Red Hat Reference Architecture Series

Red Hat Enterprise Virtualization 3.0 (RHEV) Disaster Recovery

Site to Site Failover

John Herr, Senior Software Engineer
RHCA, RHCVA.

Version 1.0
January 2012

redhat.

1801 Varsity Drive™
Raleigh NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588
Research Triangle Park NC 27709 USA

The GPG fingerprint of the security@redhat.com key is:
CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

Send feedback to refarch-feedback@redhat.com

Table of Contents

1 Executive Summary

Recovering business critical systems during a site level disaster is often difficult for many large and small enterprise environments. With careful planning and adequate resources, the Red Hat Enterprise Virtualization solution can be brought up at a designated backup location with minimal downtime.

This paper demonstrates the process of failing a RHEV environment to a backup location during a site level disaster. Simulating a site to site LUN mirroring solution, three separate environments with running virtual machines are failed over to a backup location and brought up with minimal downtime.

- NFS Data Storage Domain
- iSCSI Data Storage Domain
- Fibre Data Storage Domain

A script to automate the recovery of the failed over environment is created with explanations of the script contents.

The goal of this paper is to provide the reader with an understanding of the tasks involved in recovering a RHEV environment during a disaster using both manual and scripted processes.

2 Red Hat Enterprise Virtualization

2.1 RHEV Hypervisor

A hypervisor is a computer software platform that allows multiple “guest” operating systems to run concurrently on a host computer. The guest virtual machines interact with the hypervisor which translates guest I/O and memory requests into corresponding requests for resources on the host computer.

Running fully virtualized guests, i.e., guests with unmodified guest operating systems, used to require complex hypervisors and previously incurred a performance penalty for emulation and translation of I/O and memory requests.

Over the last few years chip vendors Intel and AMD have been steadily adding CPU features that offer hardware enhancements to support virtualization. Most notable are:

1. First-generation hardware assisted virtualization: Removes the requirement for hypervisor to scan and rewrite privileged kernel instructions using Intel VT (Virtualization Technology) and AMD's SVM (Secure Virtual Machine) technology.

2. Second-generation hardware assisted virtualization: Offloads virtual to physical memory address translation to CPU/chip-set using Intel EPT (Extended Page Tables) and AMD RVI (Rapid Virtualization Indexing) technology. This provides significant reduction in memory address translation overhead in virtualized environments.

3. Third-generation hardware assisted virtualization: Allows PCI I/O devices to be attached directly to virtual machines using Intel VT-d (Virtualization Technology for directed I/O) and AMD IOMMU. Also, SR-IOV (Single Root I/O Virtualization) which allows special PCI devices to be split into multiple virtual devices. This provides significant improvement in guest I/O performance.

The great interest in virtualization has led to the creation of several different hypervisors. However, many of these pre-date hardware-assisted virtualization, and are therefore some-what complex pieces of software. With the advent of the above hardware extensions, writing a hypervisor has become significantly easier and it is now possible to enjoy the benefits of virtualization while leveraging existing open source achievements to date.

Red Hat Enterprise Virtualization uses the Kernel-based Virtual Machine (KVM)[1], which turns Linux into a hypervisor. Red Hat Enterprise Linux 5.4 provided the first commercial-strength implementation of KVM, which is developed as part of the upstream Linux community. RHEV 3.0 uses the RHEL 6 KVM hypervisor, and inherits performance, scalability and hardware support enhancements from RHEL 6.

1 http://www.redhat.com/promo/qumranet/

2.2 Red Hat Enterprise Virtualization

Virtualization offers tremendous benefits for enterprise IT organizations – server consolidation, hardware abstraction, and internal clouds deliver a high degree of operational efficiency.

Red Hat Enterprise Virtualization (RHEV) combines the KVM hypervisor (powered by the Red Hat Enterprise Linux kernel) with an enterprise grade, multi-hypervisor management platform that provides key virtualization features such as live migration, high availability, power management, and virtual machine life cycle management. Red Hat Enterprise Virtualization delivers a secure, robust virtualization platform with unmatched performance and scalability for Red Hat Enterprise Linux and Windows guests.

Red Hat Enterprise Virtualization consists of the following two components:

- **RHEV Manager (RHEV-M):** A feature-rich virtualization management system that provides advanced capabilities for hosts and guests.
- **RHEV Hypervisor:** A modern, scalable, high performance hypervisor based on RHEL KVM. It can be deployed as RHEV-H, a small footprint secure hypervisor image included with the RHEV subscription, or as a RHEL server (purchased separately) managed by RHEV-M.

A **host** is a physical server which provides the CPU, memory, and connectivity to storage and networks that are used for the virtual machines (VM). The local storage of the standalone host is used for the RHEV-H executables along with logs and enough space for ISO uploads.

A **cluster** is a group of hosts of similar architecture. The requirement of similar architecture allows a virtual machine to be migrated from host to host in the cluster without having to shut down and restart the virtual machine. A cluster consists of one or more hosts, but a host can only be a member of one cluster.

A **data center** is a collection of one or more clusters that have resources in common. Resources that have been allocated to a data center can be used only by the hosts belonging to that data center. The resources relate to storage and networks.

A **storage domain** is a shared or local storage location for virtual machine image files, import/export or for ISO images. Storage domain types supported in RHEV 3.0 are NFS, iSCSI, Fibre Channel, and local disk storage.

The RHEV **network** architecture supports both virtual machine traffic and traffic among RHEV hypervisors and the RHEV-M server. All hosts have a network interface assigned to the logical network named *rhevm*. This network is used for the communications between the hypervisor and the manager. Additional logical networks are created on the data center and applied to one or more clusters. To become operational, the host attaches an interface to the local network. While the actual physical network can span across data centers, the logical network can only be used by the clusters and hosts of the creating data center.

Figure 2.2.1:RHEV Environment provides a graphical representation of a typical Red Hat Enterprise Virtualization environment with each component listed.

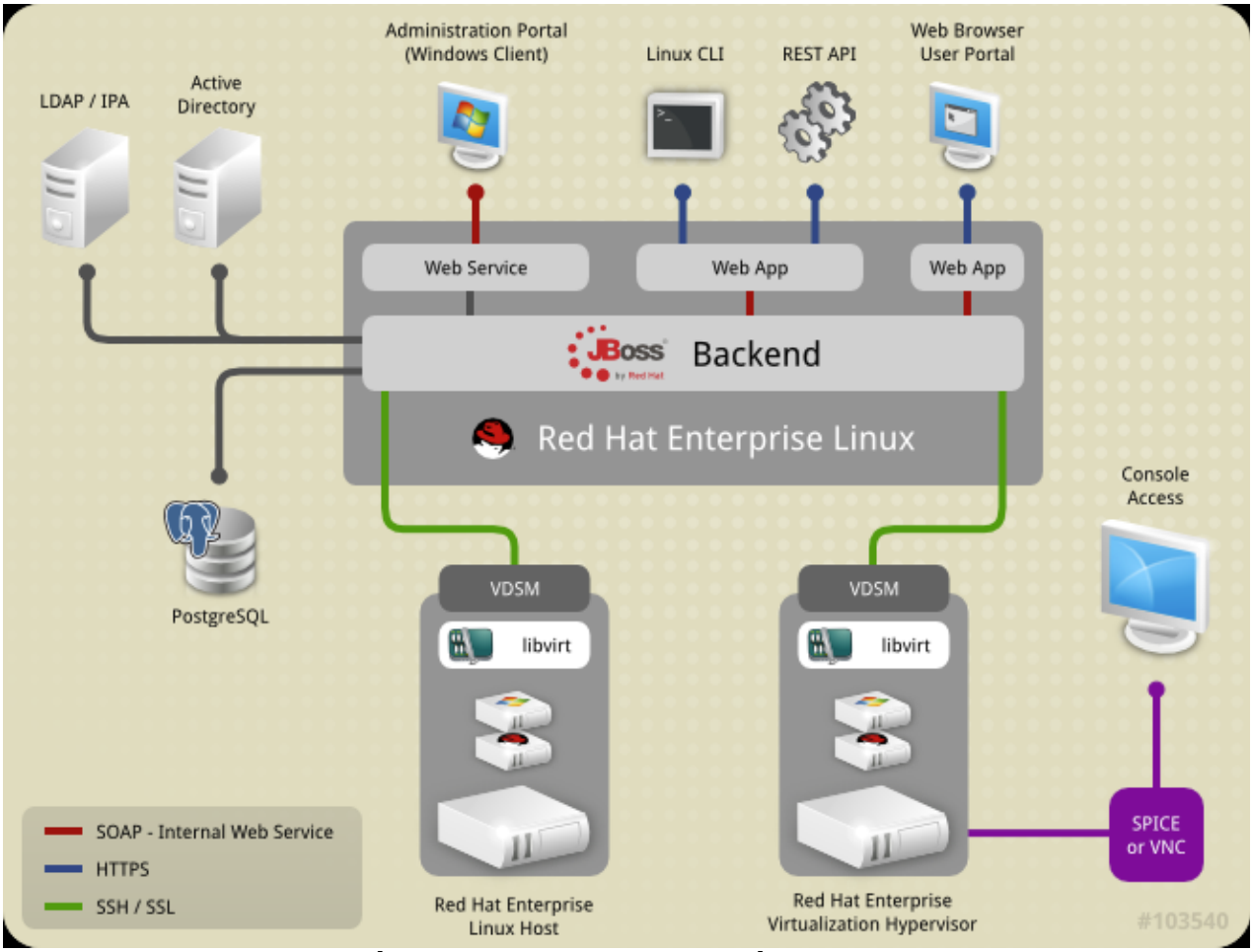

Figure 2.2.1:RHEV Environment

3 Reference Architecture Environment

This environment used in the writing of this reference architecture simulates two individual sites, *sitea.example* and *siteb.example*. Each site is a separate network with different network addresses and domain names.

Each site consists of its own DNS server, DHCP server, NFS server, iSCSI server, RHEV Manager, and hypervisor.

The two sites are connected together and to a public network via a gateway system. This gateway system uses network address translation (NAT) to map public IP addresses to addresses of systems at *sitea.example* and *siteb.example*.

The **Figure 3.1: Reference Architecture Environment** depicts the environment being used and is followed by a brief description of the systems and processes involved.

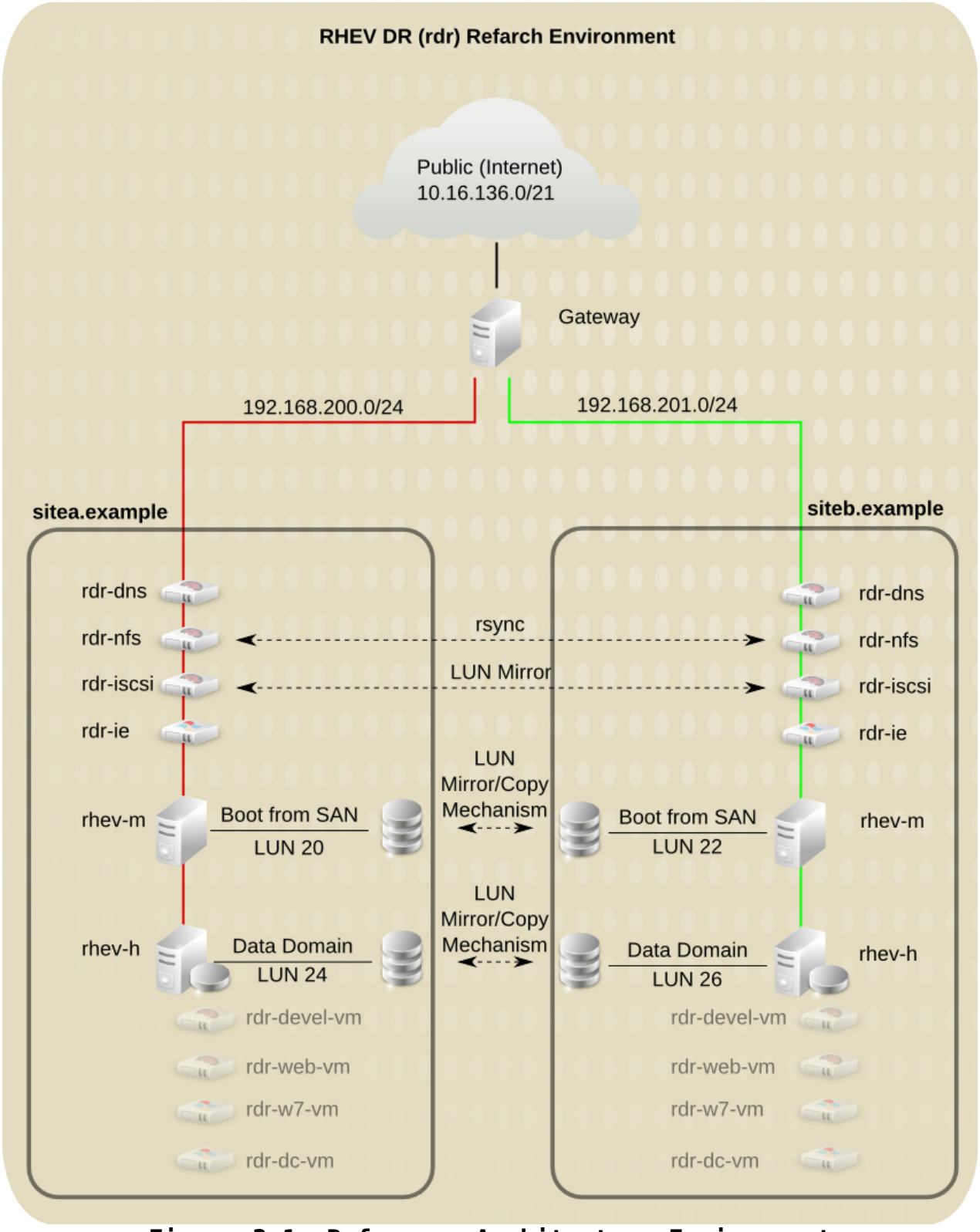

Figure 3.1: Reference Architecture Environment

Gateway	This virtual machine routes traffic between the *public*, *sitea.example*, and *siteb.example* networks. This system also provides Static NAT translation of public IP addresses to certain systems on the *sitea.example* and *siteb.example* networks.

Each site contains the following systems. Each system is isolated to its respective network and provides services to that network only. The following virtual machines exist on a seperate hypervisor and not the hypervisors used by the RHEV Environment in this paper.

rdr-dns	This virtual machine provides DNS resolution and DHCP for the *sitea.example* network.
rdr-nfs	This virtual machine exports NFS shares for the Export Domain, ISO Domain, and Data Domain. The rsync command is used within a cron job on *rdr-nfs.siteb.example* to copy the data from the Export Domain and ISO Domain on *sitea.example* to *siteb.example*.LUN mirroring is simulated on the virtual disk backing the data domain.
rdr-iscsi	This virtual machine provides an iSCSI target LUN for use as a Data Domain. This system has two virtual disks presented to it. The first virtual disk is used for the operating system. The second virtual disk used as the iSCSI target. LUN Mirroring for the secondvirtual disk is simulated by using the **dd** command on the hypervisor system to copy the virtual disk of *rdr-iscsi.sitea.example* to *rdr-iscsi.siteb.example*.
rdr-ie	This virtual machine is used to connect to the RHEV Manager.
rhev-m	This physical system is the RHEV Manager. This system is connected to fibre storage and is presented a single LUN. The OS is installed on this LUN. Mirroring of the presented LUN is simulated between the *rhev-h.sitea.example* and *rhev-h.siteb.example* systems.
rhev-h	This physical system is the RHEV Hypervisor. The OS is installed to local storage on this system. This system is presented a single fibre LUN when using a fibre Data Center. LUN Mirroring LUN mirroring for the rhev-m and rhev-h systems is simulated by presenting the both the *sitea.example* and *siteb.example* LUNS to a single system. The **dd** command is then used to copy the data from the *sitea.example* LUN to the *siteb.example* LUN. LUN mirroring for the disk backing the NFS data domain and the iSCSI data domains is simulated using the **dd** command on the hypervisor hosting the *rdr-nfs* and *rdr-iscsi* virtual machines.

The following virtual machines exist within the RHEV Environment used in this paper. These virtual machines are migrated during the site level failover.

rdr-devel-vm	This virtual machine uses the *rdr-dc-vm* for login account information.
rdr-web-vm	This virtual machine uses the *rdr-dc-vm* for login account information. It also runs httpd to provide web services.
rdr-w7-vm	This virtual machine is a Windows 7 desktop client.
rdr-dc-vm	This virtual machine is a domain controller. It is used by the *rdr-devel-vm*, *rdr-web-vm*, and *rdr-w7-vm* to provide login account information.

