vmware SOLUTION READINESS

Enterprise Java Applications on VMware High Availability Guidelines

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

Contents

1. Introduction 5
2. High Availability Designs 6
2.1 Overview: A Phased Approach to Achieving Higher Availability 6
2.2 Phase 1 Establish Zero Downtime Application Release Strategy 7
2.3 Phase 2: Establish Dynamic Runtime Mechanism to Handle Traffic Bursts 9
2.4 Phase 3: Establish a Disaster Recovery Strategy 14
3. Conclusion 15
Appendix A: Example Master Deployment Script 16
Appendix B: Load Balancer Example References 21
Appendix C: Database Synchronization Technologies 22

1. Introduction

This document addresses several dimensions of the high availability architecture of enterprise Java applications on VMware, including the:

- Ability to perform zero-down time (seamless) application releases during scheduled maintenance
- Ability to handle traffic bursts and maintain adequate SLAs
- Ability to perform effective disaster recovery

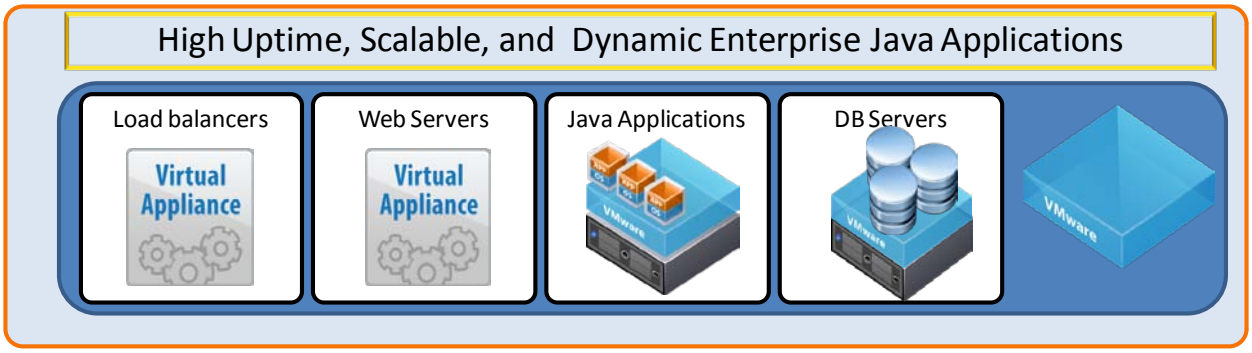
Key considerations and guidelines are provided for IT Architects who are in the process of evaluating various high availability options for enterprise Java applications.

Also see the companion document in this solutions kit, *Enterprise Java Applications on VMware Best Practices Guide*.

2. High Availability Designs

A number of designs are available to support high availability for enterprise applications on VMware. Enterprise Java application architecture has four main tiers as shown in Figure-1.

Figure 1. Enterprise Java Application Architecture

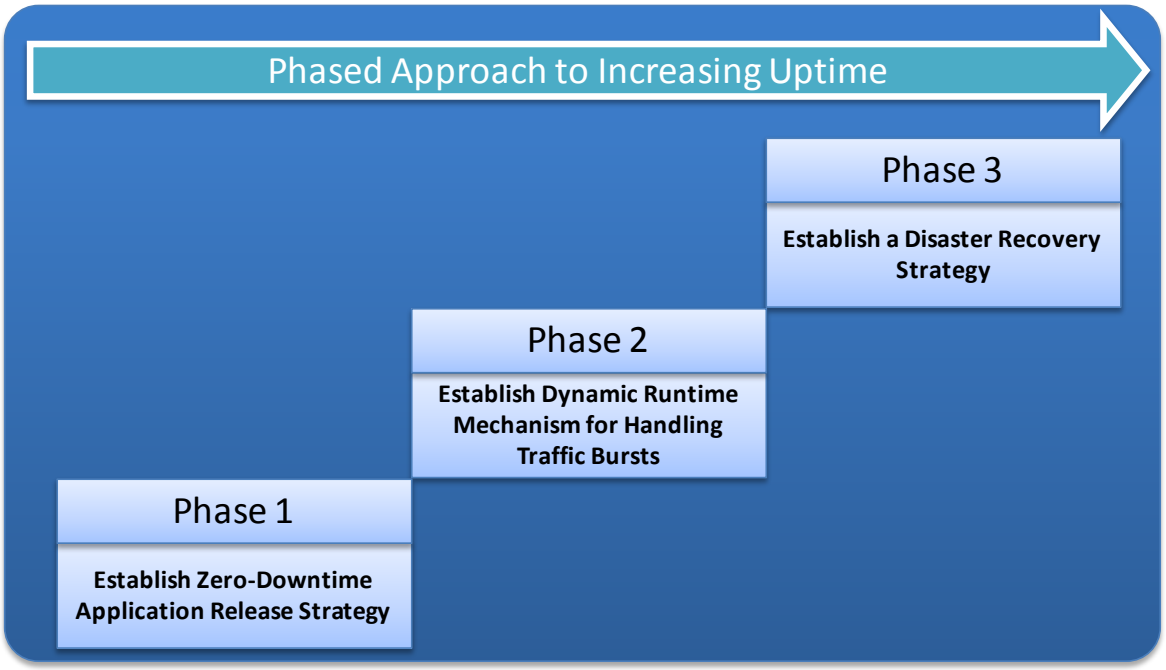


Each tier is typically separated into its own virtual machine (VM) that allows expansion both horizontally and vertically at runtime without downtime. This feature is critical to being able to deliver zero downtime application deployments, a dynamic ability to handle traffic bursts, and a better disaster recovery mechanism.

2.1 Overview: A Phased Approach to Achieving Higher Availability

Figure 2 illustrates the phased approach to achieving higher availability.

