



SOLUTION
READINESS

Enterprise Java Applications on VMware UNIX to Linux Migration Guide

© 2010 VMware, Inc. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. This product is covered by one or more patents listed at <http://www.vmware.com/download/patents.html>.

VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies.

VMware, Inc
3401 Hillview Ave
Palo Alto, CA 94304
www.vmware.com

Contents

1.	Introduction	5
1.1	Overview	5
1.2	Migration Prerequisites	5
2.	Key Unix-to-Linux Java Migration Considerations	6
2.1	Overview	6
2.2	Language Syntax and APIs Considerations	6
2.3	Java Compilation Considerations	7
2.4	Java Runtime Considerations	8
3.	FAQ: Migrating from Physical to Virtual.....	9
4.	References	11

1. Introduction

This *Enterprise Java Applications on VMware UNIX to Linux Migration Guide* focuses on key considerations for IT Architects who are in the process of migrating Java applications from UNIX to Linux as part of their VMware server consolidation project. Due to Java's cross-platform portability, Java applications are prime candidates to be migrated first with relatively low complexity and effort.

Constraints mentioned in this document are applicable for Java applications that migrate from one OS to another regardless of whether they run on physical or virtualized machines—they are not limits imposed by virtualization. The Java application code and runtime paradigm does not change when you migrate to a virtualized system. In fact, a Java application or application WAR file can be deployed as is from its physical machine form onto a virtual machine (VM) without change.

1.1 Overview

Because Java is a cross-platform, portable language, you can take custom enterprise Java applications and run them on any OS. The caveat is that the platform that the OS is running on must have a platform-specific JVM from the target vendor.

The following are key considerations when migrating custom enterprise Java applications from UNIX to Linux:

- Java language considerations
- Java compilation considerations
- Java runtime JVM considerations

An FAQ is provided to answer frequently asked questions related to migrating Java applications to VMware vSphere™.

1.2 Migration Prerequisites

Migrations of Enterprise Java applications are often part of an overall migration plan. It is assumed that you have satisfied the following prerequisites as part of an overall migration plan:

- A high-level plan for the migration process that considers all tiers of your enterprise, including both infrastructure and the software application layer.
- An application topology list of dependencies to highlight the network dependencies and any special port mapping.
- You have studied the inter-tier dependencies, such as load balancer to Web server communication.
- You may choose to migrate the Web server in Phase 1, migrate the Java application server in Phase 2, and migrate the database in Phase 3.

Because inter-tier communication is mostly via TCP/IP these dependencies are not tied to an operating system assuming the operating system has a compliant implementation. For example, a Java application server communicates with the database server via JDBC, which is an implementation that sits on top of TCP/IP and is not OS-dependent. This is equally true for Web servers communicating back with the Java application servers via HTTPs.

2. Key Unix-to-Linux Java Migration Considerations

2.1 Overview

Perform the following steps when migrating.

1. Review any potential file separator/path naming conventions as outlined in Section 2.2.
2. Consider the code compilation and deployment strategy as discussed in Section 2.3.
3. Review Section 2.4 for the outlined key runtime tuning configurations that potentially have to be migrated as they may not be supported across various operating systems.

The following sections provide examples. There may be other third-party, or other Java classes and libraries that you depend on. If so, you have to test for compliance at compilation and performance of runtime.

2.2 Language Syntax and APIs Considerations

The popularity of Java is mainly due to its having a commonly known syntax and a set of libraries that can be written once and target multiple platforms. Keep the following in mind.

In both UNIX and Linux the same file separator and path separator characters are used, so if you chose to not make any change to your code, this is acceptable and the code is still cross-portable. However, it is a good housekeeping practice to use the file separator within your Java application code. Instead of using characters such as the slash (/) in UNIX/Linux for path separators, use the following `java.io.File` available approach.

```
//my file separator is  
String fileSeparator = java.io.File.Separator
```

Equally applicable is the use of `File.pathSeparator`. Instead of using a character such as the colon (:) for path separators, use the following `java.io` available approach.

```
//my path separator is  
String pathSeparator = java.io.File.pathSeparator
```

If using `java.File.io.renameTo`, this method is OS-platform dependant, so your test script should always check the code as follows.

```
//always check your renameTo file code in unit test code  
if (oldfile.renameTo(newfile)) {  
    // test passed – rename succeeded  
}else{  
    // test failed – rename didn't succeed  
}
```

2.3 Java Compilation Considerations

Java applications can run on any OS due to the inherent capability of the language. However, as part of the migration you will probably want to recompile based on your new target Linux JVM in order to have a better overall compliance and conform to good deployment best practices.