Server	Export	Storage Domain Type
rdr-nfs.sitea.example	/exports/isodomain	ISO
	/exports/exports_configured	Export
	/exports/datadomain	Data
rdr-nfs.siteb.example	/exports/ISO	ISO
	/exports/EXPORT	Export
	/exports/DATA	Data

Table 3.1:NFS Exports

System	Specifications
Gateway [Virtual Machine]	RHEL 6.1.0.2 Kernel 2.6.32-131.17.1.el6.x86_64
	1 x QEMU Virtual CPU version (cpu64-rhel6) @ 2.67 GHz
	512 MB Memory
	2 x VirtIO Disk File @ 8 GB Operating System @ 30GB Installation Repo
	3 x VirtIO Network Adapters
rdr-dns.site{a,b}.example [Virtual Machine]	RHEL 6.1.0.2 Kernel 2.6.32-131.17.1.el6.x86_64
	1 x QEMU Virtual CPU version (cpu64-rhel6) @ 2.67 GHz
	512 MB Memory
	1 x VirtIO Disk File @ 8 GB
	2 x VirtIO Network Adapters
rdr-nfs.site{a,b}.example [Virtual Machine]	RHEL 6.1.0.2 Kernel 2.6.32-131.17.1.el6.x86_64
	1 x QEMU Virtual CPU version (cpu64-rhel6) @ 2.67 GHz
	512 MB Memory
	4 x VirtIO Disk Files @ 8 GB Operating System @ 100 GB Not Used @ 100 GB ISO Domain @ 100 GB Exports Domain
	2 x VirtIO Network Adapters
rdr-iscsi.site{a,b}.example [Virtual Machine]	RHEL 6.1.0.2 Kernel 2.6.32-131.17.1.el6.x86_64
	1 x QEMU Virtual CPU version (cpu64-rhel6) @ 2.67 GHz
	512 MB Memory
	2 x VirtIO Disk Files @ 8 GB Operating System @ 200 GB Data Domain
	2 x VirtIO Network Adapters

System	Specifications
rdr-ie.site{a,b}.example [Virtual Machine]	Microsoft Windows XP Professional SP3
	1 x Virtual CPU Qemu Virtual CPU version (cpu64-rhel6) @ 2.67 GHz
	2 GB Memory
	1 x Virtual IDE Disk File @ 20 GB
	1 x Virtual Network Adapter
rdr-rhev-m.sitea.example [HP Proliant DL580 G5]	RHEL 6.2
	rhevm 3.0.0_0001-62.el6
	4 x Quad Core Intel XEON X7350 CPUs @2.93 GHz
	64 GB Memory
	1 x LUN @ 50 GB
	2 x Single Port Broadcom NetXtreme BCM5708 Network Adapter
rdr-rhev-m.siteb.example [HP Proliant DL370 G6]	RHEL 6.2
	rhevm 3.0.0_0001-62.el6
	2 x Quad Core Intel XEON W5580 CPUs @3.2 GHz
	48 GB Memory
	1 x LUN @ 50 GB
	1 x Quad Port HP NC375i Network Adapter
rdr-rhev-h.site{a,b}.example [HP Proliant BL460c G6]	RHEv Hypervisor
	2 x Quad Core Intel Xeon CPU X550 @2.67 GHz
	48 GB Memory
	2 x 146 GB SAS internal disk drives (mirrored)
	2 x Qlogic ISP2532-based 8Gb FC HBA 1 x 200 GB LUN
	2 x Broadcom NetXtreme II BCM57711E Flex-10 10Gb Ethernet Controller
rdr-devel-vm [Virtual Machine]	RHEL 6.1 Kernel 2.6.32-131.17.1.el6.x86_64
	1 x Virtual CPU
	512 MB Memory
	1 x VirtIO Disk @ 8GB
	1 x VirtIO Network Adapter
rdr-web-vm	RHEL 6.1

System	Specifications
[Virtual Machine]	Kernel 2.6.32-131.17.1.el6.x86_64
	1 x Virtual CPU
	512 MB Memory
	1 x VirtIO Disk @ 8GB
	1 x VirtIO Network Adapter
rdr-w7-vm [Virtual Machine]	Microsoft Windows 7
	1 x Virtual CPU
	2 GB Memory
	1 x VirtIO Disk @ 20GB
	1 x VirtIO Network Adapter
rdr-dc-vm [Virtual Machine]	Microsoft Windows Server 2008 R2
	1 x Virtual CPU
	2 GB Memory
	1 x VirtIO Disk @ 40GB
	1 x VirtIO Network Adapter

Table 3.2: Server Hardware Configuration

System	Hardware Address	Interface	IP Address
rdr-rhevm.sitea.example	00:18:71:eb:a0:39	eth0	
	00:1e:0b:ce:42:78	eth1	192.168.200.40
	00:1e:0b:ce:42:7a	eth2	
rdr-rhevm.siteb.example	00:25:b3:a9:b0:01	eth0	
	00:25:b3:a9:b0:00	eth1	192.168.201.40
	00:25:b3:a9:b0:02	eth2	
	00:25:b3:a9:b0:03	eth3	

Table 3.3: RHEV Manager NIC Assignments

System	Hardware Address	Network
rdr-rhevh.sitea.example	00:17:a4:77:24:34	rhevm
	00:17:a4:77:24:36	public
rdr-rhevh.siteb.example	00:17:a4:77:24:38	rhevm
	00:17:a4:77:24:3a	public

Table 3.4: RHEV Hypervisor NIC Assignments

4 REST API

The Red Hat Enterprise Virtualization Manager provides the REST API to access and modify virtualization environments. The REST API, may be used to perform tasks associated with a site level fail over. Accessing the REST API may be done using programming languages, such as Python and PERL,as well as the **curl** command. This reference architecture uses XML code and the curl command.

Calls to the REST API should be done using a secure (HTTPS) connection. A certificate file from the server is needed before calls using the secure connection can be made. The following **curl** command downloads the certificate and places it in a file called *rhevm.cer*.

```
# curl -# -o rhevm.cer http://rdr-rhevm.sitea.example:8080/ca.crt
######################################################################## 100.0%
```

When the **curl** command is used to submit requests to the REST API, the request must specify the certificate to use, the content type of the submitted data, the type of request being made, authentication information, and the URL to submit the information to. The certificate is specified using the **--cacert** option.

The type of content being submitted is specified using the **--header** option.

The type of request being submitted is specified using the **--request** option. The request types can be **POST**, **PUT**, **GET**, and **DELETE**.

Authentication information is specified using the **--user** option. This options takes the user name in the form of ***user@domain*:*password***.

The following **curl** command queries information about the cluster and returns the output as XML.[2]

```
# curl --cacert rhevm.cer \
  --header "Content-Type: application/xml"    \
  --request "GET"                             \
  --user "admin@internal:[PASSWORD]"          \
  https://rdr-rhevm.sitea.example:8443/api

[ ... XML OUTPUT TRUNCATED ... ]
```

Some calls to the REST API require that XML code is submitted with the request. This XML code can be submitted using the **--data** option. This option takes one argument. If the argument begins with an @ symbol, then the remainder of the argument is taken as a file name that contains the XML code to be transmitted. If the argument does not begin with an @ symbol, then it is assumed to be XML code that is included within the command.

2 http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Virtualization/3.0/html-single/REST_API_Guide/index.html

The following command submits a request to approve a new host. This command uses a file that contains the XML code needed.

```
# echo “<action/>” > approve.xml

# curl --silent --cacert rhevm.cer      \
  --header "Content-Type: application/xml"         \
  --request "POST"                                 \
  --user "admin@internal:[PASSWORD]"               \
  --data @approve.xml                              \
  https://rdr-rhevm.sitea.example:8443/api/hosts/5ee01ada-0016-11e1-a866-
001e0bce4278/approve
```

The following command submits the same request to approve a new host, but specifies the XML code within the command.

```
# curl --silent --cacert rhevm.cer      \
  --header "Content-Type: application/xml"         \
  --request "POST"                                 \
  --user "admin@internal:[PASSWORD]"               \
  --data "<action/>"                               \
  https://rdr-rhevm.sitea.example:8443/api/hosts/5ee01ada-0016-11e1-a866-
001e0bce4278/approve
```

4.1 Parsing Output

The output returned by the REST API is in XML format. XML format can be difficult to parse and read. The **xpath** command, provided by the **perl-XML-XPath** package, allows the output to be easily parsed. The output is parsed using a syntax similar in format to accessing files and directories on a filesystem. The outermost tag is treated as the parent directory entry. Each sub-tag is accessed as if it was a subdirectory of its parent tag.

The following examples demonstrate the usage of **xpath** as it is used in this reference architecture. For more information on **xpath** and its capabilities, see the documentation provided by the **perl-XML-XPath** package or other online resources.

For an example, consider a file called *host.xml* that contains the following XML code.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<hosts>
    <host id="4f2b4996-12ba-11e1-90af-001871eba039"
href="/api/hosts/4f2b4996-12ba-11e1-90af-001871eba039">
        <name>rdr-rhevh.sitea.example</name>
        <address>192.168.200.42</address>
        <status>
            <state>non_responsive</state>
        </status>
        <cluster id="ddbf36dc-12b9-11e1-ae84-001871eba039"
href="/api/clusters/ddbf36dc-12b9-11e1-ae84-001871eba039"/>
    </host>
    <host id="805bb14c-2505-11e1-b809-0025b3a9b001"
href="/api/hosts/805bb14c-2505-11e1-b809-0025b3a9b001">
        <name>rdr-rhevh.siteb.example</name>
        <address>192.168.201.42</address>
        <status>
            <state>pending_approval</state>
        </status>
        <cluster id="99408929-82cf-4dc7-a532-9d998063fa95"
href="/api/clusters/99408929-82cf-4dc7-a532-9d998063fa95"/>
    </host>
</hosts>
```

To determine the name of the hosts in the cluster, executing the following command returns the values in the <*NAMES*> tags under the <*HOSTS*> and <*HOST*> parent tags.

```
# cat host.xml | xpath /hosts/host/name
Found 2 nodes:
-- NODE --
<name>rdr-rhevh.sitea.example</name>-- NODE --
<name>rdr-rhevh.siteb.example</name>
```

Elements of a tag are accessed using @ character. Executing the following returns the element identified as **id** from within the <*HOST*> tag.

```
# cat host.xml | xpath /hosts/host/@id
Found 2 nodes:
-- NODE --
 id="4f2b4996-12ba-11e1-90af-001871eba039"-- NODE --
 id="805bb14c-2505-11e1-b809-0025b3a9b001"
```

Multiple tags and elements are retrieved at the same time by using the pipe character between the queries. The entire query must be enclosed in quotes to prevent the bash shell from interpreting the pipe character as a bash reserved character. Executing the following returns both the <*NAME*> tag and id element.

```
# cat host.xml | xpath "/hosts/host/name | /hosts/host/@id"
Found 4 nodes:
-- NODE --
 id="4f2b4996-12ba-11e1-90af-001871eba039"-- NODE --
<name>rdr-rhevh.sitea.example</name>-- NODE --
 id="805bb14c-2505-11e1-b809-0025b3a9b001"-- NODE --
<name>rdr-rhevh.siteb.example</name>
```

A specific tag can be retrieved by specifying an element.

```
# cat host.xml | xpath "/hosts/host[@id=\"805bb14c-2505-11e1-b809-
0025b3a9b001\"]/name"
Found 1 nodes:
-- NODE --
<name>rdr-rhevh.siteb.example</name>
```

4.2 Command Line Shortcuts

The **curl** command, when used with the REST API, can become long and confusing. The variable and alias capabilities of bash allow the command line to be shortened and simplified. Shortening the command line makes the command line more readable and also helps prevent typos.

The common **curl** command line options used in this reference architecture are assigned to an alias in bash as follows.

```
# alias curl='curl --silent --cacert rhevm.cer --header "Content-Type:
application/xml" --user "admin@internal:[PASSWORD]"'
```

Since all the URIs to access the REST API begin the same, the base URI is assigned to a variable in bash to further reduce the length and complexity of the command line.

```
# APIBASE="https://rdr-rhevm.sitea.example:8443/api"
```

Many of the calls to the REST API require sending one or more unique identifiers (UIDs) to complete the call successfully. These UIDs can be lengthy and difficult to type. Using the associative array capabilities of bash, a more human readable name can be created and used instead of typing the UID directly. This greatly reduces the complexity of the commands.

In-depth discussion of associative arrays in bash is beyond the scope of this document. See the bash man page or online bash tutorials for more information.

Associative arrays in bash must be declared priot to use. The following example declares a bash associative array called *EX* and assigns UIDs to keys called **host1** and **cluster**.

```
# declare -A EX

# EX[host1]="5ee01ada-0016-11e1-a866-001e0bce4278"

# EX[cluster]="32421bf1-2344-4342-4cce-23e8ce90248a"
```

The individual values for a key are accessed as follows.

```
# echo ${EX[host1]}
5ee01ada-0016-11e1-a866-001e0bce4278

# echo ${EX[cluster]}
32421bf1-2344-4342-4cce-23e8ce90248a
```

All the keys and all the values are accessed as shown below.

```
# echo ${!EX[@]}
cluster host1

# echo ${EX[@]}
32421bf1-2344-4342-4cce-23e8ce90248a 5ee01ada-0016-11e1-a866-001e0bce4278
```

Using the methods described above, the command to approve a new host changes from a complex looking command to a command that is shorter and easier to type and read.

The original complex command to approve a host.

```
curl --silent --cacert rhevm.cer --header "Content-Type: application/xml" \
>  --user "admin@internal:[PASSWORD]"     \
>  --data “<action><cluster id=\”32421bf1-2344-4342-4cce-
23e8ce90248a\”/></action>”     \
>  https://rdr-rhevm.sitea.example:8443/api/hosts/5ee01ada-0016-11e1-a866-
001e0bce4278/approve
```

The shortened command to approve a host.

```
# curl –request “POST”       \
>  --data “<action><cluster id=\”${EX[cluster]}\”></action>”    \
>  ${APIBASE}/hosts/${EX[host1]}/approve
```

5 RHEV Manager Database

Information about the configuration and state of the cluster is stored in a PostgreSQL database called *rhevm*. This database is stored on the Red Hat Enterprise Virtualization Manager. During a site level failover, the database is modified to fix information about the storage domain configuration.

Modifications to the database can be made by sending SQL statements to the PostgreSQL database, through the REST API, or by using the web portal. While all modifications can be made manually, it is prone to user errors that can render the environment unusable. Manually updating the database is not supported by Red Hat. Utilizing the REST API and web portal is used whenever possible. There are only a few instances where manual manipulation of the database is necessary.

Manipulations to the database are made using the **root** account on the RHEV Manager and the **psql** command. The RHEV Manager service (**jbossas**) must be stopped before any changes are manually made to the database. Care must be taken when manually manipulating the database since it can cause data loss if done incorrectly. As always, creating a backup of the database before making any changes is highly recommended.

The **pg_dump** command is used to backup the database and the **pg_restore** command is used to restore the database if needed. The following example shows how to backup the database using the **pg_dump** command.

```
# pg_dump --format custom --username postgres --file rhevm_db.pgdump rhevm
```

The **jbossas** service must be stopped before restoring the *rhevm* database. It is best to remove the database before restoring it. The **pg_restore** command is used to restore the database and the **--clean** option removes the the database before restoring it.

```
# service jbossas stop
Stopping jbossas:                                          [  OK  ]

# pg_restore --dbname rhevm --clean --username postgres rhevm_db.pgdump
```

Querying and manipulating the database is done using the **psql** command. SQL statements are passed via standard input (STDIN) or by using the **--command** option. The following example views the contents of the **luns** table in the *rhevm* database.