Figure 2. Phased Approach to Achieving Higher Availability



2.2 Phase 1 Establish Zero Downtime Application Release Strategy

2.2.1 Background

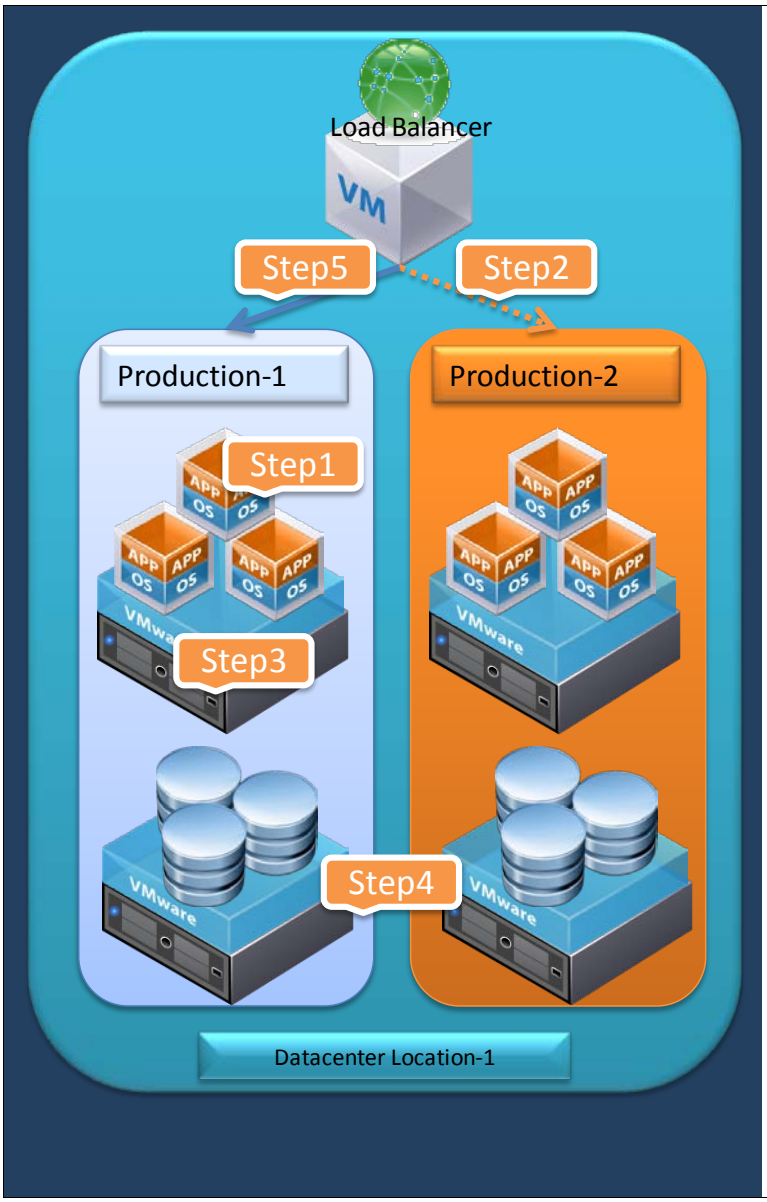
The following are key challenges when designing and implementing zero downtime releases for enterprise Java applications:

- Eliminate manual release steps through scripting – Application deployment often takes an unnecessarily long time because of too many manual steps in the release process. The first step towards a zero down time design should be to fully script each action. A scripting approach that helps to gather all the steps in each tier is to use Apache Ant along with various shell scripts, and the VMware vSphere™ API (see the VI-Java API presented in Appendix A). Fully automating all of the deployment steps helps to automate the rollback process as well. VMware provides the ability to take a snapshot of the VM's state, which is helpful if a rollback needs to be triggered.
- Determine your failover point – The most prudent method of switching is from the load balancer layer. At the load balancer layer you have the ability to divert traffic in a user-friendly manner with minimal impact or disruption to your application's SLA.
- Dealing with synchronizing the state of the database – Ultimately there needs to be two databases: one is for the primary site and the other is for the secondary site. Synchronizing and keeping the second database up to date is critical. This is particularly true if your database schema is being altered during the release. In some cases, where the schema does not need to be changed for the release because there are only data changes, the release can be more easily accomplished without having to use multiple database instances.
- Release orchestration challenges – Understanding the steps and dependencies of the various applications running on various VMs is critical to formulating the release orchestration. Java applications running on VMware vSphere can benefit from features such as VMware vMotion™ to move VMs around without downtime, or move VMs between hosts during a scheduled maintenance with minimal downtime using VMware HA.
- Choosing a Release Window – It is not an easy task to choose an appropriate release outage window for an enterprise Java application that has a 24/7 active user base. VMware vSphere features along with the zero-downtime release strategy help alleviate the need to negotiate a release window. Even so, it is prudent to perform releases during off-peak hours as the need to switch between Production-1 to a scaled down version of Production-2 (see Figure 3) minimizes additional infrastructure complexity.

2.2.2 An Architecture for Zero-Downtime Application Release

Figure-3 shows two production site replicas within one datacenter. This configuration provides the ability to switch-over from the Production-1 application cluster to the Production-2 application cluster. A high-level procedure for switching from one application cluster to another is also provided.

Figure 3. Architecture for Zero Downtime Application Release



To switch from Production-1 to Production-2

1. Before the release date, make a production replica of Production-1 and name it Production-2. Use VM templates to quickly and easily create the replica.
2. Switch from Production-1 to Production-2 using the load balancer (see Appendix B).
3. Take a snapshot of the VMs in the Production-1 environment. Begin the application deployment/release process using a master deployment script (see Appendix A for a sample script).
4. Synchronize the databases between Production-1 and Production-2 because more data might have been introduced during the time Production-2 was live. (See Appendix C for database synchronization technologies.)
5. If the QA process confirms that the deployment was successful you can switch at the load balancer level to Production-1. Otherwise, rollback to the last saved snapshot (from Step 3) and then switch back.