Perhaps as part of the larger migration strategy you had to reconstruct your UNIX shell scripts. Also, the Java compilation and deployment scripts may have been included as part of the script reconstruction exercise. Instead of just reconstructing or reformatting your Java compilation scripts to the new Linux shell format (which is also a valid approach), it may be beneficial to use Apache Ant (<http://ant.apache.org/>) scripting to write a coherent Java application compilation and deployment script that is cross-platform compatible.

The following is an example of an Ant build.xml script used to compile Java source code. Refer to the highlighted Compile Ant target.

```
project name="MyProject" default="dist" basedir=".">
  <description>
    simple example build file
  </description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist" location="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="Compile" depends="init"
    description="compile the source " >
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile"
    description="generate the distribution" >
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean"
    description="clean up" >
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

- See the [Apache Ant Manual](#) for information about installing, using, and running Ant scripts.
- Also see the [javac – Java programming language compiler](#) manual page.

2.4 Java Runtime Considerations

Depending on which UNIX platform you are migrating from you will find that certain JVM and or OS runtime options are not available for you to configure. Though Java is cross-platform portable, because the target platform JVM is changing from one platform to another, it is recommended that you perform a load test exercise to establish the new benchmark for your Java application. When migrating from one OS to another, even though Java is portable, it is advisable that you load test and profile the characteristics of your application under the new technology stack.

2.4.1 Migrating from Solaris to Linux

Attribute	Impact
-XX:+UseLargePages	- On Solaris, the UseLargePages flag is selected by default. It needs to be turned on in Linux as using large pages improves performance. - When you migrate your Java application to Linux use these settings: - Set the -XX:+UseLargePages at the JVM level for Sun HotSpot. On IBM JVM it is -XIp, and JRockit -XXlargePages. - Set the huge pages at OS level. Refer to: How to enable Large pages in Windows and Linux.

2.4.2 Migrating from 32-bit JVM to 64-bit JVM

Often, migrating from UNIX to Linux may present an opportunity to also upgrade from a 32-bit to 64-bit JVM, but there are few considerations to keep in mind. It is not mandatory to upgrade to a 64-bit JVM to use a 64 bit OS, but it is a good practice if you are able to do so.

Attribute	Impact
Memory Size	- You need to load test your application, Refer to <i>Enterprise Java applications on VMware Design and Sizing Guidelines</i> for design and sizing guidelines. - Memory utilization of the same application within a 32-bit architecture will likely use more heap space as a result of 64-bit object representation. Therefore: - It is advisable to retune your heap space setting to new values that you determine by doing load testing. Values such -Xms and -Xmx need to be adjusted to accommodate the increase in internal object and types representation of 64-bit JVM. - You potentially need to re-adjust your GC settings. - The benefits of going to a 64-bit JVM and OS include the ability to use larger maximum heap settings. Many of the 32-bit limitations, such as the 1.54GB to 3.8GB heap limitation (depending which OS you were using) are removed and 64-bit heap sizes can be much larger. We see customers with 12GB heap sizes. Very large heap sizes come with the need to tune your GC. For tuning guidelines refer to your JVM vendor's documentation.

3. FAQ: Migrating from Physical to Virtual

With UNIX-based hardware I have very large machines running all of my Java applications. What should be my migration sizing strategy and what are the VMware vSphere maximums that I need to be aware of?

- One of the most important steps is to conduct a load test that will help you determine the ideal individual VM size and how many JVMs you can stack up (*Vertical Scalability*). Based on this *repeatable building block VM*, scale-out to determine what is best for your application traffic profile.
- It helps to know the VMware vSphere maximums. For information, see [Configuration Maximums: VMware vSphere 4.1](#).

The following table lists some important maximums.

vSphere Configuration	Maximum
Per VM	• 8 vCPUs • 255GB • 2TB of storage minus 512bytes
Per Host	• 512vCPUs • 320 VMs • 25 vCPUs per core
Per vCenter	• 1000 hosts • 10000 powered on VMs • 15000 registered VMs • 10 linked vCenter Servers • 3000 hosts in linked vCenter servers • 30000 powered on VMs in linked vCenter Servers • 50000 registered VMs in linked vCenter Servers • 100 concurrent vSphere Clients • 400 hosts per datacenter