```
# echo "select * from luns;" | psql --dbname rhevm --username postgres
           phisical_volume_id           |    lun_id     |
volume_group_id             |      serial       | lun_mapping | vendor_id |
product_id  | device_size
---------------------------------------+--------------
+---------------------------------------+------------------+------------
+-----------+--------------+-------------
 VfXIWw-fU2H-utOS-uS1D-LypB-eh4j-RTdWnl | 1IET_00010001 | DIv0nZ-atbO-uLm7-
Hq5q-5XRQ-eM4x-KaNFmR | SIET_VIRTUAL-DISK |           1 | IET       |
VIRTUAL-DISK |         199
```

6 Network Preparation

Correct network configuration is crucial to ensure the failed over environment functions as desired. IP addresses must be changed to reflect the new environment as does name resolution. This applies to physical servers as well as the virtual machines.

6.1 DHCP

DHCP is used to provide the network configuration to the systems in the environment and allows for an easier failover. Without DHCP, it would be necessary to connect to each physical and virtual system in the environment and manually reconfigure its network configuration. This includes IP information, routing, and name resolution.

The DHCP servers used in this reference architecture supply IP addresses, FQDN host names, DNS servers, and default gateways to each physical server and virtual machine.

All the systems at SiteA receive IP addresses on the 192.168.200.0/24 network and fully qualified domain names on the *sitea.example* domain. All the systems, except the RHEV Manager system, at SiteB receive IP adresses on the 192.168.201.0/24 network and fully qualified domain names on the *siteb.example* network.

The RHEV Manager system receives an IP address on the 192.168.201.0/24 network, but it receives a fully qualified domain name on the SiteA network. The RHEV Manager requires the same FQDN due to the certificates it created at the time it was installed. The certificates are created using the hosts FQDN.

The entry in the DHCP configuration file for the RHEV Manager provides the server with IP addresses, DNS servers, and a gateway on the SiteB network, but it provides it an FQDN on the *sitea.example* domain.

6.2 DNS

The RHEV Manager must have a DNS entry that resolves its original FQDN to the new IP address and back since the certificates use the original FQDN. Similar considerations may be needed for the virtual machines in the environment as well, depending on its function and the applications it runs.

Certificate validation is an example when this might be needed. Applications and scripts that access the virtual machines by using their FQDNs are another example.

The system names used after the failover remain the same, but the FQDNs change. This allows scripts and applications that use the DNS resolvable short name to continue to work if DNS is configured properly. For example: *rdr-nfs.sitea.example* becomes *rdr-nfs.siteb.example*.

The DNS server at *siteb.example* contains forward and reverse zone definitions for the systems on the *siteb.example* domain. It also contains a forward and reverse zone definition for the *sitea.example* domain. The configuration file for the *sitea.example* zones only contain entries for *rdr-rhevm* and *rdr-dns*. Both definitions point to IP addresses on the SiteB network.

7 Failover

The following section walks through the failover of three Red Hat Enterprise Virtualization environments. The configuration of the environments are identical except for the type of Data Storage Domain used. Refer to **Section 3: Reference Architecture Environment** for the configuration of the environment. The process to failover the environments is identical with the exception of the Data Storage Domains. For brevity, the duplicated tasks have been consolidated and not repeated three times.

For each environment, the datacenter, cluster, and the SiteA hypervisor are configured. All the virtual machines in the environment are powered on and functioning as well. Each has the identical Export Storage Domain and ISO Storage Domain configured. The Export and ISO Storage Domains use rsync to copy the data from the NFS export directories at SiteA to SiteB.

After the failover occurs, the RHEV Hypervisor at SiteB is installed, configured, and added to the SiteB cluster.

7.1 Failover Scenarios

7.1.1 iSCSI Data Domain

The first simulated failover uses an iSCSI target as the Data Storage Domain. To simulate a site wide failure, the virtual machines acting as the Storage Domains are halted. The virtual machines *rdr-nfs.sitea.example* and *rdr-iscsi.sitea.example* provide NFS exports and an iSCSI target to be used as Storage Domains. These virtual machines are running on a system outside the RHEV environment and are halted by issuing a **`virsh destroy`** command on the system running them. This does not give the systems time to gracefully power down.

The physical server running the RHEV Manager and the RHEV Hypervisor have the power removed at the same time as the **`virsh destroy`** command is executed against the virtual machines.

LUN mirroring is simulated on the iSCSI server by using the **`dd`** command to copy the logical volume that backs the iSCSI target on the *rdr-iscsi.sitea.example* virtual machine to the logical volume that backs the iSCSI target on the *rdr-iscsi.siteb.example* system.

LUN mirroring is simulated on the RHEV Manager system by removing the LUN mapping to the *rdr-rhevm.sitea.example* system and mounting it and the LUN used for the *rdr-rhevm.siteb.example* system onto a single system. The **`dd`** command is used to copy one LUN to the other. The LUNS are then mapped to the correct systems.

All the systems at SiteB are then powered on.

7.1.2 Fibre Data Domain

The second simulated failover uses a fibre LUN as the Data Storage Domain. To simulate a site wide failure, the virtual machines acting as the Storage Domains are halted. The virtual machine *rdr-nfs.sitea.example* provides NFS exports to be used as Storage Domains. These virtual machines are running on a system outside the RHEV environment and are halted by issuing a **`virsh destroy`** command on the system running them. This does not give the systems time to gracefully power down.

The physical server running the RHEV Manager and the RHEV Hypervisor have the power removed at the same time as the **`virsh destroy`** command is executed against the virtual machines.

LUN mirroring is simulated on the RHEV Manager and RHEV Hypervisor systems by removing the LUN mapping to the systems and mounting the LUNs used at SiteA and the LUNs used at SiteB onto a single system. The **dd** command is used to copy one LUN to the other. The LUNS are then mapped to the correct systems.

All the systems at SiteB are then powered on.

7.1.3 NFS Data Domain

The third simulated failover uses an NFS export as the Data Storage Domain. To simulate a site wide failure, the virtual machines acting as the Storage Domains are halted. The virtual machine *rdr-nfs.sitea.example* provide NFS exports to be used as Storage Domains. These virtual machines are running on a system outside the RHEV environment and are halted by issuing a **`virsh destroy`** command on the system running them. This does not give the systems time to gracefully power down.

The physical server running the RHEV Manager and the RHEV Hypervisor have the power removed at the same time as the **`virsh destroy`** command is executed against the virtual machines.

LUN mirroring is simulated on the NFS server for the Data Storage Domain by using the **dd** command to copy the logical volume that backs the exported directory on the *rdr-nfs.sitea.example* virtual machine to the logical volume that backs the exported directory on the *rdr-nfs.siteb.example* system.

LUN mirroring is simulated on the RHEV Manager system by removing the LUN mapping to the system and mounting the LUN used at SiteA and the LUN used at SiteB onto a single system. The **dd** command is used to copy one LUN to the other. The LUNS are then mapped to the correct systems.

All the systems at SiteB are then powered on.

7.2 RHEV Manager

Network configuration on the RHEV Manager may need to be modified on the failed over RHEV Manager. This is due to the different network adapters and hardware addresses that are in the new system.

The **udev** rules and interface configuration files need to be modified to account for the new hardware addresses. This helps avoid modifying the firewalls, reconfiguring bonding, and possibly other configurations.

View the current running network configuration using the **ip addr** command. The output shows the current interfaces start enumerating at eth3 instead of eth0. The **udev** rules need to be modified to adjust the nic enumeration.

```
# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth5: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 00:25:b3:a9:b0:00 brd ff:ff:ff:ff:ff:ff
3: eth3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 00:25:b3:a9:b0:01 brd ff:ff:ff:ff:ff:ff
4: eth6: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 00:25:b3:a9:b0:02 brd ff:ff:ff:ff:ff:ff
5: eth4: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 00:25:b3:a9:b0:03 brd ff:ff:ff:ff:ff:ff
```

Edit the */etc/udev/rules.d/70-persistent-net.rules* file. Each network adapter is configured using a single line in the configuration file. The file below shows the single line wrapped across three lines.

```
# This file was automatically generated by the /lib/udev/write_net_rules
# program, run by the persistent-net-generator.rules rules file.
#
# You can modify it, as long as you keep each rule on a single
# line, and change only the value of the NAME= key.

# PCI device 0x8086:0x10b9 (e1000e)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:18:71:eb:a0:39", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth0"

# PCI device 0x14e4:0x164c (bnx2)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:1e:0b:ce:42:78", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth1"

# PCI device 0x14e4:0x164c (bnx2)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:1e:0b:ce:42:7a", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth2"

# PCI device 0x4040:0x0100 (netxen_nic)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:25:b3:a9:b0:01", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth3"

# PCI device 0x4040:0x0100 (netxen_nic)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:25:b3:a9:b0:03", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth4"

# PCI device 0x4040:0x0100 (netxen_nic)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:25:b3:a9:b0:00", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth5"

# PCI device 0x4040:0x0100 (netxen_nic)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:25:b3:a9:b0:02", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth6"
```

Each network adapter has an entry in the file. This includes the current adapters as well as the original adapters. The entries for the original adapters need to be removed or commented. The entries for the new adapters need to be enumerated correctly. The information from **Table 3.3: RHEV Manager NIC Assignments**, shows that the network interface connected to the network on the original RHEV Manager is eth1. The same table also shows that the network adapter with the hardware address of 00:25:b3:a9:b0:00 should is connected to the network.

Modify the file to ensure the network is configured correctly.

```
# This file was automatically generated by the /lib/udev/write_net_rules
# program, run by the persistent-net-generator.rules rules file.
#
# You can modify it, as long as you keep each rule on a single
# line, and change only the value of the NAME= key.

# PCI device 0x8086:0x10b9 (e1000e)
# SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:18:71:eb:a0:39", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth0"

# PCI device 0x14e4:0x164c (bnx2)
# SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:1e:0b:ce:42:78", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth1"

# PCI device 0x14e4:0x164c (bnx2)
# SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:1e:0b:ce:42:7a", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth2"

# PCI device 0x4040:0x0100 (netxen_nic)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:25:b3:a9:b0:01", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth3"

# PCI device 0x4040:0x0100 (netxen_nic)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:25:b3:a9:b0:03", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth2"

# PCI device 0x4040:0x0100 (netxen_nic)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:25:b3:a9:b0:00", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth1"

# PCI device 0x4040:0x0100 (netxen_nic)
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*",
ATTR{address}=="00:25:b3:a9:b0:02", ATTR{type}=="1", KERNEL=="eth*",
NAME="eth0"
```

The interface configuration files may contain hardware specific configuration. Issue the following commands to see how many interface configuration files there are and view the contents of each.

```
# ls -1 /etc/sysconfig/network-scripts/ifcfg-*
/etc/sysconfig/network-scripts/ifcfg-eth0
/etc/sysconfig/network-scripts/ifcfg-eth1
/etc/sysconfig/network-scripts/ifcfg-eth2
/etc/sysconfig/network-scripts/ifcfg-lo

# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE="eth0"
HWADDR="00:18:71:EB:A0:39"
NM_CONTROLLED="yes"
ONBOOT="no"

# cat /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE="eth1"
BOOTPROTO="dhcp"
HWADDR="00:1E:0B:CE:42:78"
IPV6INIT="yes"
MTU="1500"
NM_CONTROLLED="yes"
ONBOOT="yes"
TYPE="Ethernet"

# cat /etc/sysconfig/network-scripts/ifcfg-eth2
DEVICE="eth2"
HWADDR="00:1E:0B:CE:42:7A"
NM_CONTROLLED="yes"
ONBOOT="no"
```

The configuration files define the **HWADDR** variable. The variable is used to ensure an interface configuration file is assigned to the correct network adapter. This prevents the configuration files from using the new network adapters.

Comment the lines defining the **HWADDR** variable.

```
# sed --in-place --expression 's/^HWADDR/# HWADDR/' /etc/sysconfig/network-
scripts/ifcfg-eth*

# cat /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE="eth0"
# HWADDR="00:18:71:EB:A0:39"
NM_CONTROLLED="yes"
ONBOOT="no"
```

The network configuration should now be fixed. Reboot the system to ensure all configurations are used.

```
# init 6
```

After rebooting the system, the RHEV Manager is accessible through its web portal.

7.3 Storage Domains

The REST API does not allow modification of storage domain attributes. ISO Storage Domains and Export Storage Domains can be imported into the environment, but Data Storage Domains require the RHEV Managers database to be manually updated. Importing the ISO and Export Storage Domains requires the metadata stored on the storage domain to be manually modified. Manually updating the database will be done for all storage domains.

7.3.1 Red Hat Enterprise Virtualization Manager service

Before manipulating any database entries, the Red Hat Enterprise Virtualization Manager service must be stopped. Failure to stop the service can result in unpredictable behavior.

The service is called **`jbossas`** and is stopped using the service command.

```
# service jbossas stop

Stopping jbossas:                                          [  OK  ]
```

7.3.2 Database Tables

There are four tables involved when manually manipulating the database. The tables are the **storage_domain_static**, **storage_server_connections**, **lun_storage_server_connection_map**, and the **luns** tables. The tables needing modification and the modifications themselves depend on the type of storage being used.

7.3.2.1 storage_domain_static table.

The **storage_domain_static** table holds the storage domain names, UIDs that map to entries in the **storage_server_connections** and **luns** tables, as well as other information.

This table does not need any updates made to it, it is used for the information contained it in only. The table contains the following columns:

- id
- storage
- storage_name
- storage_domain_type
- storage_type
- storage_domain_format_type
- _create_date
- _update_date

Only the **storage** and **storage_name** columns are needed when manipulating the database. The **storage_name column** contains the name of the storage domain as it is presented in the user interface. The **storage column** contains a UID which maps to an entry in either the **storage_server_connections** or the **luns** table.

7.3.2.2 storage_server_connections

The **storage_server_connections** table contains information about network attached storage such as NFS and iSCSI.

This table gets updated when modifying the connection parameters for network attached storage. The table contains the following columns:

- id
- connection
- user_name
- password
- iqn
- port
- portal
- storage_type

7.3.2.3 lun_storage_server_connection_map

The **lun_storage_server_connection_map** table gets updated when modifying the parameters for fibre or iSCSI based storage. The table contains the following columns:

- lun_id
- storage_server_connection

The **lun_id** column links to the **lun_id** column in the **luns** table as a foreign key. Special consideration must be made when updating the **lun_id** columns in either table.

7.3.2.4 luns

The **luns** table contains disk parameters such as serial number, vendor, and LVM UIDs for fibre and iSCSI based storage.

This table contains the following columns:

- phisical_volume_id
- lun_id
- volume_group_id
- serial
- lun_mapping
- vendor_id
- product_id
- device_size

All columns except the **phisical_volume_id**, **volume_group_id**, and d**evice_size** are subject to modification when fibre or iSCSI storage is attached.

The **phisical_volume_id** and **volume_group_id** map to information concerning LVM. Since mirrored LUNs are used, the LVM information should not change. The same holds true for the **device_size** column.

7.3.3 Domain Information

The UID for each storage domain must be known to make modifications to the correct entries in the tables. The **storage_domain_static** table contains this information. The following is a sample of the table and the data stored in it. The table stores the user friendly name of the storage domain in the **storage_name** column and a UID of the storage domain in the **storage** column.

```
# echo "SELECT storage_name,storage FROM storage_domain_static;" | psql -d
rhevm -U postgres

 storage_name |                storage
--------------+--------------------------------------
 DataDomain   | DIv0nZ-atbO-uLm7-Hq5q-5XRQ-eM4x-KaNFmR
 ISODomain    | cf810eeb-24df-4fdd-ae98-62416bd3e047
 ExportDomain | 6364acf1-754a-4525-9d97-3c6d5c8f9b6f
(3 rows)
```

7.3.4 Data Domain

Data storage domains can be backed by NFS, fibre, or iSCSI storage. Each of these storage types require different modifications to the database. These modifications can be an update to a single entry in a database table to multiple updates to multiple tables.

7.3.4.1 NFS Data Domain

An NFS based storage domain requires a change to the connection column in the **storage_server_connections** table. It is useful to query the **storage_name** and **storage** columns of the **storage_domain_static** table to display the mapping between the user friendly name of the storage domain and its UID. The UID is needed to locate the correct entry in the **storage_server_connections** table.

Query the **storage_domain_static** table to view its entries. The output shows that the UID for the data storage domain called **DataDomain** is ***5caa3ebc-6a61-48f4-9201-f261b9402e12*** . This UID is used to determine which entry in the **storage_server_connections** table need modification.

```
# echo "SELECT storage_name,storage FROM storage_domain_static;" | psql -d
rhevm -U postgres
 storage_name |               storage
--------------+--------------------------------------
 DataDomain   | 5caa3ebc-6a61-48f4-9201-f261b9402e12
 ExportDomain | b6a5ff4b-330d-4ac6-9874-8a408f903da0
 ISODomain    | 58611688-43e6-4366-b6af-7c04a7363a1d
(3 rows)
```

Connection information for NFS based storage domains is contained in the **connection** column of the **storage_server_connections** table. Using the UID from the **storage_domain_static** table for the export storage domain called **ExportDomain** as reference, query the **storage_server_connections** table for the NFS storage information. Only the **id** and **connection** columns need to be queried and ultimately updated.

```
# echo "SELECT id, connection FROM storage_server_connections;" | psql -d
rhevm -U postgres
                  id                  |                   connection
--------------------------------------
+------------------------------------------------
 5caa3ebc-6a61-48f4-9201-f261b9402e12 | rdr-
nfs.sitea.example:/exports/datadomain

 b6a5ff4b-330d-4ac6-9874-8a408f903da0 | rdr-
nfs.sitea.example:/exports/exports_configured

 58611688-43e6-4366-b6af-7c04a7363a1d | rdr-
nfs.sitea.example:/exports/isodomain
(3 rows)
```

Referring to **Table 3.1:NFS Exports**, the NFS export should be ***rdr-nfs.siteb.example:/exports/DATA***. Update the **storage_server_connections** table with the correct information and verify the table has the correct data in it.

```
# echo "UPDATE storage_server_connections SET connection='rdr-
nfs.siteb.example:/exports/DATA' where id='5caa3ebc-6a61-48f4-9201-
f261b9402e12';" | psql -d rhevm -U postgres
UPDATE 1

# echo "SELECT id, connection FROM storage_server_connections;" | psql -d
rhevm -U postgres
                  id                  |                   connection
--------------------------------------
+------------------------------------------------
 b6a5ff4b-330d-4ac6-9874-8a408f903da0 | rdr-
nfs.sitea.example:/exports/exports_configured

 58611688-43e6-4366-b6af-7c04a7363a1d | rdr-
nfs.sitea.example:/exports/isodomain

 5caa3ebc-6a61-48f4-9201-f261b9402e12 | rdr-nfs.siteb.example:/exports/DATA
```

The database now contains the correct information to allow it to activate the data storage domain.

7.3.4.2 Fibre Data Domain

Query the **storage_name** and **storage** columns to display a mapping between the user friendly name of the storage domain and its UID.