Because a release is typically performed during off peak hours, Prodcution-2 may be a scaled down version of Production.-

2.3 Phase 2: Establish Dynamic Runtime Mechanism to Handle Traffic Bursts

The ability to handle traffic bursts requires that you are able, at runtime and without downtime, to increase the size of VMs, add more ESX hosts to a cluster, and increase number of VMs. Additionally, it helps if you use a load balancer that is able to integrate with the VMware API to divert traffic to newly added resources. The F5 load balancer has this capability.

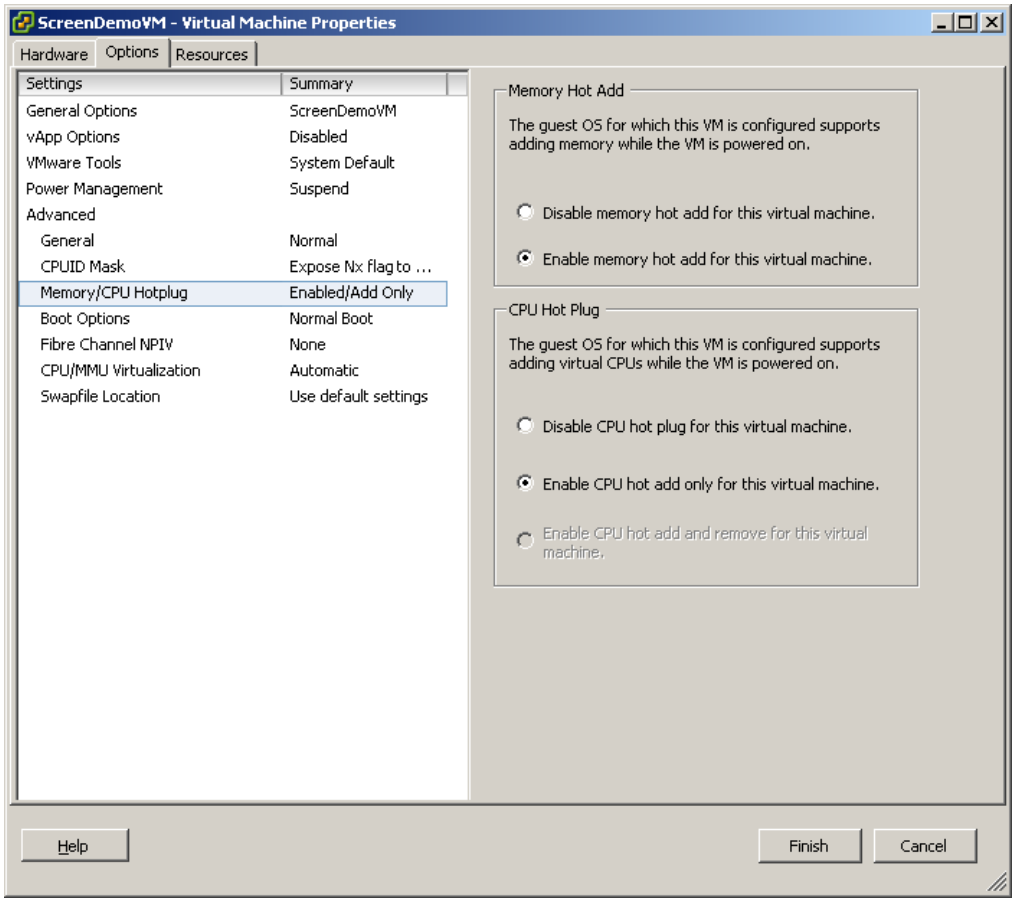
In addition to being able to add resources, you can easily remove them if they are no longer needed.

2.3.1 Vertical Scalability of VMs – CPU and Memory Hot Add

VMs with guest operating systems that support hot add of CPU and memory can take advantage of this ability to change the VM configuration at runtime without any interruption to VM operations. This is particularly useful when you are trying to increase the ability of the VM to handle more traffic.

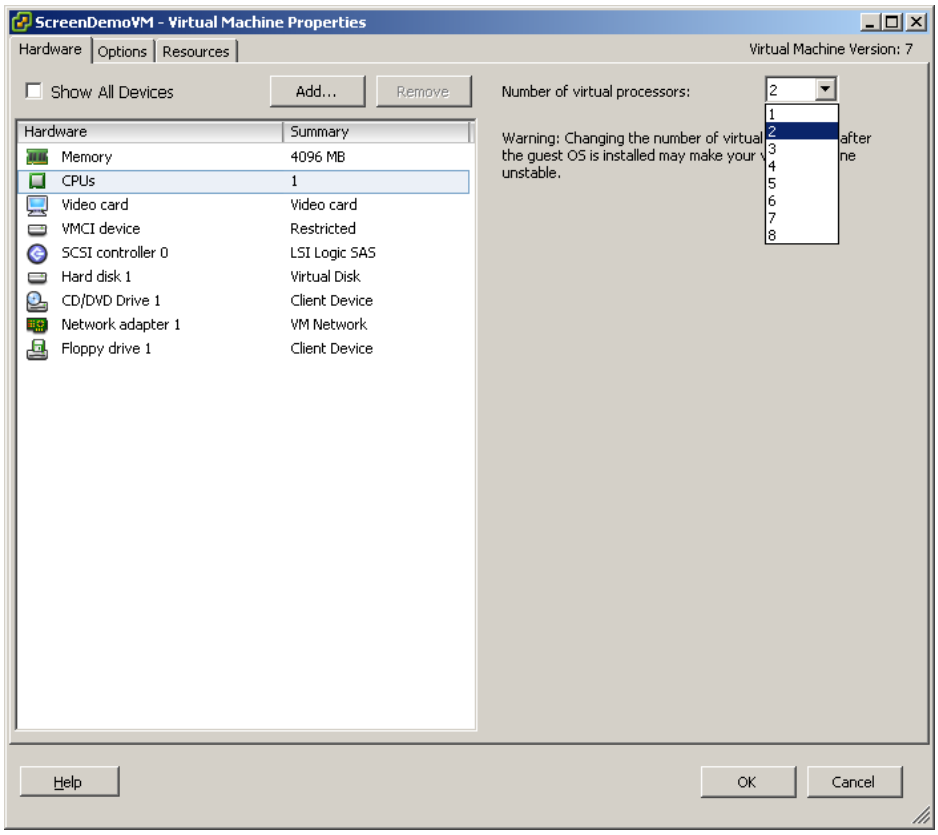
- Plan ahead and enable this feature. The VM must be turned off to enable the hot plug feature. Once enabled, hot add of CPU and memory can be done at runtime without VM shut down as long as the guest OS supports it.
- To enable hot plug/hot add feature, click **VM** > **Edit Setting**, and click the **Options** tab.