What are new decisions that must be made for applications running on virtual that we did not have to make for native environments?

You have to determine the optimal size of the repeatable building block VM. Establish this by benchmarking, along with total scale-out factor. Determine how many concurrent users each single vCPU configuration of your application can handle and then extrapolate that to your production traffic to determine overall compute resource requirements for vCPU, memory, storage, and network. Having a symmetrical building block, for example every VM having the same number of vCPUs, helps keep load distribution even from your load balancer. The benchmarking tests help you determine how large a single VM should be (*vertical scalability*) and how many VMs you will need (*horizontal scalability*).

You need to pay special attention to scale-out factor and see up to what point it is linear within your application running on top of VMware. Enterprise Java applications are multi-tier and bottlenecks can occur at any point along the scale-out performance line and quickly cause non-linear results. The assumption of linear scalability may not always be true and it is essential to load test a pre-production replica (production to be) of your environment to accurately size for you traffic.

I have conducted extensive GC sizing and tuning for our current enterprise Java application running on physical. Do I have to adjust any sizing when moving this Java application to virtualized environment?

- No. All tuning that you perform for your Java application on physical is transferrable to your virtual environment. However, because virtualization projects are typically about driving a high consolidation ratio, it is advisable that you follow the guidelines in *Enterprise Java applications on VMware Design and Sizing Guidelines* to establish the ideal compute resource configuration for your individual VMs, the number of JVMs within a VM, and the overall number of VMs on the ESX/ESXi host.
- Additionally, because this type of migration involves an OS/platform change along with JVM vendor change, it is advisable to review Section 2 of this document along with your vendors tuning advice for both OS and JVM.

How many and what size of virtual machines will I need?

This depends on the nature of your application; however, we most often see 2 vCPU VMs as a common building block for Java applications. One of the guidelines from *Enterprise Java applications on VMware Design and Sizing Guidelines* is to tune your system for more scale-out as opposed to scale up. This is not an inflexible rule as it depends on your organization's architectural best practices. Smaller more scaled-out VMs may provide better overall architecture; but you incur additional guest OS licensing costs. If this is a constraint, you can tune towards larger 4 vCPU VMs and stack more JVMs on it.

What is the correct number of JVMs per virtual machine?

- There is no one definitive answer as this largely depends on the nature of your application. The benchmarking you conduct can reveal the limit of the number of JVMs you can stack up on a single VM.
- The more JVMs you put on a single VM the more JVM overhead/cost is incurred when initializing a JVM. Alternately, instead of stacking up multiple JVMs within a VM, you can increase the JVM size vertically by adding more threads and heap size. This can be achieved if your JVM is within an application server such as Tomcat, so instead of increasing the number of JVMs, you can increase the number of concurrent threads available and resources that a single Tomcat JVM can service for your n-number of applications deployed and their concurrent requests per second. The limitation of how many applications you can stack up within a single application server instance/JVM is bounded by how large you can afford your JVM heap size to be and by performance impact. A very large JVM heap size beyond 4GB needs to be tested for performance and GC cycle impact, and the trade-offs need to be examined. This concern is not specific to virtualization—it equally applies to physical server setup.
- Follow the recommendations in *Enterprise Java applications on VMware Design and Sizing Guidelines*. They provide you with guidance on how to start tuning of GC if you have not done this exercise already.

4. References

- *Enterprise Java applications on VMware Design and Sizing Guidelines*
- *Guidelines: UNIX to Linux Novell Migration Tools: Novell Migration Tools* at http://www.novell.com/developer/porting_and_migration_tools.html
- UNIX to Linux IBM Migration Guide
 - IBM Linux Center: <http://www.ibm.com/developerworks/linux/>
 - Linux Migration tools: <http://www.ibm.com/developerworks/linux/library/l-roadmap.html>
 - IBM assessment paper: <http://www.ibm.com/developerworks/linux/library/l-solar/>
 - IBM guide <http://www.redbooks.ibm.com/redbooks/pdfs/sg247186.pdf>
- UNIX to Linux RedHat Migration Tools: *Red Hat Migration Center* page at http://www.redhat.com/migrate/rhel_overview/
- Best Practices for running Java in a virtual machine: <http://www.vmware.com/resources/techresources/1087>