Qlusters

openQRM Technical Overview

Open Source - Data Center Management Software

November 2006

Qlusters, Inc.
1841 Page Mill Road, G2
Palo Alto, CA 94304
www.qlusters.com
650-812-3200

Table of Contents

1. High Level Overview

openQRM is a collection of data center management tools that covers many facets of management. openQRM provides provisioning of the entire software stack on physical servers and virtual machines like VMware and Xen. openQRM also has a policy engine so that resources can be automatically provisioned based on external business needs as well as the requirements of internal organizations.

Out-of-the-box functions include, but are not limited to: provisioning, monitoring and high availability. Other functionality and integration with third-party software can be added through plug-ins. Extended capabilities are provided for the Nagios integration with openQRM.

1.1. Terminology and Abbreviations

Computer Resource ("Resource") - A medium that can run services (i.e. a node or a partition).
Virtual Environment – Description of how to provision a service. Required hardware profile, governance policies, etc.
Image – Captures a service's file base.
Provisioning – Assigns resources to a service.
Deployment – Ensures that a resource uses a certain Image (i.e. begin a service).

1.2. Provisioning

OpenQRM allows for the separation of applications and resources. The application (and OS) is captured in an Image and stored on central network storage. The image can then be deployed to a suitable resource, which can either be a physical machine or a partition.

Determining what images to use on which machines is defined by an entity called a Virtual Environment (VE). The VE definition holds everything necessary to deploy a given image on any range of hardware — including hardware requirements, which images to use, HA requirements and provisioning policies.

When provisioning, the openQRM server selects the appropriate resources from a pool of idle (unused) resources and assigns them to the VE. The selection is done based on the VE's configuration. The image is then deployed — the resource can commence running the service that the VE represents. When the VE stops (the service is no longer needed), the resource is de-assigned from the VE/image and is considered idle.

OpenQRM supports several types of deployments (more on each below):

- Network based
  - PXE + NFS
  - PXE + software iSCSI
  - Hardware iSCSI nics
- Local deployment
  - Physical or virtual machines
  - Special support for virtual VMware ESX machines

1.3. Integration & Extensions

openQRM provides a rich set of capabilities which allows the enhancement of its functionality and enables the integration of its operations with other systems.

- Hooks – predefined bash functions utilized for special events
- qrm-cli – an extensive CLI which manages openQRM from scripts
- Integration scripts
  - Storage management scripts
  - Network management scripts
- Free text parameters set in the entity definitions: resource, VE, image, storage server, etc. These can be used by the integration scripts, cli scripts or plug-ins to provide additional data (i.e. the password to the storage server).
- Plug-ins – These offer the highest degree of extension. Plug-ins allow the enhancement of the GUI and provide the ability to add functionality, react to events and use inner methods of the engine, etc.

1.4. Supported Operating Systems

openQRM supports the following operating systems:

- Windows – hardware iSCSI, local deployment
- Red Hat/SUSE Linux – all versions of Linux
- Solaris Sparc/x86 – as a community plug-in – network deployment
- FreeBSD - as a community plug-in

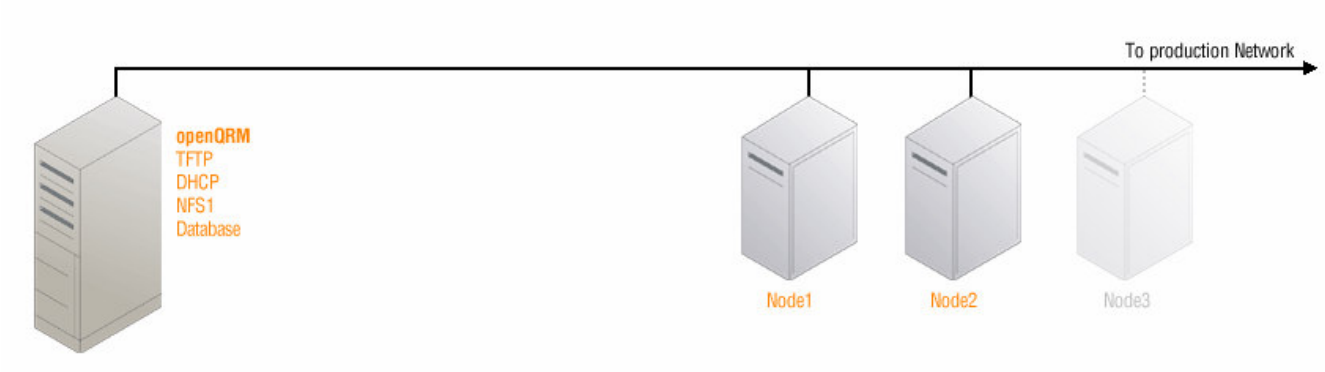
openQRM supports the following virtualization technologies:

- VMware ESX
- Xen – as a community plug-in
- vserver – as a community plug-in
- qemu – as a community plug-in

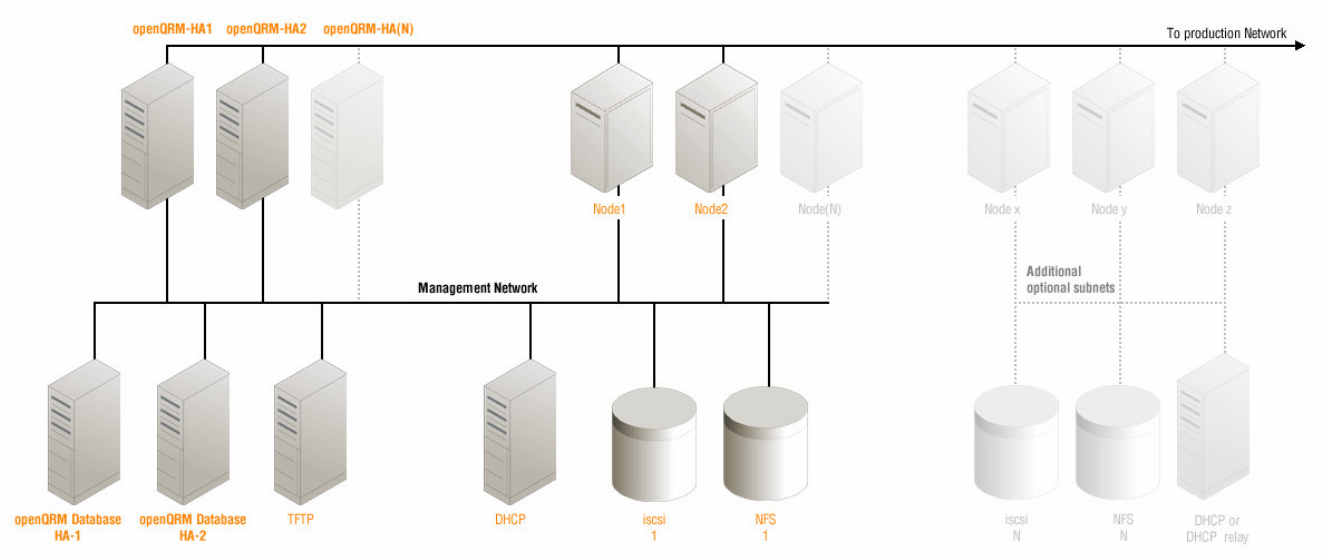
1.5. System Architecture

The openQRM server engine is a Java-based web application that runs under Tomcat with a MySQL back end. All other functionality is either called or changed from this server.

Much of openQRM's day-to-day management of is done via the web interface exposed from the engine. There is also a set of command line utilities that can be used for most management functions. The openQRM server performs or interacts with a variety of roles within the data center. In addition to the server itself, it uses DHCP, TFTP and NFS or iSCSI. These roles can be placed on different servers for better scalability and availability.



Drawing 1: Minimal openQRM installation



Drawing 2: Expanded openQRM installation

- DHCP – can either be a dedicated DHCP server (or servers), or a service running on the openQRM server. In the latter case, it is recommended to use the dhcpd plug-in to ease setup
- TFTP – similarly to DHCP, this can be a dedicated server, or installed as a plug-in. If a dedicated server, openQRM should be able to write to the directory it serves.

2. Booting

In general, the booting cycle of a resource is as follows:

- Request the resource's parameters from the openQRM server. This is done by sending an HTTP request which includes the ID, MAC and IP of the node along with 'starting_system' event. The openQRM server sends the resource parameters back in

response to the request including (among other items) information needed to mount the root file system of the assigned image.

- Set-up of boot services – boot services is a means of dynamically associating services that run when a resource boots. A service can be associated with a specific resource, all resources, a specific VE or all VEs, a specific image or all images.
  - For example, a monitoring agent can be bundled as a boot service, and associated with all resources. A backup service can also be associated with all VEs (in this case it will not run on idle resources).