Figure 4. Virtual Machine Properties Options Tab



To add CPU without shutting down, select CPUs, select the number of virtual processors, and click **Add**. (You can add memory in a similar manner.)

Figure 5. Virtual Machine Properties – Adding CPUs



Often, when there is a need to increase Java heap space, there is also a need to increase vCPU count to get the best GC cycle performance. Keep in mind that there are many other aspects to GC tuning. Refer to Section 4 for additional references, including JVM documentation.

Always size the guest OS at least 1GB greater than the total heap space used. if multiple JVMs are being deployed on the VM, add each JVM's max heap size to the final guest OS memory allocation.

2.3.2 Horizontal Scalability – Add new VMs

You can create new VMs and add them to a DRS cluster, or create them on the intended host. Some load balancers such as F5 BIG-IP can detect the new VM, add it to its pool configuration, and divert traffic to it.

2.3.3 Adding New Hosts to a Cluster Using DRS and Resource Pools

Various hosts that make-up the tiers of a particular enterprise Java application can be grouped in a cluster where HA can be used to restart VMs on another host in case of a host failure. You can also use DRS to balance load between hosts in a cluster to achieve best performance.

Multiple resource pools can be used within a cluster to manage compute resource consumption by either reserving the needed memory for the VMs within a resource pool, or by limiting/restricting it to a specified level. This feature also helps meet SLAs and quality of service objectives.

Figure 6 shows how to enable HA and DRS.

Figure 6. Screen for Enabling HA and DRS

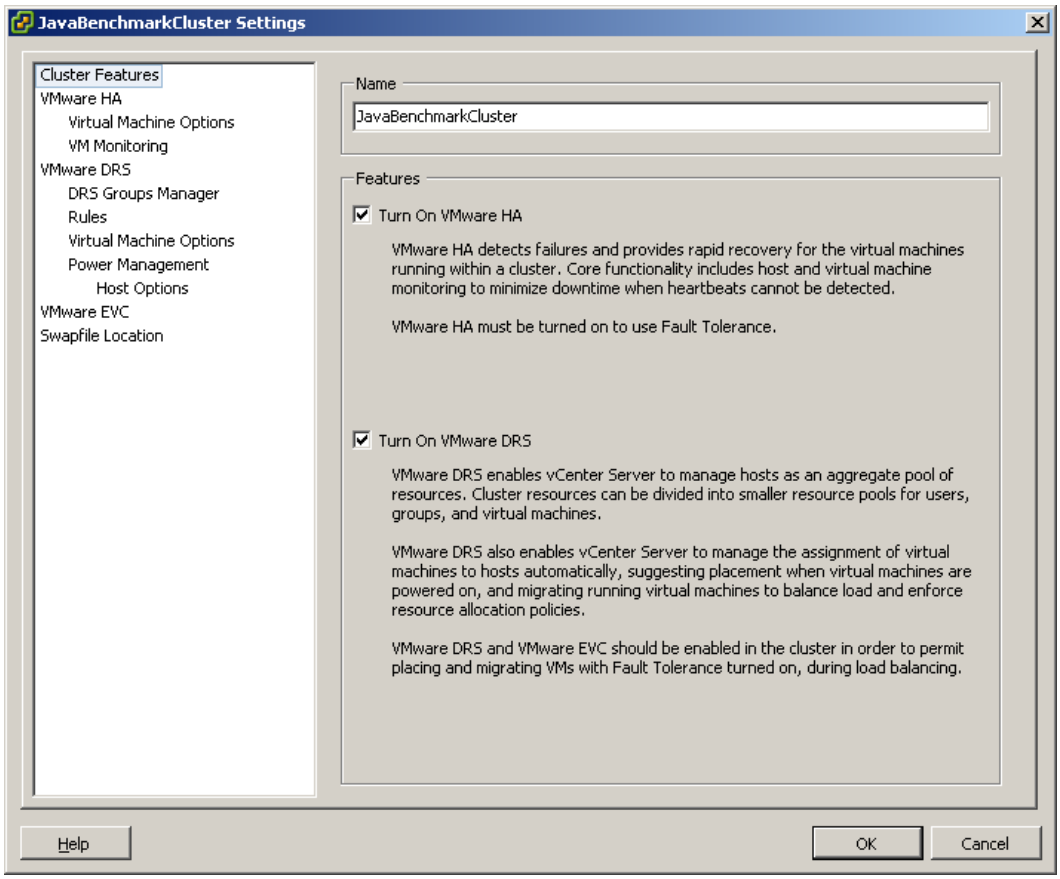
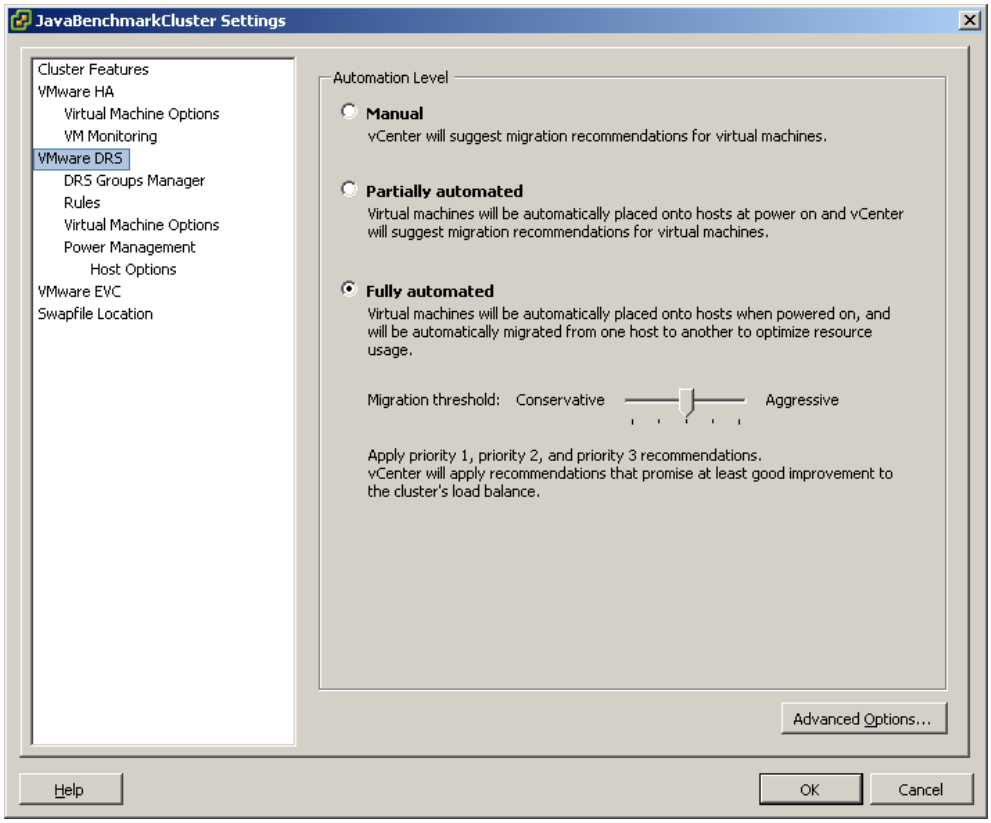