```
# echo "SELECT storage_name,storage FROM storage_domain_static;" | psql -d
rhevm -U postgres
 storage_name |                storage
--------------+--------------------------------------
 DataDomain   | qDlAWy-0YPw-Ie1T-8Yt5-vldq-h1q1-tmaJSa
 ISODomain    | 11121133-c07c-414d-8c54-4ce7d26c1495
 ExportDomain | da180f76-5b46-4bb3-b83f-8a9704b16e58
(3 rows)
```

Correcting the database for a fibre based storage domain requires an update to the **luns** table only. The **storage** column of the **storage_domain_static** table contains an entry that maps to the **volume_group_id column** of the **luns** table.

View the **luns** table. **Table 7.3.4.1: luns Table - Fibre** contains the information in an easier to read format.

```
# echo "SELECT * FROM luns;" | psql -d rhevm -U postgres
           phisical_volume_id           |              lun_id
|            volume_group_id             |                    serial
| lun_mapping | vendor_id | product_id | device_size
----------------------------------------+----------------------------------
+----------------------------------------
+------------------------------------------------+-------------+-----------
+------------+-------------
 yQAp5K-X4Ik-tvx9-vGS9-RNly-11JB-Jz4DNC | 3600c0ff000d7e69d69b89d4e01000000
| qDlAWy-0YPw-Ie1T-8Yt5-vldq-h1q1-tmaJSa |
SHP_MSA2324fc_00c0ffd7e69d000069b89d4e01000000 |          24 | HP        |
MSA2324fc  |         200
(1 row)
```

Column	Value
phisical_volume_id	yQAp5K-X4Ik-tvx9-vGS9-RNly-11JB-Jz4DNC
lun_id	3600c0ff000d7e69d69b89d4e01000000
volume_group_id	qDlAWy-0YPw-Ie1T-8Yt5-vldq-h1q1-tmaJSa
serial	SHP_MSA2324fc_00c0ffd7e69d000069b89d4e01000000
lun_mapping	24
vendor_id	HP
product_id	MSA2324fc
device_size	200

Table 7.3.4.1: luns Table - Fibre

The **phisical_volume_id** column contains the UID of the LVM physical volume on the hypervisor host. The **volume_group_id** column contains the UID of the LVM volume group on the hypervisor host. The **device _size** column contains the size of the fibre LUN. Since the environment uses mirrored luns for the data, neither of these entries change during a failover.

The information to modify the remaining columns is obtained from the new hypervisor host. Login to the hypervisor as the admin user and press the *F2* key to access a support shell. Execute the following **pvs** command to view the physical volume UID and name of the physical volumes on the system.

```
# pvs --noheadings --options pv_uuid,pv_name
  XGPO3I-y74E-mdc6-AbCe-bmxN-XdmG-gf1cD9
/dev/mapper/3600508b1001030374142384343301800p4

  yQAp5K-X4Ik-tvx9-vGS9-RNly-11JB-Jz4DNC
/dev/mapper/3600c0ff000d7e69d85b89d4e01000000
```

Looking for the value stored in the **phisical_volume_id** column of the **luns** table, the output shows that the physical volume name for the fibre LUN is ***/dev/mapper/3600c0ff000d7e69d85b89d4e01000000***.

This physical volume name is used for several command. To avoid errors when entering it, assign it to the variable ***PVNAME***.

```
# PVNAME=/dev/mapper/3600c0ff000d7e69d85b89d4e01000000
```

The values for the **vendor_id**, **product_id**, and **serial** columns of the database is retrieved by querying page *0x80* of the scsi device. This is done using the **scsi_id** command. All the options are required in order to return the output in the appropriate format.

```
# scsi_id --page=0x80 --whitelisted --export --replace-whitespace --device=$
{PVNAME}
ID_SCSI=1
ID_VENDOR=HP
ID_VENDOR_ENC=HP\x20\x20\x20\x20\x20\x20
ID_MODEL=MSA2324fc
ID_MODEL_ENC=MSA2324fc\x20\x20\x20\x20\x20\x20\x20
ID_REVISION=M112
ID_TYPE=disk
ID_SERIAL_RAW="SHP      MSA2324fc      00c0ffd7e69d000085b89d4e01000000"
ID_SERIAL=SHP_MSA2324fc_00c0ffd7e69d000085b89d4e01000000
ID_SERIAL_SHORT=00c0ffd7e69d000085b89d4e01000000
```

The lines for **ID_VENDOR**, **ID_MODEL**, and **ID_SERIAL** are used for the **vendor_id**, **product_id**, and **serial** columns respectively.

The value for the **lun_id** column is retrieved by querying page *0x83* of the scsi device. The line for **ID_SERIAL** contains the value to use in the **lun_id** column.

```
# scsi_id --page=0x83 --whitelisted --export --replace-whitespace --device=${PVNAME}
ID_SCSI=1
ID_VENDOR=HP
ID_VENDOR_ENC=HP\x20\x20\x20\x20\x20\x20
ID_MODEL=MSA2324fc
ID_MODEL_ENC=MSA2324fc\x20\x20\x20\x20\x20\x20\x20
ID_REVISION=M112
ID_TYPE=disk
ID_SERIAL_RAW="3600c0ff000d7e69d85b89d4e01000000"
ID_SERIAL=3600c0ff000d7e69d85b89d4e01000000
ID_SERIAL_SHORT=600c0ff000d7e69d85b89d4e01000000
ID_WWN=0x600c0ff000d7e69d
ID_WWN_VENDOR_EXTENSION=0x85b89d4e01000000
ID_WWN_WITH_EXTENSION=0x600c0ff000d7e69d85b89d4e01000000
ID_SCSI_SERIAL=00c0ffd7e69d000085b89d4e01000000
```

The **multipath** command provides the information for the **lun_mapping** column and also for the **device_size** column. The value for the **lun_mapping** column is the last number in the scsi identifier. The scsi identifier is the four numbers that are seperated by colons. The last number in the scsi identifier is the same and it is the value to place in the **lun_mapping** column of the database. The number 26 is the value to use in the database.

The value for the **device_size** column does not change when using mirrored luns, but the value can be verified if desired.

```
# multipath -ll ${PVNAME}
3600c0ff000d7e69d85b89d4e01000000 dm-10 HP,MSA2324fc
size=200G features='1 queue_if_no_path' hwhandler='0' wp=rw
`-+- policy='round-robin 0' prio=50 status=active
  |- 1:0:0:26 sdb 8:16 active ready running
  |- 2:0:0:26 sdd 8:48 active ready running
  |- 2:0:1:26 sde 8:64 active ready running
  `- 1:0:1:26 sdc 8:32 active ready running
```

The entry in the **luns** table can be updated since all the information needed is known. The following table contains the information gathered.

Column	Value
phisical_volume_id	yQAp5K-X4Ik-tvx9-vGS9-RNly-11JB-Jz4DNC
lun_id	3600c0ff000d7e69d85b89d4e01000000
volume_group_id	qDlAWy-0YPw-Ie1T-8Yt5-vldq-h1q1-tmaJSa
serial	SHP_MSA2324fc_00c0ffd7e69d000085b89d4e01000000
lun_mapping	26
vendor_id	HP
product_id	MSA2324fc
device_size	200

Table 7.3.4.2: luns Table - Fibre - Fixed

On the RHEV Manager server, update the database entry using the information gathered.

```
# echo "UPDATE luns SET lun_id='3600c0ff000d7e69d85b89d4e01000000',
serial='SHP_MSA2324fc_00c0ffd7e69d000085b89d4e01000000', lun_mapping='26',
vendor_id='HP', product_id='MSA2324fc' WHERE phisical_volume_id='yQAp5K-
X4Ik-tvx9-vGS9-RNly-11JB-Jz4DNC';" | psql -d rhevm -U postgres

UPDATE 1

# echo "SELECT * FROM luns;" | psql -d rhevm -U postgres
           phisical_volume_id           |              lun_id
|            volume_group_id             |                     serial
| lun_mapping | vendor_id | product_id | device_size
----------------------------------------+----------------------------------
+----------------------------------------
+------------------------------------------------+-------------+-----------
+------------+-------------
 yQAp5K-X4Ik-tvx9-vGS9-RNly-11JB-Jz4DNC | 3600c0ff000d7e69d85b89d4e01000000
| qDlAWy-0YPw-Ie1T-8Yt5-vldq-h1q1-tmaJSa |
SHP_MSA2324fc_00c0ffd7e69d000085b89d4e01000000 |          26 | HP        |
MSA2324fc  |         200
(1 row)
```

Updating the database to fix the fibre based data storage domain is complete.

7.3.4.3 iSCSI Data Domain

Query the **storage_name** and **storage** columns to display a mapping between the user friendly name of the storage domain and its UID.

```
# echo "SELECT storage_name,storage FROM storage_domain_static;" | psql -d
rhevm -U postgres

 storage_name |                storage
--------------+--------------------------------------
 DataDomain   | DIv0nZ-atbO-uLm7-Hq5q-5XRQ-eM4x-KaNFmR
 ISODomain    | cf810eeb-24df-4fdd-ae98-62416bd3e047
 ExportDomain | 6364acf1-754a-4525-9d97-3c6d5c8f9b6f
(3 rows)
```

Correcting the database for iSCSI storage domains requires changes to the **storage_server_connections**, **lun_storage_server_connection_map**, and **luns** tables.

Using the UID gathered from the **storage_domain_static** table, query the **luns** table for the entry where the **volume_group_id** column matches the UID. The output is difficult to read, its contents are placed in **Table 7.3.4.3: storage_domain_static Table - iSCSI** for readability.

```
# echo "SELECT * FROM luns WHERE volume_group_id='DIv0nZ-atbO-uLm7-Hq5q-
5XRQ-eM4x-KaNFmR';" | psql -d rhevm -U postgres

           phisical_volume_id           |    lun_id     |
volume_group_id             |      serial       | lun_mapping | vendor_id |
product_id  | device_size
---------------------------------------+---------------
+---------------------------------------+------------------+------------
+-----------+--------------+-------------
 VfXIWw-fU2H-utOS-uS1D-LypB-eh4j-RTdWnl | 1IET_00010001 | DIv0nZ-atbO-uLm7-
Hq5q-5XRQ-eM4x-KaNFmR | SIET_VIRTUAL-DISK |           1 | IET       |
VIRTUAL-DISK |         199
(1 row)
```

Column	Value
phisical_volume_id	VfXIWw-fU2H-utOS-uS1D-LypB-eh4j-RTdWnl
lun_id	1IET_00010001
volume_group_id	DIv0nZ-atbO-uLm7-Hq5q-5XRQ-eM4x-KaNFmR
serial	SIET_VIRTUAL-DISK
lun_mapping	1
vendor_id	IET
product_id	VIRTUAL-DISK
device_size	199

Table 7.3.4.3: storage_domain_static Table - iSCSI

The **lun_id**, **serial**, **lun_mapping**, **vendor_id**, and **product_id** columns can change when the Red Hat Enterprise Virtualization environment is brought up at the new site. The remaining columns should not change since the storage device is a mirrored LUN.

The values for the changing columns is derived from the scsi device parameters as reported by the **`scsi_id`** command. The **`scsi_id`** command is executed twice, once to query page *0x80* of the scsi device and once to query page *0x83*.

Before the **`scsi_id`** command can be executed to retrieve the needed information, the iSCSI target must be temporarily presented to a server.

The **`iscsiadm`** command is used to discover the iSCSI targets on the *siteb.example* network. The *rdr-iscsi.siteb.example* server presents storage as an iSCSI target on the *siteb.example* network. Discovering and mounting iSCSI targets requires the **`iscsi-initiator-utils`** package to be installed.

Discover the iSCSI targets presented by the *rdr-iscsi.siteb.example* server. The output is explained below and entered into **Table 7.3.4.4: iSCSI Target Information**.

```
# iscsiadm --mode discovery --type sendtargets --portal rdr-
iscsi.siteb.example

Starting iscsid:                                           [  OK  ]
192.168.201.32:3260,1 iqn.2011-11.siteb.example:rdr-iscsi.data
```

The information returned by the discovery mode of the **iscsiadm** command is used to modify the **storage_server_connections** table. The output of the discovery mode is in the following format:

IP_ADDRESS:PORT,TPGT TARGET_NAME

Where:

IP_ADDRESS is the IP address of the iSCSI targets server.

PORT is the port to use when connecting via iSCSI.

TPGT is the Target Portal Group Tag

TARGET_NAME is the iqn of the iSCSI target.

The output maps to the columns in the storage_server_connections table as follows:

connection:port,portal iqn

Table Column	Value
connection	192.168.201.32
iqn	iqn.2011-11.siteb.example:rdr-iscsi.data
port	3260
portal	1

Table 7.3.4.4: iSCSI Target Information

Login to the iSCSI target presented by *rdr-iscsi.siteb.example*.

```
# iscsiadm --mode node --targetname iqn.2011-11.siteb.example:rdr-iscsi.data
--portal rdr-iscsi.siteb.example --login

Logging in to [iface: default, target: iqn.2011-11.siteb.example:rdr-
iscsi.data, portal: 192.168.201.32,3260] (multiple)
Login to [iface: default, target: iqn.2011-11.siteb.example:rdr-iscsi.data,
portal: 192.168.201.32,3260] successful.
```

Checking the status of the iscsi service displays information about the attached devices. This information includes the LUN number used for the **lun_mapping** column in the **luns** table. The status also displays the linux scsi device name. The device name is needed to use the **scsi_id** command.

Get the status of the iscsi service. The last two lines of the output show the iSCSI target is attached as device sde using LUN number 8. The LUN number is the value needed for the **lun_mapping** column.

```
# service iscsi status
iSCSI Transport Class version 2.0-870
version 2.0-872.28.el6
Target: iqn.2011-11.siteb.example:rdr-iscsi.data
     Current Portal: 192.168.201.32:3260,1
     Persistent Portal: 192.168.201.32:3260,1
          **********
          Interface:
          **********
          Iface Name: default
          Iface Transport: tcp
          Iface Initiatorname: iqn.1994-05.com.redhat:363bfeb2a6c3
          Iface IPaddress: 192.168.201.40
          Iface HWaddress: <empty>
          Iface Netdev: <empty>

[ ... Output Truncated ...]

          Attached SCSI devices:
          ************************
          Host Number: 4   State: running
          scsi4 Channel 00 Id 0 Lun: 0
          scsi4 Channel 00 Id 0 Lun: 8
               Attached scsi disk sde      State: running
```

The linux device name and LUN id can also be determined by looking at the */dev/disk/by-path* directory and the */sys/block/*/device* directories.

```
# ls -l /dev/disk/by-path
total 0
lrwxrwxrwx. 1 root root  9 Jan  8 23:51 ip-192.168.201.32:3260-iscsi-
iqn.2011-11.siteb.example:rdr-iscsi.data-lun-8 -> ../../sde

# ls -l /sys/block/*/device
lrwxrwxrwx. 1 root root 0 Jan  8 12:18 /sys/block/sda/device
-> ../../../2:0:0:22/
lrwxrwxrwx. 1 root root 0 Jan  8 12:18 /sys/block/sdb/device
-> ../../../2:0:1:22/
lrwxrwxrwx. 1 root root 0 Jan  8 18:19 /sys/block/sdc/device
-> ../../../3:0:0:22/
lrwxrwxrwx. 1 root root 0 Jan  8 18:19 /sys/block/sdd/device
-> ../../../3:0:1:22/
lrwxrwxrwx. 1 root root 0 Jan  8 23:51 /sys/block/sde/device
-> ../../../4:0:0:8/
```

The **scsi_id** command can be executed against the device */dev/sde* to gather the remaining information needed to modify the database. The **--page** option specifies the device page to query. The **--device** option specifies the device to query.

By default the **scsi_id** command blacklists all devices and does not return any information when queried. The **--whitelisted** option changes this behavior and treats the device as whitelisted.

The **--export** option is used to print all data returned from the page using key/value pairs. This information may contain spaces. The **--replace-whitespace** option converts all whitespaces returned with underscores.

Query page *0x80* of scsi device */dev/sde*.

```
# scsi_id --page=0x80 --whitelisted --export --replace-whitespace
--device=/dev/sde

ID_SCSI=1
ID_VENDOR=RA-Group
ID_VENDOR_ENC=RA-Group
ID_MODEL=iscsi101
ID_MODEL_ENC=iscsi101
ID_REVISION=0001
ID_TYPE=disk
ID_SERIAL_RAW="SRA-Groupiscsi101"
ID_SERIAL=SRA-Groupiscsi101
ID_SERIAL_SHORT=
```

Page *0x80* provides the values for the **serial**,**vendor_id**, and **product_id** columns of the **luns** table. The value for the **serial** column is taken from the value specified by the **ID_SERIAL** key. The **vendor_id** and **product_id** columns take their values from the **ID_VENDOR** and **ID_MODEL** keys respectively.

Values for all columns but the **lun_id** column are known. The value for the **lun_id** column is the value specified by the **ID_SERIAL** key when querying page *0x83* of the scsi device.

Query page *0x83* to get the value for the **lun_id** column.