2.1.1. Node Agents

A boot service that is continuously in use is the node-agent's boot service. It commences two small daemons for openQRM to use. The first is called qrm-infod, and is an information or monitoring daemon. The second is qrm-execd, and is used by the openQRM server to send remote commands to the node (like reboot). For security reasons, qrm-execd has a limited set of commands that it can run (however these can be extended by plug-ins), and it will only listen to commands sent from the openQRM server.

In order to be properly managed, especially as idle, all resources must be able to boot using PXE. This requires the use of DHCP and TFTP servers as mentioned above.

2.2. PXE Boot

The DHCP server is configured so that its 'next-server' parameter points to the TFTP server. When booting the kernel via PXE, nodes are recognized by their MAC address.

PXE requests a DHCP lease, and in response to this request, the DHCP server tells the node which server has the pxelinux configuration file. According to its MAC address, the node will try to obtain a configuration file from the TFTP server.

Once the initrd is uncompressed, it runs an adjusted (by openQRM during the boot image creation process) linuxrc file that requests the parameters for this resource from the openQRM server. These parameters dictate how to proceed.

2.3. Non PXE Boot

Some deployment methods do not work with PXE boot. In a case where a resource is assigned to an image without PXE boot, it will still attempt to boot with PXE. The PXE configuration, however, will instruct it to go to the next device (which can be local disk, HBA card, etc.).

2.4. Idle Resources

Idle resources boot from PXE, and run from inside the initrd, waiting for commands.

3. Resources

openQRM can use either physical nodes or virtualization partitions. In openQRM, they are almost indistinguishable from one another.

Managed nodes are auto-discovered on the first boot: When the node tries to reach its default PXE configuration, one does not exist — so it gets the default. When the initrd loads, it recognizes this (the parameter 'id' in /proc/cmdline has value 0), and sends a “new_node” event to the openQRM server. The server then adds the node to the database. During registration, the openQRM server creates pxelinux configuration files in the TFTP directory ensuring that the next time the node boots, it will receive its specific pxelinux configuration file.

Managed Partitions are created explicitly by the user, through the openQRM GUI.

4. Images & Storage

When a resource is assigned to a VE, an image associated with that VE is deployed on the resource. Images are a capture of a file system (in several ways, described below). They are stored in a storage server which can be anywhere within the routable network. Today, this includes both NFS and iSCSI servers.

To manage images and storage servers, definitions should be created in the openQRM server. The definition of an image includes the identifier and storage server where it is located. The definition of a storage server includes its IP.

To utilize the special capabilities of enterprise storage servers, such as NetApp, openQRM contains the concept of a *storage server type.* A user can define a new type and configure a management script used to manage storages of that type. The script supports the commands 'mirror,' 'clone,' 'delete,' 'map,' and 'unmap.'

Storage could potentially be run directly from the openQRM server as a Linux service, (it is recommended to use the local_nfs plug-in). This, however, this is not as robust as running on a second (or more) server from a storage vendor such as NetApp. Assuming that the export options and permissions are set correctly, any NFS server can be used as a storage server.

Creating file system images is done in several ways. See below for more details

5. Deployment

5.1. PXE Based

With PXE-based deployment, the resource boots with PXE, receives a kernel and initrd and mounts the image as specified by the resource parameters received from the openQRM server. After mounting (and possibly preparing) the specified storage location (NFS/iSCSI,

etc.), the initrd performs some limited changes and verification of the image. This includes adjusting any init scripts required (for example turning off the network stop so that there are no issues when trying to reboot a node with a network mounted file system). These changes are only related to system initialization and do not affect the application or environment. After this, the initrd changes root into the mounted file system and start the init process normally.

NFS file systems are created using the CLI command 'qrm-filesystem-image.' After specifying the existing online source system and target storage server, the command takes care of the image copy and performs extra openQRM-specific preparation of the image. The image is taken at the file level — not at the block level as NFS is used.