Figure 7 shows the three available automation levels for DRS: **Manual**, **Partially automated**, and **Fully automated**. In most cases, administrators new to virtualization first use **Manual** mode and watch the DRS recommendations over a period of time. When satisfied that the recommendations are valid they set it to **Fully automated**. This is especially helpful when you have hundreds of hosts to manage.

Further, DRS allows you to separate or couple VM rules. For example, if you have two Web servers load balancing traffic for your application, you would not want to place both of them on the same host cause both would go down if the host fails. You can set up a DRS rule that both Web servers should always be on separate hosts. Conversely, you may choose to always have two VMs on the same host if they frequently need to communicate with each other in a way that creates a lot of traffic.

With VMware HA and VMware DRS turned on, it is possible to achieve 99.9% uptime without the need for any additional clustering technologies. This assumes that other parts of the infrastructure have adequate redundancy and there are no underlying Java application code issues.

Figure 7. Screen Showing Automation Levels for DRS

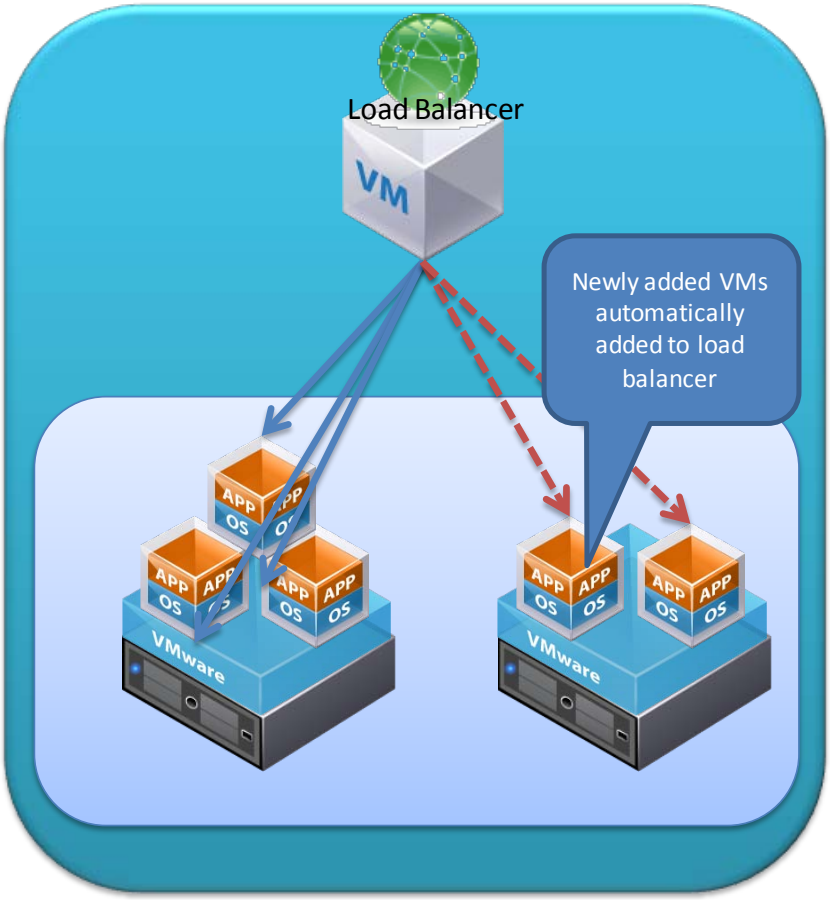


2.3.4 Using a Load Balancer to Discover Newly Added VMs

Creating new VMs in vSphere is straightforward; however, it has always been a challenge to make the necessary network changes to configure these new VMs so that a load balancer can direct traffic to them. F5 BIG-IP, through its iControl API, integrates with VMware vCenter to receive instructions that can trigger it to adjust network traffic in response to a spike in application usage. When new VMs are added in vCenter, the F5 BIG-IP Load Balancer can automatically add those new servers to its load balancing pool and can direct traffic to them.