```
# scsi_id --page=0x83 --whitelisted --export --replace-whitespace
--device=/dev/sde
ID_SCSI=1
ID_VENDOR=RA-Group
ID_VENDOR_ENC=RA-Group
ID_MODEL=iscsi101
ID_MODEL_ENC=iscsi101
ID_REVISION=0001
ID_TYPE=disk
ID_SERIAL_RAW="1IET     00010008"
ID_SERIAL=1IET_00010008
ID_SERIAL_SHORT=IET_00010008
ID_SCSI_SERIAL=                          00ufo
```

The information gathered is presented in Table 7.3.4.5: luns Table Information - iSCSI.

luns Table Column	Value
lun_id	1IET_00010008
serial	SRA-Groupiscsi101
lun_mapping	8
vendor_id	RA-Group
product_id	iscsi101

Table 7.3.4.5: luns Table Information - iSCSI

All the information needed to modify the row for the iSCSI storage domain is known. The iSCSI target no longer needs to be connected. Log out of the iSCSI target.

```
# iscsiadm --mode node --targetname iqn.2011-11.siteb.example:rdr-iscsi.data
--portal rdr-iscsi.siteb.example --logout
Logging out of session [sid: 1, target: iqn.2011-11.siteb.example:rdr-
iscsi.data, portal: 192.168.201.32,3260]
Logout of [sid: 1, target: iqn.2011-11.siteb.example:rdr-iscsi.data, portal:
192.168.201.32,3260] successful.
```

The **luns** and **lun_storage_server_connection_map** tables can be updated. However, there is a foreign key policy configured between the **lun_id** columns of the **luns** table and the **lun_storage_server_connection_map** table. This prevents modifying the entry in the **luns** table.

This issue is resolved by inserting a new entry into the **luns** table, then modifying the **lun_storage_server_connection_map** table, and finally removing the original entry from the **luns** table.

Using the information from **Table 7.3.4.3: storage_domain_static Table - iSCSI** and **Table 7.3.4.5: luns Table Information - iSCSI**, insert a new entry into the **luns** table.

```
# echo "INSERT INTO luns VALUES ('VfXIWw-fU2H-utOS-uS1D-LypB-eh4j-RTdWnl',
'1IET_00010008', 'DIv0nZ-atbO-uLm7-Hq5q-5XRQ-eM4x-KaNFmR', 'SRA-
Groupiscsi101', '8', 'RA-Group', 'iscsi101', '199');" | psql -d rhevm -U
postgres

INSERT 262022 1

# echo "SELECT * FROM luns;" | psql -d rhevm -U postgres

           phisical_volume_id           |    lun_id     |
volume_group_id             |      serial       | lun_mapping | vendor_id |
product_id  | device_size
----------------------------------------+---------------
+----------------------------------------+------------------+------------
+-----------+-------------+-------------
 VfXIWw-fU2H-utOS-uS1D-LypB-eh4j-RTdWnl | 1IET_00010001 | DIv0nZ-atbO-uLm7-
Hq5q-5XRQ-eM4x-KaNFmR | SIET_VIRTUAL-DISK |           1 | IET       |
VIRTUAL-DISK |         199
 VfXIWw-fU2H-utOS-uS1D-LypB-eh4j-RTdWnl | 1IET_00010008 | DIv0nZ-atbO-uLm7-
Hq5q-5XRQ-eM4x-KaNFmR | SRA-Groupiscsi101 |           8 | RA-Group  |
iscsi101     |         199
(2 rows)
```

The new entry is now in the **luns** table. This allows the **lun_id** column of the **lun_storage_server_connection_map** table to be modified. View the **lun_storage_server_connection_map** table.

```
# echo "SELECT * FROM lun_storage_server_connection_map;" | psql -d rhevm -U
postgres

    lun_id     |      storage_server_connection
---------------+--------------------------------------
 1IET_00010001 | 56fd614d-e267-4756-b175-3035b6d4fa78
```

Modify the **lun_id** in the table so it contains the new **lun_id** value.

```
# echo "UPDATE lun_storage_server_connection_map SET lun_id='1IET_00010008'
WHERE lun_id='1IET_00010001';" | psql -d rhevm -U postgres
UPDATE 1

# echo "SELECT * FROM lun_storage_server_connection_map;" | psql -d rhevm -U
postgres
    lun_id     |      storage_server_connection
---------------+--------------------------------------
 1IET_00010008 | 56fd614d-e267-4756-b175-3035b6d4fa78
(1 row)
```

The **lun_storage_server_connection_map** table contains the correct **lun_id**, the original entry in the **luns** table can be removed. The UID in the **storage_server_connection** column is noted for later use.

Remove the original entry from the **luns** table.

```
# echo "DELETE FROM luns where volume_group_id='DIv0nZ-atbO-uLm7-Hq5q-5XRQ-
eM4x-KaNFmR' AND lun_id='1IET_00010001';" | psql -d rhevm -U postgres

DELETE 1

# echo "SELECT * FROM luns;" | psql -d rhevm -U postgres

           phisical_volume_id           |    lun_id     |
volume_group_id             |      serial       | lun_mapping | vendor_id |
product_id | device_size
----------------------------------------+---------------
+----------------------------------------+-------------------+-------------
+-----------+------------+-------------
 VfXIWw-fU2H-utOS-uS1D-LypB-eh4j-RTdWnl | 1IET_00010008 | DIv0nZ-atbO-uLm7-
Hq5q-5XRQ-eM4x-KaNFmR | SRA-Groupiscsi101 |           8 | RA-Group  |
iscsi101   |         199
(1 row)
```

The **storage_server_connections** table contains incorrect connection information for the iSCSI target. View the table to see the current configuration.

```
# echo "SELECT * FROM storage_server_connections;" | psql -d rhevm -U
postgres

                  id                  |              connection
| user_name | password |                      iqn                       |
port | portal | storage_type
--------------------------------------
+--------------------------------------+-----------+----------
+------------------------------------------------+------+--------
+--------------
 56fd614d-e267-4756-b175-3035b6d4fa78 | 192.168.200.32
|           |          | iqn.2011-11.sitea.example:rdr-iscsi.datadomain |
3260 | 1      |            3
 6364acf1-754a-4525-9d97-3c6d5c8f9b6f | rdr-
nfs.siteb.example:/exports/EXPORT |           |          |
|      |        |            1
 cf810eeb-24df-4fdd-ae98-62416bd3e047 | rdr-nfs.siteb.example:/exports/ISO
|           |          |                                                |
|        |           1
(3 rows)
```

The entry for the iSCSI storage domain must be modified. **Table 7.3.4.6: storage_server_connections Table Information - iSCSI** is completed using the UID from the **storage_server_connection** column of the **lun_storage_server_connection_map** table as noted **above**, to determine the entry in the **storage_server_connections** table that contains the values for the iSCSI storage domain.

Table Column	Value
id	56fd614d-e267-4756-b175-3035b6d4fa78
connection	192.168.200.32
user_name	
password	
iqn	iqn.2011-11.sitea.example:rdr-iscsi.datadomain
port	3260
portal	1
storage_type	3

Table 7.3.4.6: storage_server_connections Table Information - iSCSI

Only the **connection**, **iqn**, **port** and **portal** columns need modified since the new connection does not require authentication.

Modify the entry in the table using the information from Table 7.3.4.4: iSCSI Target Information.

```
# echo "UPDATE storage_server_connections SET
connection='192.168.201.32',iqn='iqn.2011-11.siteb.example:rdr-
iscsi.data',port='3260',portal='1' WHERE id='56fd614d-e267-4756-b175-
3035b6d4fa78';" | psql -d rhevm -U postgres

UPDATE 1

# echo "SELECT * FROM storage_server_connections;" | psql -d rhevm -U
postgres
                  id                  |              connection
| user_name | password |                   iqn                    | port |
portal | storage_type
--------------------------------------
+--------------------------------------+-----------+----------
+--------------------------------------+------+--------+-------------
 6364acf1-754a-4525-9d97-3c6d5c8f9b6f | rdr-
nfs.siteb.example:/exports/EXPORT |           |          |
|      |        |            1
 cf810eeb-24df-4fdd-ae98-62416bd3e047 | rdr-nfs.siteb.example:/exports/ISO
|           |          |                                          |      |
|            1
 56fd614d-e267-4756-b175-3035b6d4fa78 | 192.168.201.32
|           |          | iqn.2011-11.siteb.example:rdr-iscsi.data | 3260 | 1
|            3
(3 rows)
```

The database now contains the correct information to activate the iSCSI data storage domain.

7.3.5 Export Storage Domain

Export storage domains in the RHEV Environment use NFS based storage. The process to modify the database to fix export storage domains is the same process as fixing an NFS based data storage domain.

NFS based storage requires a change to the **connection** column in the **storage_server_connections** table. Query the **storage_name** and **storage** columns of the **storage_domain_static** table to display a mapping between the user friendly name of the storage domain and its UID.

```
# echo "SELECT storage_name,storage FROM storage_domain_static;" | psql -d
rhevm -U postgres
 storage_name |                storage
--------------+--------------------------------------
 DataDomain   | qDlAWy-0YPw-Ie1T-8Yt5-vldq-h1q1-tmaJSa
 ISODomain    | 11121133-c07c-414d-8c54-4ce7d26c1495
 ExportDomain | da180f76-5b46-4bb3-b83f-8a9704b16e58
(3 rows)
```

Connection information for NFS based storage domains is contained in the **connection** column of the **storage_server_connections** table. Using the UID from the **storage_domain_static** table for the export storage domain called **ExportDomain** as reference, query the **storage_server_connections** table for the NFS storage information. Only the **id** and **connection** columns need to be queried and updated since the export storage domain is NFS.

```
# echo "SELECT id, connection FROM storage_server_connections;" | psql -d
rhevm -U postgres
                  id                  |                   connection
--------------------------------------
+------------------------------------------------
 da180f76-5b46-4bb3-b83f-8a9704b16e58 | rdr-
nfs.sitea.example:/exports/exports_configured

 11121133-c07c-414d-8c54-4ce7d26c1495 | rdr-
nfs.sitea.example:/exports/isodomain
(2 rows)
```

Referring to **Table 3.1:NFS Exports**, the NFS export should be ***rdr-nfs.siteb.example:/exports/EXPORT***. Update the **storage_server_connections** table with the correct information and verify the table has the correct data in it.

```
# echo "UPDATE storage_server_connections SET connection='rdr-
nfs.siteb.example:/exports/EXPORT' where id='da180f76-5b46-4bb3-b83f-
8a9704b16e58';" | psql -d rhevm -U postgres
UPDATE 1

# echo "SELECT id, connection FROM storage_server_connections;" | psql -d
rhevm -U postgres
                  id                  |               connection
--------------------------------------
+--------------------------------------
 11121133-c07c-414d-8c54-4ce7d26c1495 | rdr-
nfs.sitea.example:/exports/isodomain

 da180f76-5b46-4bb3-b83f-8a9704b16e58 | rdr-
nfs.siteb.example:/exports/EXPORT
(2 rows)
```

The database now contains the correct information to allow it to activate the export storage domain.

7.3.6 ISO Storage Domain

Like the export storage domains, ISO Storage domains use NFS storage. Fixing the database for the ISO storage domain is done the same as an export storage domain.

Using the UID gathered in from the **storage_domain_static** table, query the **id** and **connection** columns of the **storage_server_connections** table.

```
# echo "SELECT id, connection FROM storage_server_connections WHERE
id='11121133-c07c-414d-8c54-4ce7d26c1495';" | psql -d rhevm -U postgres
                  id                  |               connection
--------------------------------------
+--------------------------------------
 11121133-c07c-414d-8c54-4ce7d26c1495 | rdr-
nfs.sitea.example:/exports/isodomain
(1 row)
```

Update the **storage_server_connections** table with the correct information.

```
# echo "UPDATE storage_server_connections SET connection='rdr-
nfs.siteb.example:/exports/ISO' where id='11121133-c07c-414d-8c54-
4ce7d26c1495';" | psql -d rhevm -U postgres
UPDATE 1

# echo "SELECT id, connection FROM storage_server_connections WHERE
id='11121133-c07c-414d-8c54-4ce7d26c1495';" | psql -d rhevm -U postgres
                  id                  |             connection
--------------------------------------+-----------------------------------
 11121133-c07c-414d-8c54-4ce7d26c1495 | rdr-nfs.siteb.example:/exports/ISO
(1 row)
```

The database now contains the correct information to allow it to activate the ISO storage domain.

7.3.7 RHEV Manager Service (jbossas)

Once all the modifications to the database are complete, it is safe to start the RHEV Manager service.

```
# service jbossas start
Starting jbossas:                                          [  OK  ]
```

The service must be started to connect to the web interface or the REST API.

7.4 Hypervisors

New hypervisors must be installed and added to the RHEV environment. Depending on the environment, hypervisors may be installed prior to failover to assist in limiting downtime during a failover. The servers running the hypervisors must have compatible architectures with the original hypervisors.

This paper does not discuss the installation of a hypervisor.[3] When configuring the hypervisor to join the RHEV environment, the original FQDN of rdr-*rhevm.sitea.example* must be used.

Enabling remote access to the hypervisor itself makes certain tasks easier and allows for a more complete automation solution. A public key is placed in the */root/.ssh/authorized_keys* file on the RHEV Hypervisor. This allows remote execution of commands into the hypervisor using ssh. The public key is placed in the */root/.ssh/authorized_keys* file by logging into the RHEV Hypervisor as the user ***admin*** and pressing the *F2* key. This brings up a support shell. The following commands, issued at the command prompt, place a copy the public key from */root/.ssh* on the RHEV Manager into the */root/.ssh/authorized_keys* file..

```
# cd /root/.ssh

# ssh rdr-rhevm.sitea.example "cat /root/.ssh/id_rsa.pub" >> authorized_keys
root@rdr-rhevm.sitea.example's password: [PASSWORD]
```

Once the hypervisor is configured and added to the RHEV environment, it is approved into the environment using the web portal or using the REST API.

The UID of the host and the cluster are needed in order to approve the hypervisor into the environment using the REST API. The REST API is used to gather the UIDs using the URI paths */api/hosts* and */api/clusters* respectively. The UIDs of the hosts are stored in an associative array named HH. The UIDs of the clusters are stored in an array named HC.

```
# curl --request "GET" ${APIBASE}/hosts  | xpath '/hosts/host/@id'
Found 2 nodes:
-- NODE --
 id="4f2b4996-12ba-11e1-90af-001871eba039"-- NODE --
 id="9f9f3b04-3a78-11e1-86d7-0025b3a9b001"

# curl --request "GET" ${APIBASE}/hosts | xpath '/hosts/host[@id="4f2b4996-
12ba-11e1-90af-001871eba039"]/name'
Found 1 nodes:
-- NODE --
<name>rdr-rhevh.sitea.example</name>

# curl --request "GET" ${APIBASE}/hosts | xpath '/hosts/host[@id="9f9f3b04-
3a78-11e1-86d7-0025b3a9b001"]/name'
Found 1 nodes:
-- NODE --
<name>rdr-rhevh.siteb.example</name>

# declare -A HH
```

3 http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Virtualization/3.0/html/Installation_Guide/index.html

```
# HH[rdr-rhevh.sitea.example]=4f2b4996-12ba-11e1-90af-001871eba039
# HH[rdr-rhevh.siteb.example]=9f9f3b04-3a78-11e1-86d7-0025b3a9b001


# curl --request "GET" ${APIBASE}/clusters | xpath '/clusters/cluster/@id'
Found 3 nodes:
-- NODE --
 id="99408929-82cf-4dc7-a532-9d998063fa95"-- NODE --
 id="ddbf36dc-12b9-11e1-ae84-001871eba039"-- NODE --
 id="9495276a-205a-11e1-9bb5-001871eba039"

# curl --request "GET" ${APIBASE}/clusters | xpath
'/clusters/cluster[@id="99408929-82cf-4dc7-a532-9d998063fa95"]/name'
Found 1 nodes:
-- NODE --
<name>Default</name>

# curl --request "GET" ${APIBASE}/clusters | xpath
'/clusters/cluster[@id="ddbf36dc-12b9-11e1-ae84-001871eba039"]/name'
Found 1 nodes:
-- NODE --
<name>SiteA</name>

# curl --request "GET" ${APIBASE}/clusters | xpath
'/clusters/cluster[@id="9495276a-205a-11e1-9bb5-001871eba039"]/name'
Found 1 nodes:
-- NODE --
<name>SiteB</name>

# declare -A HC
# HC[Default]=99408929-82cf-4dc7-a532-9d998063fa95
# HC[SiteA]=ddbf36dc-12b9-11e1-ae84-001871eba039
# HC[SiteB]=9495276a-205a-11e1-9bb5-001871eba039
```

To approve the hypervisor using the REST API, issue the following command:

```
# curl --request "POST" --data "<action><cluster id=\"$
{HC[SiteB]}\"/></action>" ${APIBASE}/hosts/${HH[rdr-
rhevh.siteb.example]}/approve
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<action>
    <cluster id="9495276a-205a-11e1-9bb5-001871eba039"/>
    <status>
        <state>complete</state>
    </status>
</action>
```

After the host is approved, check the status to ensure the host is in a usable state. A call to the REST API displays this information.

```
# curl --request "GET" ${APIBASE}/hosts/${HH[rdr-rhevh.siteb.example]} |
xpath '/host/status'
Found 1 nodes:
-- NODE --
<status>
        <state>non_operational</state>
        <detail>network_unreachable</detail>
    </status>
```

The status indicates there is a required network that is unreachable from this host. Viewing the event logs indicates the missing network is called **public**.