Software iSCSI images are created using an “automatic” method from either running servers or from existing NFS images. This involves adding some extra parameters to the file system image definition in openQRM. These parameters specify extra iSCSI features for the target such as the target name, partition number and LUN, etc. It also includes NFS source parameters. In order for the installation to take place, you must set 'INSTALL_IMAGE=true' in the file system image parameters. When this is set and the file system image is initially assigned, the initrd will create a file system and partitions on the iSCSI target. After mounting both the iSCSI target and the NFS source, it then copies files from the source to the target and does some image preparation similar to the NFS installation. Note that since the source is NFS, the copy is still done at the file level instead of at the block level. NFS file system images can be created using this method as well.

5.1.1. Shared

openQRM can mount the same NFS file system image from multiple physical nodes. This helps with management and administration — allowing you to apply a change to one node — thereby updating every node using this image. One option provided by shared images is to keep a single file system image as a base image and then dynamically mounting your application based on the identity (in this case the relevant identity is most likely the Virtual Environment that the node belongs to) of a given node.

Clearly, certain parts of the file system will need to be mounted with a private copy so that each node can write independently to this area. This includes most notably /tmp and certain parts of /var (like 'log' and 'run'). This is handled by keeping a '/private' directory on each node that is dynamically mounted during boot (in the initrd) based on the VE ID and the VE member ID (an incremental number given to each resource assigned to a VE). These are mounted with the 'mount --bind' option so that they are mounted in line as part of the same file system. Any directory can be included as private including /, etc.

When a new VE member is added to a VE, it copies the original directory contents into a new private copy. Each time a new VE member boots, it retains this copy so that changes are persisted to its private directories.

Under private directories you can still share areas among all nodes that use the file system image (i.e. /var/lib/rpm) which contains the local RPM database. You can configure this with private excludes during the file system image creation. Using this, you can also ensure that all nodes view the same RPM database or any other configuration items that you wish to remain common.

5.1.2. Cloning

When deploying an NFS image, it is occasionally desired for each resource to use a clone of the original image, rather than mounting it as shared. With clones, each image can be manipulated freely without the possibility that the changes will affect other resources. Cloned images do not need the /private manipulation.

openQRM supports cloning using enterprise storage capabilities (such as NetApp's FlexClone). This is done using the storage server type management script (see above). Depending on the storage server, cloning is fast and takes up almost no space. openQRM supports NetApp FlexClone.

5.2. Hardware Network Boot

Hardware network boot uses special hardware cards to network boot a system. The hardware card acts as a SCSI device and connects to the storage server when data is requested.

When contacting the storage server, a mapping in the server tells it what image to use to provide data to the card. In order to create this mapping, the 'map' command in the storage type management script is called during resource assignment.

When a resource is assigned to a HW network boot type, its PXE configuration is changed so that the resource does not continue to boot from PXE, but continues to the HW device.

5.2.1. Hardware iSCSI

Hardware iSCSI cards communicate with the storage server using iSCSI. The qrm-hw-iscsi-manage script creates the right mapping in the storage type based on the IQN parameter that is set in the resource.

With such cards, the image is created by mapping an unmanaged node to the desired location (LUN) in the storage server, booting it, and installing the OS and services on it.

Hardware iSCSI deployment supports Linux and Windows. When using Windows, a package needs to be installed on the golden image during creation.

5.3. Local

Local deployment is when the image is copied from the storage into a local disk.

5.3.1. Local Disk

When deploying an image to a local disk, the resource first boots to PXE and loads the default initrd. The initrd code recognizes that the image to use is a local deploy type and proceeds to copy it from the storage server to the local disk (which is automatically discovered). Once complete, the pxe configuration changes to tell the resource to boot from the local disk, and the resource reboots.

To create an image, a node that has been installed locally is used. The CLI 'qrm-cli filesystem local grab' is used to notify the openQRM server with the details of the node. The node should be rebooted and set to boot from PXE. It loads the default kernel and initrd from the server, but is not detected as a managed node. The code in the initrd recognizes its need to obtain the local disk and copies its contents (partitioning table, partitions and MBR) to the storage server location as defined in the CLI. Once complete, the node reboots locally again.

This method supports Linux and Windows. When using Windows, a package needs to be installed on the golden image before grabbing.

5.3.2. VMware ESX

With the use of VMware ESX, the image is a VMware virtual disk. When a partition is assigned to this image, it is copied and the partition is configured to use the copy.

The image is generated by creating a locally booted partition, installing the OS and services on it and then defining the image parameters in openQRM. (In ESX, virtual disks always reside inside a storage server.) This method supports Linux and Windows. When using Windows, a package must be installed on the golden image.

