when other CICS tasks are still running Java applications.

**NOTE:** You will not see any CICS GC transactions running in a JVM server.


**Which GC policy to use:** The default GC policy for both Java 6.0.1 and CICS TS V4.2 is gencol but you are at liberty to use any of the GC policies and tuning parameters that are detailed in the Java Diagnostics Guide.

- The default GC policy for the previous versions of Java and CICS was opthrun.
- There is no significant difference between opthrun and gencol is that opthrun treats the Java heap as a single space whereas gencol splits the heap into an area to allocate new objects (a nursery) and an area to keep long-lived objects (a tenure space).

**NOTE:** Garbage collection of a single space heap is normally a less frequent but longer lasting process than garbage collecting a smaller nursery area. gencol is better suited to Java workloads that create many small short-lived objects or large long-lived objects, typical behaviour of a transactional workload.

**Who pays to run a JVM in CICS:**

- The CPU cost of starting a pooled JVM is paid by the user transaction that runs the first Java application in a JVM which triggers its creation.
- The cost of starting a JVM server is paid by a CICS-supplied CJSR transaction which is invoked when the JVM server resource is installed.
- Each pooled JVM or JVM server runs in its own ENCLAVE with its own 31-bit and 64-bit memory areas.
- The 64-bit JVM server by both pooled JVM and JVM server in CICS TS V4.2 still has a small requirement for some 31-bit memory but not much of the storage used by the JVM, such as the Java heap, is located in 64-bit memory.
- The 31-bit JVM used in by CICS TS V4.1 needs to allocate memory from the same area of 31-bit memory used to allocate CICS EDAS.
- This competition for the limited 31-bit memory space in a CICS region restricts the number of JVMs that a CICS TS V4.1 region can support. This limit is effectively removed in CICS TS V4.2.

**NOTE:** Storage management in CICS TS V4.2 provide a measure of how many real memory frames have been used to support 64-bit memory objects by a CICS region.

- The main benefit to using JVM server with Java workload is that delivers similar throughput at similar CPU cost but uses significantly less memory than running multiple JVMs.

**NOTE:** JVM server uses the shared class cache provided by Java 6.0.1 directly.

- Unlike pooled JVMs there is no management interface (using SPI or CEMT commands for example) provided between CICS and the shared class cache.

**NOTE:** Class data sharing between JVMs is described in the IBM User Guide for Java 6 on z/OS at [http://publib.boulder.ibm.com/infocenter/javasdk/v6r1/](http://publib.boulder.ibm.com/infocenter/javasdk/v6r1/).

**The potential benefits of using a shared class cache are:**

- Reduced memory consumption if using multiple JVMs
- Faster JVM startup
- Faster application startup

**NOTE:** CICS Java Server uses the Open Services Gateway Initiative (OSGi) framework which is a module system and service platform for the Java programming language that implements a complete and dynamic component model.