```
# curl --request "GET" ${APIBASE}/events

    <event id="504" href="/api/events/504">
        <description>Detected new Host rdr-rhevh.siteb.example. Host state
was set to Non Operational.</description>
        <code>13</code>
        <severity>normal</severity>
        <time>2011-12-13T14:18:28.438-06:00</time>
        <host id="805bb14c-2505-11e1-b809-0025b3a9b001"
href="/api/hosts/805bb14c-2505-11e1-b809-0025b3a9b001"/>
        <cluster id="9495276a-205a-11e1-9bb5-001871eba039"
href="/api/clusters/9495276a-205a-11e1-9bb5-001871eba039"/>
    </event>
    <event id="503" href="/api/events/503">
        <description>Host rdr-rhevh.siteb.example does not comply with the
cluster SiteB networks, the following networks are missing on host: 'public'
</description>
        <code>519</code>
        <severity>warning</severity>
        <time>2011-12-13T14:18:28.217-06:00</time>
        <host id="805bb14c-2505-11e1-b809-0025b3a9b001"
href="/api/hosts/805bb14c-2505-11e1-b809-0025b3a9b001"/>
        <cluster id="9495276a-205a-11e1-9bb5-001871eba039"
href="/api/clusters/9495276a-205a-11e1-9bb5-001871eba039"/>
    </event>
```

To attach the missing network called **public** to a network interface on the server, the UUID of a network interface on the RHEV Hypervisor and the UUID of the *PUBLIC* network are needed. The REST API provides the */api/networks* URI to retrieve the network names and UIDs.

Query the REST API for the networks. The output shows three networks, two named **rhevm** and one named **public**. Two networks with the same name exist since both the **Default** data center and the **RefArch** datacenter have networks named **rhevm** defined. Since only one network named **public** exists, this is the network the hypervisor needs.

```
# curl --request "GET" ${APIBASE}/networks | xpath '/networks/network/@id'
Found 3 nodes:
-- NODE --
 id="00000000-0000-0000-0000-000000000009"-- NODE --
 id="ee84a644-289c-494c-bad1-6a891fe5c8a7"-- NODE --
 id="bf2317b6-3f86-4583-b985-6c34afc81c16"

# curl --request "GET" ${APIBASE}/networks | xpath
'/networks/network[@id="00000000-0000-0000-0000-000000000009"]/name'
Found 1 nodes:
-- NODE --
<name>rhevm</name>

# curl --request "GET" ${APIBASE}/networks | xpath
'/networks/network[@id="ee84a644-289c-494c-bad1-6a891fe5c8a7"]/name'
Found 1 nodes:
-- NODE --
<name>rhevm</name>

# curl --request "GET" ${APIBASE}/networks | xpath
'/networks/network[@id="bf2317b6-3f86-4583-b985-6c34afc81c16"]/name'
Found 1 nodes:
-- NODE --
<name>public</name>
```

If multiple networks called **public** existed, the */API/NETWORKS* URI could provide the datacenter UID for the network as well. The output shows two unique data center UIDs, showing one datacenter has multiple networks defined.

```
# curl --request "GET" ${APIBASE}/networks | xpath
'/networks/network[@id="00000000-0000-0000-0000-
000000000009"]/data_center/@id'
Found 1 nodes:
-- NODE --
 id="632b8642-12b8-11e1-9936-001871eba039"

# curl --request "GET" ${APIBASE}/networks | xpath
'/networks/network[@id="ee84a644-289c-494c-bad1-
6a891fe5c8a7"]/data_center/@id'
Found 1 nodes:
-- NODE --
 id="10b3c00e-9371-48ce-8d28-28158cf26d96"

# curl --request "GET" ${APIBASE}/networks | xpath
'/networks/network[@id="bf2317b6-3f86-4583-b985-
6c34afc81c16"]/data_center/@id'
Found 1 nodes:
-- NODE --
 id="10b3c00e-9371-48ce-8d28-28158cf26d96"
```

The */api/datacenters* URI provides the datacenter UID and name. This URI is used to gather the user friendly name of the datacenter using the UIDs of the datacenter. Since the datacenter UIDs are used in various calls to the REST API, add them to an associative array.

```
# curl --request "GET" ${APIBASE}/datacenters | xpath
'/data_centers/data_center[@id="632b8642-12b8-11e1-9936-001871eba039"]/name'
Found 1 nodes:
-- NODE --
<name>Default</name>

# curl --request "GET" ${APIBASE}/datacenters | xpath
'/data_centers/data_center[@id="10b3c00e-9371-48ce-8d28-28158cf26d96"]/name'
Found 1 nodes:
-- NODE --
<name>RefArch</name>

# declare -A DC
# DC[Default]=632b8642-12b8-11e1-9936-001871eba039
# DC[RefArch]=10b3c00e-9371-48ce-8d28-28158cf26d96
```

The last piece of information needed to attach the **public** network to the host is the UID of a nic on the host. This environment uses the interface with the MAC address of 00:17:A4:77:24:3A for the **public** network. The following gathers the needed information.

```
# curl --request "GET" ${APIBASE}/hosts/${HH[rdr-rhevh.siteb.example]}/nics
| xpath '/host_nics/host_nic/@id'
Found 8 nodes:
-- NODE --
 id="f9a68349-6489-4944-9f42-078c9d0c5ec8"-- NODE --
 id="aff0cf2d-b0c0-48c1-a294-8360839f9518"-- NODE --

# curl --request "GET" ${APIBASE}/hosts/${HH[rdr-rhevh.siteb.example]}/nics
| xpath '/host_nics/host_nic[@id="f9a68349-6489-4944-9f42-
078c9d0c5ec8"]/name'
Found 1 nodes:
-- NODE --
<name>eth0</name>

# curl --request "GET" ${APIBASE}/hosts/${HH[rdr-rhevh.siteb.example]}/nics
| xpath '/host_nics/host_nic[@id="f9a68349-6489-4944-9f42-
078c9d0c5ec8"]/mac/@address'
Found 1 nodes:
-- NODE --
 address="00:17:A4:77:24:38"

# curl --request "GET" ${APIBASE}/hosts/${HH[rdr-rhevh.siteb.example]}/nics
| xpath '/host_nics/host_nic[@id="aff0cf2d-b0c0-48c1-a294-
8360839f9518"]/name'
Found 1 nodes:
-- NODE --
<name>eth1</name>

# curl --request "GET" ${APIBASE}/hosts/${HH[rdr-rhevh.siteb.example]}/nics
| xpath '/host_nics/host_nic[@id="aff0cf2d-b0c0-48c1-a294-
8360839f9518"]/mac/@address'
Found 1 nodes:
-- NODE --
 address="00:17:A4:77:24:3A"
```

All the information needed to attach the **public** network to the host is now known. The REST API provides a means to attach the network to a host. The URI to attach the host requires the UIDs of the host and nic. The request also requires the UID of the network to be passed in an XML format. The XML code is passed using the **`---data`** option to the **`curl`** command.

The URI to attach the network is */api/hosts/**`HOST_UID`**/nics/**`NIC_UID`**/attach*.

The XML code used is:

```
<action>
  <network id="NET_UID"/>
</action>
```

The following values are used in the above URI and XML code.

HOST_UID	${HH[rdr-rhevh.siteb.example]}
NIC_UID	aff0cf2d-b0c0-48c1-a294-8360839f9518
NET_UID	bf2317b6-3f86-4583-b985-6c34afc81c16

Attach the network to the host using the following command.

```
# curl --request "POST" --data "<action><network id=\"bf2317b6-3f86-4583-
b985-6c34afc81c16\"/></action>" ${APIBASE}/hosts/${HH[rdr-
rhevh.siteb.example]}/nics/aff0cf2d-b0c0-48c1-a294-8360839f9518/attach
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<action>
    <network id="bf2317b6-3f86-4583-b985-6c34afc81c16"/>
    <status>
        <state>complete</state>
    </status>
</action>
```

After the network is attached to the interface on the host, the network configuration must be saved. Failing to save the network configuration makes the network unavailable if the host is rebooted. The following command and URI save the network configuration of the host.

```
# curl --request "POST" --data "<action/>" ${APIBASE}/hosts/${HH[rdr-
rhevh.siteb.example]}/commitnetconfig
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<action>
    <status>
        <state>complete</state>
    </status>
</action>
```

The host is activated and used in the environment. The following activates the host.

```
# curl --request "POST" --data "<action/>" ${APIBASE}/hosts/${HH[rdr-
rhevh.siteb.example]}/activate
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<action>
    <status>
        <state>complete</state>
    </status>
</action>
```

The hypervisors at the original site, SiteA, show to be the *Storage Pool Manager (SPM)*. These hypervisors need to be fenced to allow a hypervisor at SiteB to become the SPM. The Data Storage Domain must be fixed and accessible before the fencing process will work.

Fence the hypervisor from SiteA.

```
# curl --request "POST" --data
"<action><fence_type>manual</fence_type></action>" ${APIBASE}/hosts/$
{HH[rdr-rhevh.sitea.example]}/fence

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<action>
    <fence_type>manual</fence_type>
    <status>
        <state>complete</state>
    </status>
</action>
```

The **Spm Status** of the SiteA hypervisor changes to **None**. After a few seconds, the **Spm Status** of the SiteB hypervisor changes to **Contend**. After a few minutes, the status should change to **SPM**. The SiteB hypervisor is now the Storage Pool Manager. All the storage domains become active.

7.4.1 Moving the VMs

The virtual machines are assigned to the SiteA cluster. They cannot be started until they are assigned to the SiteB cluster. The REST API is used to assign the VM to the SiteB cluster.

The UID of the virtual machine is needed for the REST API request. The */api/vms* URI retrieves the information about the virtual machines.

```
# curl --request "GET" ${APIBASE}/vms | xpath "/vms/vm/@id | /vms/vm/name"
Found 8 nodes:
-- NODE --
 id="bdbd6360-7c5b-4f97-9534-1afab57de724"-- NODE --
<name>rdr-devel-vm</name>-- NODE --

 id="0d1d84e7-be6f-49fb-9294-a232a8cbd044"-- NODE --
<name>rdr-w2k8</name>-- NODE --

 id="808f87e6-3a2f-46ba-a35b-4089cf18360d"-- NODE --
<name>rdr-W7</name>-- NODE --

 id="2dc70b1c-7c7f-49d6-b3dd-ba891e5283f8"-- NODE --
<name>rdr-web-vm</name>
```

A *PUT* request is made to the */api/vms* URI to modify the properties of the virtual machine. The name of the new cluster is submitted as XML code. The following loop changes the virtual machine to be in the SiteB cluster.

```
# for vmid in bdbd6360-7c5b-4f97-9534-1afab57de724 0d1d84e7-be6f-49fb-9294-
a232a8cbd044 808f87e6-3a2f-46ba-a35b-4089cf18360d 2dc70b1c-7c7f-49d6-b3dd-
ba891e5283f8
> do
> curl --request "PUT" --data
"<vm><cluster><name>SiteB</name></cluster></vm>" ${APIBASE}/vms/${vmid}
> done

[ ...OUTPUT TRUNCATED... ]
```

7.4.2 Starting the Virtual Machines

The virtual machines need to be powered on if they do not come up automatically. This is done using the web portal to prevent any resource race conditions.

Virtual machines can be powered on using the REST API as well. The following loop starts the virtual machines with a delay of 5 seconds between each start request.

```
# for vmid in bdbd6360-7c5b-4f97-9534-1afab57de724 0d1d84e7-be6f-49fb-9294-
a232a8cbd044 808f87e6-3a2f-46ba-a35b-4089cf18360d 2dc70b1c-7c7f-49d6-b3dd-
ba891e5283f8
> do
> curl --request "POST" --data "<action/>" ${APIBASE}/vms/${vmid}/start
> sleep 5
> done
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<action>
    <status>
        <state>complete</state>
    </status>
</action>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<action>
    <status>
        <state>complete</state>
    </status>
</action>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<action>
    <status>
        <state>complete</state>
    </status>
</action>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<action>
    <status>
        <state>complete</state>
    </status>
</action>
```

8 Automating the Failover

Automating the failover process can be easily done using various scripting languages such as Python, Perl, bash, etc..

This section demonstrates using scripts to automate the failover of the environments used in this reference architecture.

This scripts are not designed to work for every environment and is not meant for a production environment since it does not perform extensive error checking. The intent of the scripts are to demonstrate the automation of a site level failover of a Red Hat Enterprise Virtualization environment is possible with a little planning. The scripts are written in bash as it is more widely known than other languages such as python or Perl.

The scripts use a configuration file to store their settings. Each line of the configuration file begins with the type of entry followed by a keyword and up to three values. If a line begins with a pound (#) or semicolon (;) then it is considered a comment line and is ignored. The types and keys that may exist in the configuration file are listed in **Table 8.1: Configuration File.**

The following example shows a type of **CONFIG** followed by a keyword and then a value. The keywords in this example are logfile and working_dir.

```
CONFIG            logfile          rdr.log
CONFIG            working_dir      /tmp/rdr
```

The information from the configuration file is stored in associative arrays that have identical names to the **Type** column in **Table 8.1: Configuration File** with a key from the **Key** column. Referencing the **Type** and **keys** are done using the syntax of **Type[key]**. This is identical to how the associative arrays are referenced in the script. For instance, when the key **logfile** of type **CONFIG** is referenced, it will be presented in the document as **CONFIG[logfile]**.

Type	Key		Value
API	host	Hostname of the RHEV Manager	
	port	Secure port to connect to when making REST API requests.	
	base	The base URI for the API. Usually ***api***.	
	user	User name to use when accessing the REST API. Format ***admin@internal***	
	pass	Password for the API user.	
	cert	Name of the cert on the RHEV Manager as accessed via URL. Usually ***ca.crt.***	
	cert_port	Port to connect to on the RHEV Manager when retrieving the certificate. Usually ***8080***.	
DB	user	PostgreSQL user name to use when accessing the database.	
	pass	Password for PostgreSQL user if needed.	
CONFIG	logfile	File to log output to.	
	working_dir	Directory to place log files and such.	
Storage_Domain	**[Data Domain Name]**	The connection information for the Storage Domains after failover.	
			The first value after the key identifies the type of storage as either ***nfs***, ***fibre***, or ***iscsi***.
			If the storage type is ***nfs***, the second value specifies the NFS export to use.
			If the storage type is ***fibre***, the second value after the key identifies the host name of a hypervisor that can be used to gather information about the fibre LUN. The third value after the key identifies the scsi serial number as presented by the storage. The host needs remote ssh access into the root account.
			If the storage type is ***iscsi***, the second value after the key identifies the host name or IP address of the iscsi target. The third value specifies the ***iqn*** of the iSCSI target.

Type	Key		Value
SITE_INFO	datacenter	The name of the datacenter performing a failover.	
	main_site_cluster	Cluster name for the main site.	
	bkup_site_cluster	Cluster name for the site to fail over to.	
VIRT_NET	**[hypervisor name]**	The virtual networks that need to be configured on the hypervisors.	
			The first value after the key is the network interfaces hardware address to map to a virtual network. The second value after the key is the name of the virtual network to map.
RHEVM_NIC	**[NIC Hardware Address]**	The network hardware address to interface name mappings to use on the RHEV Manager.	
			The first value after the key is the interface name to map to.
START_VM	**[VM name]**	Delay in seconds to wait before starting this virtual machine.	

Table 8.1: Configuration File

8.1 Functions

Most of the work in the scripts are done using functions. Functions make fixing and changing aspects of the script easier. They also break the script into pieces that allow for a better understanding of their purpose.

Some of the functions allow the script to look nicer when executed, but most functions perform the tasks associated with the failover. An explanation of how the function works is provided. The following functions are used in the script.

8.1.1 dust_settle()

Certain aspects of the failover process can take time to compete. This is not a concern when manually performing a failover since the processes are usually finished before the next process or command needs executed.

Because the system executes processes and commands much faster than can be done by hand, automating the process can cause some race conditions. Adding sleep statements to the script can fix this problem. However, when the script is sleeping, there is no indication if the script is sleeping or has entered a non-responsive state.

The **dust_settle()** function displays a countdown, presented in seconds, indicating the script is still active and in a sleep state. The function accepts one argument indicating the number of seconds to sleep.