Figure 8 illustrates how a load balancer automatically reconfigures traffic to the load balancing pool based on information it receives from vCenter to start directing traffic to the new VMs.

Figure 8. Using a load Balancer to Discover Newly Added VMs

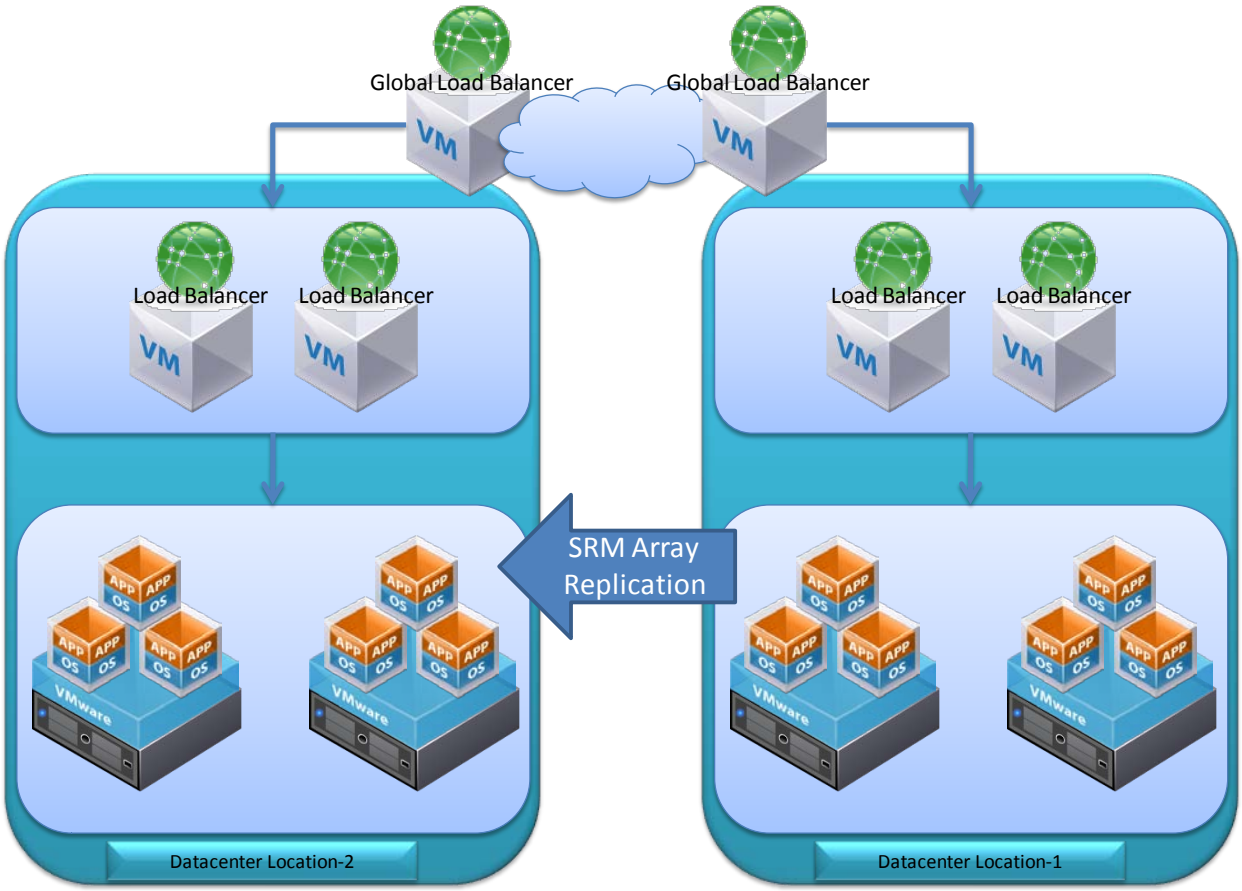


2.4 Phase 3: Establish a Disaster Recovery Strategy

The integration of F5 BIG-IP Global Load Balancer and VMware SRM provides a complete solution for automated disaster recovery between two data centers, or to the cloud. In the event of disaster, SRM automatically orchestrates the failover of VM guests and virtual infrastructure between the two sites, while the BIG-IP Global Load Balancer redirects all incoming client application traffic to the secondary site. The F5 BIG-IP Global Load Balancer and SRM are easily integrated via the F5 iControl API.

Figure 9 shows a typical vCenter Site Recovery Manager setup between two sites.