5.3.3. Monitor Only/Easy Migration

It is possible to take an active node running a particular service and integrate it into openQRM. A VE is created to represent the service, and the node and VE are monitored. However, no deployment occurs.

The benefits of using this are as follows:

- Monitoring the node/service using openQRM tools
- Integrating services that are required by openQRM (see VMware ESX support below)
- Grid computing – the node is set to PXE boot. Should the VE stop, the node is idle and can be used for other VEs (as long as they don't use its disks). When the VE starts, it will be assigned to that node and will boot locally – running the original service again (assuming that it is idle)
- Using boot services on that particular node.

6. Virtualization Support

openQRM can manage and utilize virtual machines.
Supported virtualization technologies are implemented through plug-ins:

- VMware ESX
- Xen – as a community plug-in
- vserver – as a community plug-in
- qemu – as a community plug-in

A plug-in can easily add support for other technologies.

When a virtualization plug-in is installed, a VE can be defined as Host using that technology (e.g., VMware ESX host). When the VE is active, it is assigned nodes that act as hosts. The user can create and start partitions on those nodes using the openQRM GUI or CLI. These partitions then become part of the idle resources pool.

When the user wants to use partitions in a VE, the user specifies the VE configuration to use partitions of the desired technology. Matching partitions are then assigned to the VE.

6.1. VMware ESX

To work with VMware ESX, the hosts need to be integrated with openQRM using 'monitor only,' as described above.

The VMware ESX support also handles local deployments better by directly manipulating the virtual disks.

7. High Availability

openQRM has three different types of high availability (HA) or failover. Two types can be used for managing resources, while the third is for the openQRM server itself.

7.1. Automatic Hardware Failover (AHF)

As openQRM monitors system metrics constantly, it uses the monitoring data as a heartbeat between the managed resources and openQRM server (explicit heartbeats are sent if the monitoring statistics don't change). qrm-infod on the managed resource handles sending these metrics, and posts the information directly to the openQRM server. To enable AHF on a specific VE, simply select 'AHF' in the GUI, and openQRM takes care of the rest. Should openQRM lose its heartbeat from a node for a specified period of time, openQRM will provision a new resource with that same image and try to kill the original resource to prevent conflict. If power management or a managed switch is available, this can be tied into the openQRM hooks to forcibly ensure that an active-active scenario is not encountered.

7.1.1. High Availability Pool

In order to not risk a case where a VE requires a replacement resource, and there are no (matching) resources in the idle pool, the user can set thresholds on the number of resources that should act as standby for the VE, or globally. Several VEs can rely on the same set of resources as standbys.

The system does not enforce this threshold, but issues alerts when it is met so that new resources can be added.

7.2. Automatic Application Recovery (AAR)

In addition to using system level monitoring data as a heartbeat, AAR uses application level checks to determine the health of the application. These application level checks are accomplished through application health agents. The health agent is what actually talks to your application to see if things are going smoothly. For example, a health agent can ensure that it can read or write a specific database table. If it cannot read the table, it sends an AAR event to the openQRM server with 'qrm-cli aar'. A VE can be configured to require an active heartbeat and trigger an alert if it is missed, or just wait for an event and then take specific action. The actions AAR can trigger are either to reboot a particular resource, reboot all resources that belong to that Virtual Environment, or simply report the error.

7.3. openQRM High Availability

The openQRM server itself can have high availability through a plug-in called qrm-ha. In this case you would set up most of the openQRM server's application files to be mounted from NFS. Two systems are configured to boot to the local hard drives. They will have openQRM installed, but still residing on the NFS share. The first system to come up is the active system. It mounts the application directories and starts the application server. The second system that comes up is the hot standby node. It will run only the qrm-ha agent while running. When the qrm-ha agent on the standby node determines there has been a failure of the active openQRM node, it becomes the active node, takes over the IP address, and starts the appropriate services. If the first node is able to recover, it will come back into the HA pool as a standby node. The HA pool can have multiple standbys.

8. Provisioning Policies

8.1. General

Policies are used by openQRM to determine how many resources a VE should be using at any given time. There are two types of policies: internal and external. As part of the policy, each VE has a range of systems that it uses. It has "min," "max." and "start-with" parameters — with the latter determining the number of resources that the VE should use when first starting.