```
dust_settle() {
  count=$1

  for (( count=$1; count>0; count-- ))
  do
    output "                \r\c" screen
    output "   --- ${count} \r\c" screen
    sleep 1
  done
  output "               " screen
}
```

8.1.2 output()

Although printing to Standard Out (**STDOUT)** and to a logfile is easily done using the **echo** command and redirection, this can make the script look cluttered and can make changes to the output tedious.

The **output()** function allows cleaner output and versatility as to when and where to display the output.

The function accepts one or two arguments. The first argument specifies the text to print. The second argument indicates where to print the text. If the second argument is the word **screen**, then the text is printed to **STDOUT**. If the second argument is the word **log** and a file to log output to has been specified, then the text is printed to the file. Finally, if the second argument is not specified then the output is printed to **STDOUT** and to the file if it has been specified.

The name of the file for writing the output to is specified using the **CONFIG[logfile]** and **CONFIG[working_dir]** entry's in the configuration file. These entries specify the name of the logfile and the directory to place the file, respectively.

The code for the **output()** function is:

```
output() {
  if [ "$2" != "log" ] && [ ! ${CONFIG[quiet]} ]
  then
    echo -e "$1"
  fi

  if [ "$2" != "screen" ] && [ ${CONFIG[logfile]} ]
  then
    echo -e "$1" >> ${CONFIG[working_dir]}/${CONFIG[logfile]}
  fi
}
```

8.1.3 check_needed_commands()

The **xpath** command is needed to parse the XML output returned from the REST API calls. This function checks to see if the **xpath** command is installed into the users path. This function also checks if the **iscsiadm** command is installed. If a command is not installed in the users path, the script prints a suggested package that might contain the script and exits. This function does not take any arguments to function.

An associative array called **NEEDED_CMDS** is defined. This array contains the name of the command and the package that contains the command.

Declare the **NEEDED_CMDS** associative array and add entries for the **xpath** and **iscsiadm** commands. This information is not stored in the configuration file since it required by the script.

```
declare -A NEEDED_CMDS

NEEDED_CMDS[iscsiadm]=iscsi-initiator-utils
NEEDED_CMDS[xpath]=perl-XML-XPath
```

The **check_needed_commands()** function.

```
check_needed_commands() {
  missing=0
  for command in ${!NEEDED_CMDS[@]}
  do
    which ${command} > /dev/null 2>&1
    if [ $? -ne 0 ]
    then
      output "${command}: not found and is needed."
      output "     This command may be provided by the ${NEEDED_CMDS[$
{command}]} package."
      missing=1
    fi
  done

  if [ ${missing} -eq 1 ]
  then
    output "Exiting..."
    exit
  fi
}
```

8.1.4 REST_get_cert()

The **REST_get_cert()** function retrieves the certificate from the RHEV Manager. This function does not take any arguments. The configuration file stores the values needed for this function to operate. The function uses the **CONFIG[working_dir]** from the configuration file as the location to store the certificate.

The function also uses the **API[cert]**, **API[host]**, and the **API[cert_port]** definitions. These specify the name of the certificate on the REST API server, the servers host name,and port respectively. This information is stored an associative array called **API**.

The function takes no arguments when called.

```
REST_get_cert() {
  output "Retrieving certificate..... \c"
  curl --silent -o ${CONFIG[working_dir]}/${API[cert]} http://${API[host]}:$
{API[cert_port]}/${API[cert]}
  output "Done"
}
```

8.1.5 REST_submit()

The **REST_submit()** is used to submit requests to the REST API instead of issuing lengthy **`curl`** commands throughout the script. This function accepts up to three arguments when called. The third argument is optional, but the first two are mandatory.

The first argument specifies the request type to make to the REST API. Common values for this argument are **GET**, **POST**, and **PUT**.

The second argument specifies the branch of the API to submit to. To query the virtual machines, this argument would be **vms**.

The optional third argument is the XML code to send to the REST API when needed. The third argument is used with the **POST** and **PUT** requests.

The function uses the **CONFIG[working_dir]**, **API[cert]**, **API[host]**, **API[user]**, **API[pass]**, **API[port]**, and **API[base]** values from the configuration file to complete the requests.

The function builds the request to the REST API.

```
REST_submit() {
  # request_type, api_branch, data
  curl --silent                                                \
       --cacert  ${CONFIG[working_dir]}/${API[cert]}           \
       --header  "Content-Type: application/xml"               \
       --user    "${API[user]}:${API[pass]}"                   \
       --request "$1"                                          \
       --data    "$3"                                          \
       https://${API[host]}:${API[port]}/${API[base]}/$2
}
```

8.1.6 get_scsi_domain()

The **get_scsi_domain()** function takes a fibre storage domain as its only argument. The function then connects to the server specified in the **FIBRE[STORAGE DOMAIN]** array. Using the wwid stored in the array and the **`scsi_id`** command, the function gathers the information needed to modify the database. The information is then stored into the **FIBRE[STORAGE DOMAIN]** element of the array.

```
get_scsi_domain() {
  sname=$1
  fibre_info=( $( echo ${FIBRE[${sname}]} | tr '|' ' ' ) )

  dev_wwn=( $( ssh ${fibre_info[0]} 'for disk in $( ls -d /sys/block/sd* |
sed "s/^.*\///" )
  do
    echo -e "${disk} \c"
    scsi_id --whitelisted --page 0x83 --export /dev/${disk} | grep
ID_SCSI_SERIAL | sed "s/^.*=//"
  done' | grep ${fibre_info[1]} ) )

  out1=( $( ssh ${fibre_info[0]} "scsi_id --page=0x80 --whitelisted --export
```

```
--replace-whitespace --device=/dev/${dev_wwn}" | awk -F '=' '/^ID_VENDOR=|
^ID_MODEL=|^ID_SERIAL=/ { printf "|" $2 }' ) )

  out2=( $( ssh ${fibre_info[0]} "scsi_id --page=0x83 --whitelisted --export
--replace-whitespace --device=/dev/${dev_wwn}" | awk -F '=' '/^ID_SERIAL=/ {
printf "|" $2 }' ) )

  out3=( $( ssh ${fibre_info[0]} "ls -l /sys/block/${dev_wwn}/device " | awk
-F ':' '{ sub(/\/$/, ""); print $NF }' ) )

  FIBRE[${sname}]="${FIBRE[${sname}]}${out1}${out2}|${out3}"
}
```

8.1.7 fix_scsi_domain()

The **fix_scsi_domain()** function uses the information populated in the **FIBRE[STORAGE DOMAIN]** array to fix the RHEV Manager database. The **get_scsi_domain()** must be called before this function.

```
fix_scsi_domain() {
  sname=$1
  id=$( echo "SELECT storage FROM storage_domain_static WHERE
storage_name='${sname}';" | psql -d rhevm -U postgres --tuples-only | sed
's/^\s*//' )

  iscsi_tmp=( $( echo ${FIBRE[${sname}]} | tr '|' ' ' ) )

  echo "UPDATE luns SET lun_id='${iscsi_tmp[5]}', serial='${iscsi_tmp[4]}',
lun_mapping='${iscsi_tmp[6]}', vendor_id='${iscsi_tmp[2]}', product_id='$
{iscsi_tmp[3]}' WHERE volume_group_id='${id}';" | psql -d rhevm -U postgres
--tuples-only
}
```

8.1.8 get_iscsi_info()

The **get_iscsi_info()** function takes an iSCSI Storage Domain as its only option. The function uses the value stored in **ISCSI[[STORAGE DOMAIN]]** to log into the iSCSI target and retrieve the necessary information from the **`scsi_id`** command. The information is then appended to the original value of **ISCSI[[STORAGE DOMAIN]]**.

```
get_iscsi_info() {
  sname=$1
  iscsi_info=( $( echo ${ISCSI[${sname}]} | tr '|' ' ' ) )

  iscsiadm --mode discovery --type sendtargets --portal ${iscsi_info[0]}

  iscsiadm --mode node --targetname ${iscsi_info[1]} --portal $
{iscsi_info[0]} --login

  sleep 5
  iscsi_device=$( ls -l /dev/disk/by-path | sed -n "/${iscsi_info[1]}/p" |
```

```
sed 's/.*\///' )

  tmp1=$( scsi_id --page=0x80 --whitelisted --export --replace-whitespace
--device=/dev/${iscsi_device} | awk -F '=' '/^ID_VENDOR=|^ID_MODEL=|
^ID_SERIAL=/ { printf "|" $2 }' )

  tmp2=$( scsi_id --page=0x83 --whitelisted --export --replace-whitespace
--device=/dev/${iscsi_device} | awk -F '=' '/^ID_SERIAL=/ { printf "|" $2 }'
)

  tmp3=$( ls -l /sys/block/${iscsi_device}/device | awk -F ':' '{ sub(/\/$/,
""); print $NF }' )

  ISCSI[${sname}]="${ISCSI[${sname}]}${tmp1}${tmp2}|${tmp3}"

  iscsiadm --mode node --targetname ${iscsi_info[1]} --portal $
{iscsi_info[0]} --logout

}
```

8.1.9 fix_iscsi_domain()

The **fix_iscsi_info()** function takes an iSCSI Storage Domain as its only option. It uses the value from **ISCSI[[STORAGE DOMAIN]]** to modify the database tables to fix the storage information.

```
fix_iscsi_domain() {
  sname=$1
  id=$( echo "SELECT storage FROM storage_domain_static WHERE
storage_name='${sname}';" | psql -d rhevm -U postgres --tuples-only | sed
's/^\s*//' )

  orig_lid=$( echo "SELECT lun_id FROM luns WHERE volume_group_id='${id}';"
| psql -d rhevm -U postgres --tuples-only | sed 's/^\s*//' )

  tmp=$( echo "SELECT phisical_volume_id,device_size FROM luns WHERE
volume_group_id='${id}';" | psql -d rhevm -U postgres --tuples-only | sed
's/^\s*//' | tr --delete ' ' )
  ISCSI[${sname}]="${ISCSI[${sname}]}|${tmp}"

  ssc_id=$( echo "SELECT storage_server_connection FROM
lun_storage_server_connection_map WHERE lun_id='${orig_lid}';" | psql -d
rhevm -U postgres --tuples-only | sed 's/^\s*//' )

  iscsi_tmp=( $( echo ${ISCSI[${sname}]} | tr '|' ' ' ) )

  echo "INSERT INTO luns VALUES ('${iscsi_tmp[7]}', '${iscsi_tmp[5]}', '$
{id}', '${iscsi_tmp[4]}', '${iscsi_tmp[6]}', '${iscsi_tmp[2]}', '$
{iscsi_tmp[3]}', '${iscsi_tmp[8]}');" | psql -d rhevm -U postgres --tuples-
only

  echo "UPDATE lun_storage_server_connection_map SET lun_id='$
{iscsi_tmp[5]}' WHERE lun_id='${orig_lid}';" | psql -d rhevm -U postgres
--tuples-only
```

```
  echo "DELETE FROM luns WHERE volume_group_id='${id}' AND lun_id='$
{orig_lid}';" | psql -d rhevm -U postgres --tuples-only
  echo "UPDATE storage_server_connections SET connection='${iscsi_tmp[0]}',
iqn='${iscsi_tmp[1]}' WHERE id='${ssc_id}';" | psql -d rhevm -U postgres
--tuples-only

}
```

8.1.10 REST_get_dc()

The **REST_get_dc()** function queries the REST API for datacenter information. It parses the returned XML for the datacenter name and UID. The names and UIDs are stored in an associative array called **DC**. The name is used for the key of the array and the UID is the value.

REST_get_DC()

```
REST_get_dc() {
  output "Getting Datacenter UIDs..."

  while read uid name
  do
    DC[${name}]=${uid}
  done < <(
    REST_submit GET datacenters \
    | xpath '/data_centers/data_center/@id | /data_centers/data_center/name'
\
    | sed -e 's/id=\|"//g' -e 's/<name>/ /g' -e 's/<\/name>/\n/g'
    )
}
```

8.1.11 REST_get_hc()

The **REST_get_hc()** function queries the REST API for cluster information. It parses the returned XML for the cluster name and UID. The names and UIDs are then stored in an associative array called **HC**. The name is used for the key of the array and the UID is the value.

```
REST_get_hc() {
  output "Getting Cluster UIDs..."

  while read uid name
  do
    HC[${name}]=${uid}
  done < <(
    REST_submit GET clusters \
    | xpath '/clusters/cluster/@id | /clusters/cluster/name' \
    | sed -e 's/id=\|"//g' -e 's/<name>/ /g' -e 's/<\/name>/\n/g'
    )
}
```

8.1.12 REST_get_hh

The **REST_get_hh()** function queries the REST API for hypervisor host information. It parses the returned XML for the host name and UID. The names and UIDs are then stored in an associative array called **HH**. The name is used for the key of the array and the UID is the value.

```
REST_get_hh() {
  output "Getting Host UIDs..."

  while read uid name
  do
    HH[${name}]=${uid}
  done < <(
    REST_submit GET hosts \
    | xpath '/hosts/host/@id | /hosts/host/name' \
    | sed -e 's/id=\|"//g' -e 's/<name>/ /g' -e 's/<\/name>/\n/g'
    )
}
```

8.1.13 get_hosts_status()

The **get_hosts_status()** function checks the state of all the hosts and places the names into one of three arrays indicating what type of action is needed for the host. The function calls the **REST_check_host_status()** function to return the status of a single host.

If the status of the host is returned as **pending_approval**, then the host name is placed into the array called **host_approve**. This array is later parsed and an attempt is made to approve the

hosts stored in the array.

If the status of the host is returned as **non_responsive**, then the host name is placed into the array called **host_fence**. This array is later parsed and an attempt is made to fence the hosts stored in the array.

If the status of the host is returned as **non_operational**, then the host name is placed into the array called **host_activate**. This array is later parsed and an attempt is made to activate the hosts stored in the array.

```
get_hosts_status() {

  output "Getting status of Hypervisor Hosts"

  unset -v host_approve host_fence host_activate

  for host in ${!HH[@]}
  do
    case $( REST_check_host_status ${host} 2>/dev/null ) in
      pending_approval)
        host_approve[${#host_approve[@]}]=${host}
        ;;
      non_responsive)
        host_fence[${#host_fence[@]}]=${host}
        ;;
      non_operational)
        host_activate[${#host_activate[@]}]=${host}
        ;;
      *)
        ;;
    esac
  done
}
```

8.1.14 REST_check_host_status()

The **REST_check_host_status()** function submits a query to the REST API requesting information about a host. The function takes the name of the host to check as its only argument and returns the state of the host as reported by the REST API.

```
REST_check_host_status() {
  REST_submit GET hosts/${HH[$1]}       \
    | xpath "/host/status/state"        \
    | sed -e 's/<state>\|<\/state>//g'
}
```

8.1.15 REST_approve_host()

The **REST_approve_host()** function accepts two arguments. The first argument specifies the host name to approve and the second argument specifies the cluster to place the host into.

The events on the RHEV Manager are queried using the REST API and are checked after the host has been approved. An attempt to fix the virtual network configuration on the host is made if an event code of 519 is found. The networking is fixed using the **REST_fix_host_network()** function.

```
REST_approve_host() {
  host=$1
  cluster=$2

  tmp_approve=$(
    REST_submit GET events          \
    | xpath "/events/event[1]/@id"  \
    | sed -e 's/.*id="\|".*$//g'
    )

  tmp_out=$( REST_submit POST hosts/${HH[${host}]}/approve "<action><cluster
id=\"${HC[${cluster}]}\"/></action>" 2>&1 )
  output "${tmp_out}" log

  output "Giving hosts time to finish approval process."
  dust_settle 5

  tapp_arr=( $(
    REST_submit GET "events?search=type%3D519&from=${tmp_approve}" \
    | xpath "/events/event/description"                            \
    | awk '{ print $2 $(NF-1) }'                                   \
    | tr "'" " "
    ) )

  if [ "${tapp_arr[0]}" ]
  then
    output "Detected event code: 519"
    output "  --- Attempting to fix network '${tapp_arr[1]}' on ${host}."

    for nic in ${!VIRT_NET[@]}
    do
      t_nic_arr=( $( echo ${nic} | tr '|' ' ' ) )
      if [ "${t_nic_arr[0]}" = "${host}" ]                \
        && [ "${VIRT_NET[${nic}]}" = "${tapp_arr[1]}" ]
      then
        fix_host_network ${host} ${t_nic_arr[1]} ${tapp_arr[1]} 2>/dev/null
      fi
    done
  fi
}
```

8.1.16 fix_host_network()

The **fix_host_network()** function is called by the **REST_approve_function()**. This function accepts three arguments, the hypervisor host name, the interface hardware address, and a network name. This function attempts to attach a virtual network to a network interface on a hypervisor host. It queries the host for the UID of the interface that matches the hardware address specified when the function was called. Once the interface is attached, the function commits the network changes.

```
fix_host_network() {
  thost=$1
  tnic=$2
  tnet=$3

  fix_arr=( $(
    REST_submit GET "hosts/${HH[${thost}]}/nics"                          \
    | xpath "/host_nics/host_nic/@id | /host_nics/host_nic/mac/@address" \
    | sed -e s'/id=/\n/g' -e 's/"\|address=//g'                           \
    | grep -i ${tnic}
    ) )

  output "   --- Attaching nic ${tnic} to network ${tnet}"
  tmp_out=$( REST_submit POST hosts/${HH[${thost}]}/nics/$
{fix_arr[0]}/attach "<action><network id=\"${DCN[${tnet}]}\"/></action>"
2>&1 )
  output "${tmp_out}" log

  tmp_out=$( REST_submit POST hosts/${HH[${thost}]}/commitnetconfig
"<action/>" 2>&1 )
  output "${tmp_out}" log
}
```

8.1.17 REST_activate_host()

The **REST_activate_host()** function accepts a single argument specifying a host name to activate. The function then attempts to activate the host.