Figure 9. Typical VMware vCenter Site Recovery Manager Setup



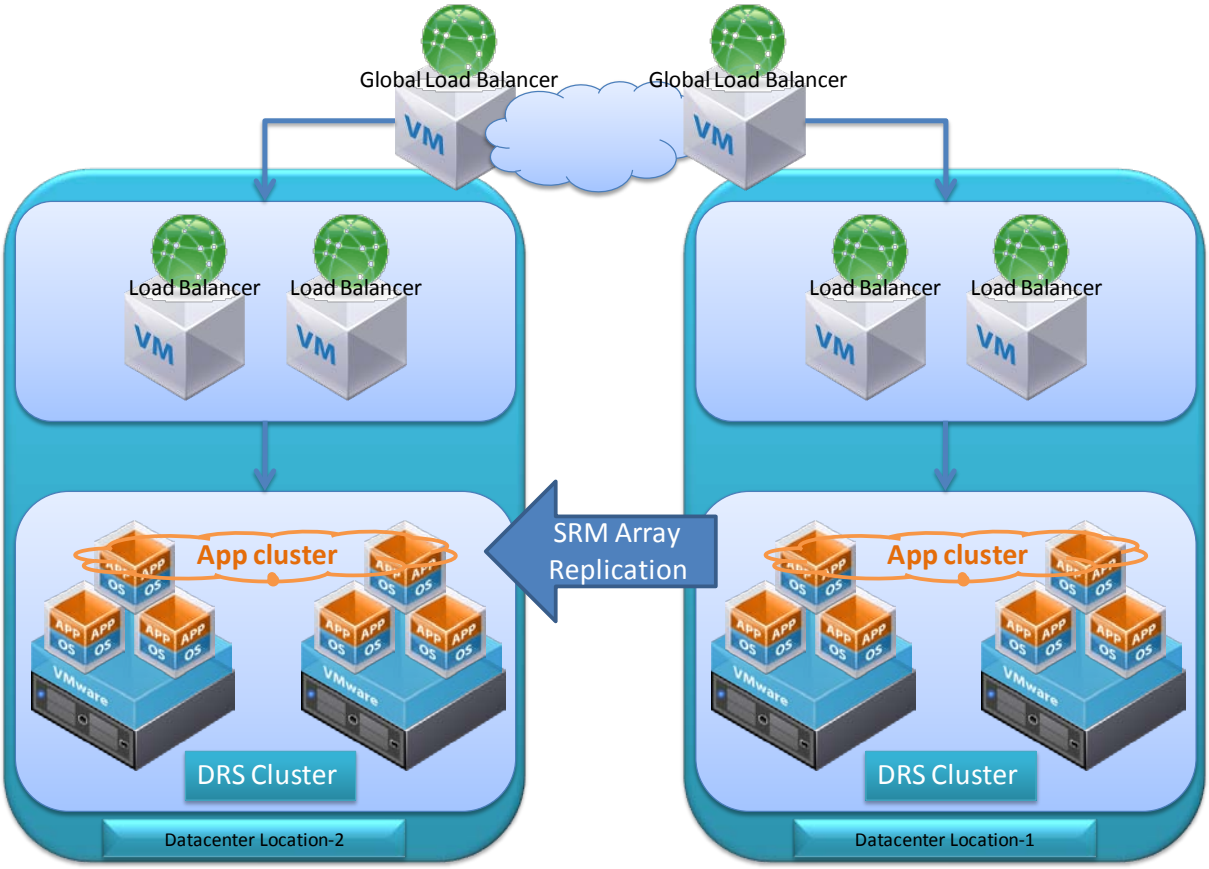
3. Conclusion

There are various ways to achieve a highly available enterprise Java application using VMware technology. The following features are critical to achieving optimal availability:

- Fully automated application deployment via master deployment script as discussed in Section 2 and Appendix A. This is critical to reducing the scheduled down time during releases
- Ability to hot plug/hot add CPU and memory of a VM at runtime without an outage
- Ability to add new hosts and VMs and have them automatically discovered and configured by the load balancer
- Ability to implement a disaster recovery solution using VMware vCenter Site Recovery Manager and load balancer technology

You can complement these features with your Java application server's cluster configuration. Figure 10 shows all of the features composed into one architecture that is capable of delivering 99.99% uptime. This assumes that other parts of the Infrastructure have adequate redundancy, there are no underlying Java application code issues, and the database server also has an uptime of 99.99%.

Figure 10. Architecture for 99.99% Availability



Appendix A: Example Master Deployment Script

This appendix provides information about creating a *master deployment script* for enterprise Java applications on VMware. The script has the ability to interact with VMs, manipulate guest OS files, and interact with Java application server (WAR file deployments) and the database. You can use Apache Ant and the VMware-VI-Java-API for interaction with the VMs to create a master script for all of the deployment steps. The following discusses various key functionalities and provides the needed script snippets.

Note Though a lot of this functionality is available from the vSphere Client via a UI, it is useful to be able to script this when you have multiple VMs to deal with.

The general framework of the master deployment script involves the following functional steps:

1. Take Snapshots of the VMs.
2. Interact with the load balancer to activate the switch-over from one application cluster to another.
3. Interact with the Java application server to deploy the application WAR file.
4. Interact with the database server to apply SQL script that may be part of your application release.
5. If rollback is need, interact with VM to revert to prior snapshots.

The following script snippets are applicable for a single VM scenario. You need to modify these scripts to handle multiple VMs. To iterate for multiple VMs you can re-use the single VM methods presented here in a multiple iteration/loop through the set of VMs that are part of the environment undergoing a Java application release

Skeletal Outline of the Master Deployment Script

This script is based on using Apache Ant and VI-Java-API.

```
<project name="MyProject" default="runMasterDeploymentScript" basedir=".">

 <target name =”runMasterDeploymentScript”
depends=”takeSnapshotOfVM,switchLoadBalancer, deployDBChanges, deployWARFile ”>
 </target>
 <target name="takeSnapshotOfVM">
        <java
         fork="true"
         maxmemory="256m"
        jvm="${JavaHome}"
        classname="yourcompay.SnapshotManager" >
        <arg value="-takeSnapshot"/> <arg value=”${vcenterURL}/>
   <arg value=”${adminUsername}”/> <arg value=”${adminPwd}/>
   <arg value=”${vmname}”/> <arg value=”${snapshotname}”/> <arg value=”${desc}”/>
                <classpath>
                  <pathelement path="${VIJavaHome}/vijava2120100715.jar"/>
                </classpath>
        </java>
 </target>
 <target name=”switchLoadBalancer”>
  <!—- invoke Load balancer API or she script to switch from Production-1 to
Production-2 -->
 </target>

<target name=”deployDbChanges”>

  <sql driver="org.database.jdbcDriver" url="jdbc:database-url" userid="sa"

    password="pass" >

    <transaction src="data1.sql"/>

</sql>

</target>
 <target name=”deployWARFile”>
 <!—use Ant Copy task to copy file to destination and Ant Unzip task to extract →
</target>

<target name="rollBackToPriorSnapshot">
        <java
         fork="true"
         maxmemory="256m"
        jvm="${JavaHome}"
        classname="yourcompay.SnapshotManager" >
                <arg value="-revertSnapshot"/>
                <arg value=”${vcenterURL}/>
   <arg value=”${adminUsername}”/> <arg value=”${adminPwd}/>
   <arg value=”${vmname}”/> <arg value=”${snapshotname
                <classpath>
                  <pathelement path="${VIJavaHome}/vijava2120100715.jar"/>
                </classpath>
        </java>
 </target>
</project>
```