8.2. Internal Policies

Internal polices are policies that are fully managed by openQRM. Using an internal policy, you can set thresholds of average utilization that determine when to add or remove resources from a VE. There are two pre-configured internal polices: high-load and low-load. The pre-set 'low-load' policy is configured as 30-60. When the average CPU utilization for all systems in your VE for the last minute tallies over 60%, you need to continue adding resources to the VE until you are either under 60%, or have reached your max number of systems. Conversely, if the average utilization goes under 30%, openQRM will remove resources until average utilization is either over 30%, or the minimum number of systems is reached. You can add new internal policy thresholds with 'qrm-cli policy'.

8.3. External Policies

External Policies are managed entirely by the user. The file selection for the external policy should be an external executable that receives its input (including metrics) from the environment's variables. Based on this input or other input gathered, the external policy can call 'qrm-cli virtual-environment' to add or remove resources. External policies are called at most once a minute and only then if the monitoring metrics have changed significantly since the last polling cycle. They can also be triggered from within the GUI.

9. Provisioning Requests

Provisioning requests enables requesting resources and services to be a standard procedure. A user requests a number of resources, with a predefined system (e.g., 'apache' or 'windows xp'), and dates to start and end the provisioning of these resources. The request waits for approval by an admin. Once approved, a VE is created on a specific start date with the requested image and resources assigned. At the end date, the VE is stopped. The user is notified by email about any changes to the request (i.e. approved/denied, started, stopped, failed).

10. Monitoring

The openQRM server collects a number of system level metrics by default. These metrics cover used memory, total memory, CPU number and utilization, and load average. They are posted to the openQRM server by qrm-infod. Great care has been taken to ensure that the agent affects as little as possible the system it is running on as well as network bandwidth.

openQRM provides reports based on this data.

10.1. Metrics

In additional to the built in metrics that are gathered, metrics can be gathered from any area desired (e.g. throughput on the file server the resource is using) by sending them to openQRM with 'qrm-cli metric'. Metrics can be obtained and set per VE or node. In addition, these metrics are passed into the environment for use by external policies. All metrics show up in the GUI.

10.2. Nagios

A plug-in provides support for monitoring managed resources with Nagios. The plug-in provides the following capabilities:

- Integrates Nagios reports into the openQRM GUI.
- Automatically configures Nagios sensors on resources. This is done by having a boot service which receives the list of sensors for each VE, as configured in the openQRM GUI, and activates them on the resource.
- Reconfigures the Nagios server when assignments/de-assignments occur so it is aware of services running in the DC. The plug-in does this by registering event listeners.

For openQRM, using Nagios provides superior monitoring capabilities. For Nagios, the openQRM plug-in makes installing Nagios sensors much easier and automatically reconfigures Nagios as the Datacenter evolves (resources are assigned), so these changes are reflected in monitoring the services. This eliminates the need to manually maintain the Nagios configuration.

10.3. Events

The openQRM server creates notifications to the user/plug-ins about various events in the managed resources, the VEs and the server itself, etc. If these events signify an error, the notification has the level of alert. If a problem is resolved, openQRM tries to reduce the level of the alert automatically by detecting that the problem has been resolved, and by finding the alert matching the problem. For example, if openQRM has not received a heartbeat from a resource, it will issue an alert that the resource is in error. If the heartbeat is received again, the server will issue an event that the resource is fine and will lower the alert level.

10.3.1. Plug-ins

Plug-ins can listen to the openQRM generated events and take action. Events have keys which are hierarchical, so, for example event a.b.c is the child of event a.b. Plug-ins choose what key to listen to and will receive notification when events with that key, or a child key are created, as well as associated data about the event. Plug-ins can choose to be notified by having the server execute code they provide which can be Java code or an executable. For example, a plug-in can register to be notified on the Resource.State.Operation.Error event (which means the openQRM server thinks the resource is in error), and then run the send_sms_to_admin.sh script (provided by the plug-in)

11. Package Management

openQRM provides support for package management through plug-ins.

There are currently two plug-ins:

- Patcher – This plug-in supports updating a Linux system with either Yum or up2date. The user can configure which to use per VE. The plug-in creates a boot service that runs the desired client. In addition, the plug-in handles NFS shared images (see above), by applying the changes an upgrade has caused to all private directories (thus, all other resources using the shared image receive all changes). The user can control a list of files/directories in the private directory to exclude from upgrade.
- Patchlink integration – In process: http://www.patchlink.com/.