How to Interact with a VM Using the VI-Java-API

Take a Snapshot of a VM

You can either wrap these calls in a Java command-line call or include them in and Ant Java task. Use the Service Instance object to connect to vCenter. This is then used to execute various VM functionality.

```
Import com.vmware.vim25.*;
//note you will need vijava2120100715.jar for example on your CLASSPATH
Public class SnapshotManager {
//…..
public ServiceInstance getServiceInstance(
String vcenterURL, String adminUsername, String adminPwd)
{
  ServiceInstance si = null;
  try {
    si = new ServiceInstance(new URL(vcenterURL), adminUsername, adminPwd, true);
  } catch (RemoteException e) {
    e.printStackTrace();
 } catch (MalformedURLException e) {
    e.printStackTrace();
 }
 return si;
}
Public static void main(String[] args)
{
 SnapshotManager ssm = new SnapshotManager();
 If (arg[0].equals("takeSnapshot")
 {
  //Ssm.takeSnapshotOfVM( vcenterURL, adminUsername,adminPwd,
vmname,snapshotname, desc)
  ssm.takeSnapshotOfVM( arg[1], arg[2],arg[3], arg[4],arg[5], arg[6])
 }else if (arg[0].equals("revertSnapshot"))
 {
   ssm. revertToSnapshotOfVM( arg[1], arg[2],arg[3], arg[4],arg[5])
 }
}
```

Using the `getServiceInstance` method you can take snapshot of a VM:

```
public String takeSnapShotOfVM( String vcenterURL, String adminUsername, String
adminPwd, String vmname, String snapshotname, String desc)
{
  ServiceInstance si = getServiceInstance(vcenterURL, adminUsername, adminPwd);
  Folder rootFolder = si.getRootFolder();
  String returnFlag = null;
  try {
    VirtualMachine vm = (VirtualMachine) new InventoryNavigator(
            rootFolder).searchManagedEntity(VIRTUAL_MACHINE, vmname);
    Task task = vm.createSnapshot_Task(snapshotname, desc, false, false);
    if(task.waitForTask()==Task.SUCCESS)
    {
      returnFlag = Task.SUCCESS;
    }else
    {
      returnFlag = "failure";
    }
      } catch (InterruptedException e) {
         // other catch clauses omitted for brevity…
         // other catch clauses go here…
      }
     return returnFlag;
}
```

If you need to take snapshots for multiple VMs, you can iterate the method N times to get a snapshot for the set of VMs.

Revert to a Snapshot

```
public String revertToSnapshotOfVM(
String vcenterURL, String adminUsername,
String adminPwd, String vmname, String snapshotname)
{
       String returnFlag = null;
       try {
       ServiceInstance si = getServiceInstance(vcenterURL, adminUsername, adminPwd);
       Folder rootFolder = si.getRootFolder();

       VirtualMachine vm =
(VirtualMachine) new InventoryNavigator(rootFolder).searchManagedEntity(
"VirtualMachine", vmname);

       VirtualMachineSnapshot vmsnap = getSnapshotInTree(vm, snapshotname);
       if(vmsnap!=null)
       {
              Task task;
              task = vmsnap.revertToSnapshot_Task(null);
           if(task.waitForTask()==Task.SUCCESS)
           {
                     System.out.println("Reverted to snapshot:"+ snapshotname);
                     returnFlag = Task.SUCCESS;
              }
       }

       } catch (InvalidProperty e1) {
                     // other catch clauses omitted for brevity…

              }
              return returnFlag;
       }
```

The above `revertToSnapshotOfVM` method depends on `getSnapshotInTree`, the full source listing is provided by SourceForge at:

http://vijava.svn.sourceforge.net/viewvc/vijava/trunk/src/com/vmware/vim25/mo/samples/vm/VMSnapshot.java?revision=189&view=markup

Interact with Load Balancer

Depending on which load balancers you use you will need to refer to the appropriate documentation provided by the vendor for using their API and Management Interface. Appendix-B lists some Load balancers for you references.

Appendix B: Load Balancer Example References

The following load balancers have a virtual appliance model for VMware vSphere.

- F5 *VMware vSphere Solutions*: http://www.f5.com/solutions/applications/vmware/vsphere/
- Zeus product information: http://www.zeus.com/products/load-balancer/index.html
- Coyote Point Equalizer VLB and Equalizer VLB Advanced product information: http://www.coyotepoint.com/

Appendix C: Database Synchronization Technologies

When switching between Production-1 and Production-2 sites as shown in Section 2 of this document, the Java application's back-end database is switched over as well and it needs to be synchronized. The synchronization may include schema changes and data changes. There are various approaches to database synchronization depending on the database vendor:

- Oracle Database: Choose from DataGuard, Oracle Streams or GoldenGate
- MS SQL: Server Transactional Replication
- MySQL: Replication

Because the original Production-1 site was updated with the master deployment scripts (latest data and schema), attention is more on the Production-2 database data that needs to be replicated back to the Production-1 database just before Production-1 is turned back to live. This is because the Production-2 database was live for the duration of the application release and new transactions would have been entered by users that now need to be replicated back to Production-1.