```
REST_activate_host() {
  thost=$1

  output "   --- Activating Host"
  tmp_out=$( REST_submit POST hosts/${HH[${thost}]}/activate "<action/>"
2>&1 )
  output "${tmp_out}" log
}
```

8.1.18 REST_fence_host()

The **REST_fence_host()** function accepts a single argument specifying a host name to fence. The function then attempts to fence the host.

```
REST_fence_host() {
 thost=$1

  output "   --- Fencing Host"
  tmp_out=$( REST_submit POST hosts/${HH[${thost}]}/fence
"<action><fence_type>manual</fence_type></action>" 2>&1 )
  output "${tmp_out}" log
}
```

8.1.19 fix_rhevm_networking()

The **fix_rhevm_networking()** function uses the keys and values stored in the **RHEVM_NIC[]** array as a reference to comment the lines in the */etc/udev/rules.d/70-persistent-net.rules* file that are not in the **RHEVM_NIC[]** array. The **RHEVM_NIC[]** array is populated from the configuration file.

After the lines have been commented, the function modifies the */etc/udev/rules.d/70-persistent-net.rules* file to map the correct interface name to the network hardware address.

The function then modifies the */etc/sysconfig/network-script/ifcfg-** files by commenting any lines beginning with **HWADDR**.

Finally the function suggests rebooting the system and then exits the script.

```
fix_rhevm_networking() {
  # Comment out any lines in the 70-persistent-net.rules that are not needed
  #
  output "Removing old interfaces."
  string=$( echo "${!RHEVM_NIC[@]}" | sed 's/ /\\|/g' )

  sed -i "/^#\|^$\|${string}/! s/^/# /" /etc/udev/rules.d/70-persistent-
net.rules

  # Name the interfaces appropriately.
  #
  output "Renaming interfaces:"
  for mac in ${!RHEVM_NIC[@]}
  do
    output "   --- ${mac} to ${RHEVM_NIC[${mac}]}"
    sed -i "/${mac}/ s/NAME=\".*\"/NAME=\"${RHEVM_NIC[${mac}]}\"/"
/etc/udev/rules.d/70-persistent-net.rules
  done


  # Fix the ifcfg-eth* files so they do not have HWADDR lines defined.
  #
  output "Removing HWADDR lines from interface configuration files."
```

```
  sed -i 's/^HWADDR/# &/' /etc/sysconfig/network-scripts/ifcfg-*


  echo -e "\n\n\n\n"
  echo    "The system should now be rebooted to ensure everything comes up
correctly."
  echo -e "\n\n\n\n"

  exit
}
```

8.1.20 REST_get_vm()

The **REST_get_vm()** function queries the REST API to get the name of the virtual machines, their UIDs and the UID of the cluster they are currently in. The function also queries the creation time of the virtual machine. The creation date is used as a delimiter for the **sed** command and is discarded.

The function uses the name of the virtual machine as a key for the **VM** array and the UID for the value. A second array called **VMCluster** is created, this array uses the name as the key and the cluster UID as the value.

```
REST_get_vm() {
  output "Getting VM UIDs..."

  while read uid name cl_uid create_time
  do
    VM[${name}]=${uid}
    VMCluster[${name}]=${cl_uid}
  done < <(
    REST_submit GET vms \
    | xpath '/vms/vm/@id | /vms/vm/name | /vms/vm/cluster/@id |
/vms/vm/creation_time' \
    | sed -e 's/id=\|"//g' -e 's/<name>\|<\/name>\|<creation_time>/ /g' -e
's/<\/creation_time>/\n/g'
    )
}
```

8.1.21 REST_get_networks()

The **REST_get_networks()** function queries the REST API for the networks that are configured for the datacenter specified by **SITE_INFO[datacenter]**. An associative array called **DCN** is populated with the information, using the network name as the key and the network UID as the value.

```
REST_get_networks() {
  output "Getting Networks for Datacenter ${SITE_INFO[datacenter]}..."

  while read uid name dc_uid state
  do
    if [ "${dc_uid}" = "${DC[${SITE_INFO[datacenter]}]}" ]
    then
      DCN[${name}]=${uid}
```

```
    fi
  done < <(
    REST_submit GET networks \
    | xpath '/networks/network/@id | /networks/network/name |
/networks/network/data_center/@id | /networks/network/status/state' \
    | sed -e 's/id=\|"//g' -e 's/<name>\|<\/name>\|<state>/ /g' -e
's/<\/state>/\n/g'
    )
}
```

8.1.22 REST_move_vms()

The **REST_move_vms()** function moves virtual machines in the cluster defined by **SITE_INFO[main_site_cluster]** and modifies their settings to place them in the cluster defined by the **SITE_INFO[bkup_site_cluster]** configuration option.

```
REST_move_vms() {
  output "Moving Virtual Machines from cluster $
{SITE_INFO[main_site_cluster]} to cluster ${SITE_INFO[bkup_site_cluster]}."

  for vm in ${!VM[@]}
  do
    if [ "${VMCluster[${vm}]}" = "${HC[${SITE_INFO[main_site_cluster]}]}" ]
    then
      output "   --- ${vm}"
      tmp_out=$( REST_submit "PUT" vms/${VM[${vm}]} "<vm><cluster><name>$
{SITE_INFO[bkup_site_cluster]}</name></cluster></vm>" 2>&1 )
      output "${tmp_out}" log
    fi
  done
}
```

8.1.23 fix_nfs_domain()

The **fix_nfs_domain()** function updates the RHEV Managers database to reflect the NFS information stored in the NFS[] array.

```
fix_nfs_domain() {
  sname=$1
  id=$( echo "SELECT storage FROM storage_domain_static WHERE
storage_name='${sname}';" \
    | psql -d rhevm -U postgres --tuples-only | sed 's/^\s*//' )

  echo "UPDATE storage_server_connections SET connection='${NFS[${sname}]}'
WHERE id='${id}';" \
    | psql -d rhevm -U postgres --tuples-only
}
```

8.1.24 usage()

A function to display the options that the script takes is useful. The **usage()** function displays this information and then exits the script.

```
usage() {
  echo -e "\nusage: $0 OPTIONS\n"
  echo "  -c cfg_file  : Specify the configuration file to use."
  echo "  -n           : Fix the RHEV Manager networking."
  echo "  -d           : Fix the storage information in the database."
  echo "  -h           : Fix the host and virtual machines."
  echo "  -s           : Start the virtual machines."
  echo "  -?           : Display usage."

  exit
}
```

8.1.25 read_cfg()

The **read_cfg()** function reads the configuration file and places its contents into the appropriate associative arrays. The associative arrays must be defined before this function is called.

The function requires a single argument when it is called. This argument is the name of the configuration file to use. If the configuration file exists, all blank lines and lines beginning with a pound (#) or semicolon (;) are removed.

The first entry on a line is considered a type of configuration entry. This entry is read and converted to all lower case characters. The case statement is used to determine which associative arrays to assign the entry and how to parse the line.

```
read_cfg() {
  #
  # This function is passed the name of the configuration file to read.
  # Normal invocation: read_cfg /some/file_name
  #

  while read type key value1 value2 value3
  do
    type=$( echo ${type} | tr [:upper:] [:lower:] )
    case ${type} in
      rhevm_nic)
        RHEVM_NIC[${key}]=${value1}
        ;;
      config)
        CONFIG[${key}]=${value1}
        ;;
      api)
        API[${key}]=${value1}
        ;;
      db)
        DB[${key}]=${value1}
        ;;
      storage_domain)
        case ${value1} in
```

```
          iscsi)
            ISCSI[${key}]="${value2}|${value3}"
            ;;
          fibre)
            FIBRE[${key}]="${value2}|${value3}"
            ;;
          nfs)
            NFS[${key}]=${value2}
            ;;
        esac
        ;;
      site_info)
        SITE_INFO[${key}]=${value1}
        ;;
      virt_net)
        VIRT_NET[${key}|${value1}]=${value2}
        ;;
      start_vm)
        START_VM[${#START_VM[@]}]="${key}|${value1}"
        ;;
    esac
  done < <(
    cat $1 | grep --extended-regexp --invert-match "^\s*$|^;|^#"
    )
}
```

8.2 Main Script

The main part of the script sets up the script environment, checks command line options, and controls how the functions are called. The interpreter part of the script is placed before the functions, while the rest of the script is appended below the functions section.

8.2.1 Interpreter

The script uses the /bin/bash as its interpreter. The interpreter is specified using the following line. This line must be the first line in the script.

```
#! /bin/bash
```

8.2.2 Variable Declarations

The declaration of the variables used within the script can be done anywhere in the script that is outside of a function.

Associative arrays must be declared before they are used. Default values can be assigned to arrays in case the definitions have not been specified in the configuration file.

```
## Variable Declarations
declare -A API DB CONFIG
declare -A SITE_INFO VIRT_NET RHEVM_NIC
declare -A ISCSI FIBRE NFS
declare -A DC HC HH DCN
declare -A VM VMCluster
declare -A NEEDED_CMDS

NEEDED_CMDS[iscsiadm]=iscsi-initiator-utils
NEEDED_CMDS[xpath]=perl-XML-XPath

# Setup some deafult values for fallback.
API[api_port]=8443
API[api_base]=api
API[api_user]=admin@internal
API[api_cert]=ca.crt
API[api_cert_port]=8080
CONFIG[working_dir]=/tmp/rdr
DB[user]=postgres
```

8.2.3 Main Code

The main code begins by checking the command line options. The bash built-in **`getopts`** function is used to parse command line options.

```
## Main
##

while getopts ":c:ndhs" opt
do
  case ${opt} in
      c) CONFIG[cfg_file]=${OPTARG}
         ;;
      n) CONFIG[fix_net]=1
         ;;
      d) CONFIG[fix_db]=1
         ;;
      h) CONFIG[fix_env]=1
         ;;
      s) CONFIG[start_vms]=1
         ;;
      *) usage
         ;;
  esac
done
```

The script checks that the configuration file exists and can be read. If it can, the script continues to create the **CONFIG[working_dir]** directory.

```
if [ "${CONFIG[cfg_file]}" ]
then
  output "Reading configuration file ${CONFIG[cfg_file]}."

  if [ -e "${CONFIG[cfg_file]}" ]
  then
    read_cfg ${CONFIG[cfg_file]}
  else
    output "Configuration file ${OPTARG} does not exist."
    exit
  fi
else
  usage
fi
```

The directory specified as the working directory is checked for its existence. If it does not exist, it is created.

```
# Make sure the working directory exists
if [ "${CONFIG[working_dir]}" ] && [ ! -d ${CONFIG[working_dir]} ]
then
  mkdir -p ${CONFIG[working_dir]}
fi
```

If the script was called with the **-n** option, then it proceeds to fix the network configuration of the RHEV Manager. The **fix_rhevm_networking()** function will cause the script to exit.

```
# Fix the RHEV Managers network configuration.
if [ "${CONFIG[fix_net]}" ]
then
  output "Fixing the RHEV Managers networking."
  fix_rhevm_networking
fi
```

If the script was not called with the **-n** option, then check to see if the needed commands exist. This check is not necessary to fix the networking on the RHEV Manager since the **fix_rhevm_networking()** function does not use them.

```
output "Checking to ensure all needed commands are available for the
script."
check_needed_commands
```

If the script was called with the **-d** option, fix the RHEV Managers database. First back up the database in case something goes wrong. Stop the RHEV Manager service so the database does not get corrupted. Fix each of the storage domains and start the RHEV Manager service.

```
# Fix the RHEV Managers database for storage.
if [ "${CONFIG[fix_db]}" ]
then
  output "Backing up the RHEVM database."
  pg_dump --format custom --username ${DB[user]} --file $
{CONFIG[working_dir]}/${DB[backup_file]} rhevm

  output "Fixing the RHEV Managers database to fix storage issues."

  service jbossas stop
  dust_settle 3

  service jbossas status | grep -q running
  is_running=$?

  if [ ${is_running} -eq 0 ]
  then
    output "jbossas service is running. Failed to stop."
    output "   --- Aborting operation."
    exit
  fi

  output -e "\nFixing iSCSI Storage Domains"
  for sd in "${!ISCSI[@]}"
  do
    output "  --- ${sd}"

    get_iscsi_info ${sd}
    fix_iscsi_domain ${sd}
  done

  output -e "\nFixing NFS Storage Domains"
  for sd in "${!NFS[@]}"
  do
    output "  --- ${sd}"

    fix_nfs_domain ${sd}
  done

  output -e "\nFixing FIBRE Storage Domains"
  for sd in "${!FIBRE[@]}"
  do
    output "  --- ${sd}"

    get_scsi_domain ${sd}
    fix_scsi_domain ${sd}
  done

  service jbossas start
  output "Waiting to let jbossas services finish."
```

```
  dust_settle 30

fi
```

If the script was called with the **-h** option, fix the RHEV environment. The REST API is needed for this section, so ensure the RHEV Manager service is running. The first call to **dust_settle()** allows the RHEV Manager to determine which hosts are non-responsive before the script continues.

The certificate is retrieved from the RHEV Manager and the datacenter, cluster, host, and virtual machine information is gathered. The status of all hosts is retrieved and placed into a queue to be approved, activated, or fenced.

Hosts are then approved and the status checked again. Hosts are then activated and then fenced. The script sleeps to give the RHEV Manager time to elect a new Storage Pool Manager before moving the virtual machines to the SiteB cluster.

```
# Fix the RHEV environment hosts and virtual machines..
if [ "${CONFIG[fix_env]}" ]
then
  output "Fixing the RHEV Managers environment."
  service jbossas status | grep -q running
  is_running=$?

  if [ ${is_running} -eq 1 ]
  then
    output "jbossas service is not running. Starting service."
    service jbossas start
    dust_settle 3

    service jbossas status | grep -q running
    is_running=$?

    if [ ${is_running} -eq 1 ]
    then
      output "jbossas service is not running. Failed to start."
      output "   --- Aborting operation."
      exit
    fi
  fi

  dust_settle 120

  REST_get_cert

  REST_get_dc 2>/dev/null
  REST_get_hc 2>/dev/null
  REST_get_hh 2>/dev/null
  REST_get_vm 2>/dev/null
  REST_get_networks 2>/dev/null

  get_hosts_status

  for host in ${host_approve}
```

```
  do
    output "Approving Host: ${host}"
    REST_approve_host ${host} ${SITE_INFO[bkup_site_cluster]} 2>/dev/null
  done

  dust_settle 20
  get_hosts_status

 for host in ${host_activate}
  do
    output "Attempting to Activate Host: ${host}"
    REST_activate_host ${host}
  done

  output "Giving hosts time to finish activating."
  dust_settle 40

  get_hosts_status

  for host in ${host_fence}
  do
    output "Fencing Host: ${host}"
    REST_fence_host ${host}
  done


  output "Giving environment time to elect new Storage Pool Manager if
necessary."
  dust_settle 180

  REST_move_vms

fi
```

If the script was called with the **-s** option, then start the virtual machines that are defined in the configuration file.

```
if [ "${CONFIG[start_vms]}" ]
then
  REST_get_cert
  output "Starting the virtual machines."
  REST_get_vm 2>/dev/null

  for temp in "${START_VM[@]}"
  do
    tarr=( $( echo ${temp} | tr '|' ' ' ) )
    output "   --- ${tarr[0]} (${tarr[1]})"
    dust_settle ${tarr[1]}

    REST_submit POST vms/${VM[${tarr[0]}]}/start "<action/>" > /dev/null
2>&1
  done
fi
```

8.3 Failover NFS

The following example shows the process of performing a site level failover of an environment using an NFS Data Storage Domain. The failover is done using the created script.

1. Place the script and configuration files in the */root/rdr* directory on the SiteA RHEV Manager.
2. Power on the virtual machines at SiteA.
3. Once the virtual machines are completely booted, remove the power from the RHEV Manager, RHEV Hypervisor, and NFS server.
4. Simulate the LUN mirroring process.
5. Mount the LUNs to the correct servers and power them on.
6. Fix the networking on the RHEV Manager by executing the script using the –n option.

```
# cd /root/rdr

# ./rhevm-failover.sh -c rhevm-failover.cfg -n

Reading configuration file rhevm-failover.cfg.
Fixing the RHEV Managers networking.
Removing old interfaces.
Renaming interfaces:
   --- 00:25:b3:a9:b0:02 to eth2
   --- 00:25:b3:a9:b0:03 to eth3
   --- 00:25:b3:a9:b0:00 to eth1
   --- 00:25:b3:a9:b0:01 to eth0
Removing HWADDR lines from interface configuration files.




The system should now be rebooted to ensure everything comes up correctly.
```

7. Reboot the RHEV Manager.

```
# init 6
```

8. Edit the scripts configuration file and verify the configuration for the storage domains.

```
Storage_Domain   DataDomain nfs   rdr-nfs.siteb.example:/exports/DATA
Storage_Domain   ISODomain       nfs   rdr-nfs.siteb.example:/exports/ISO
Storage_Domain   ExportDomain    nfs   rdr-nfs.siteb.example:/exports/EXPORT
```

9. Configure the RHEV Hypervisor. Configure networking, remote ssh access, then join it to the RHEV Manager.

10. Fix the RHEV Managers database, the environment and start the virtual machines. This is done by executing the script with the **-d**, **-h**, and **-s** options.

```
# ./rhevm-failover.sh -c rhevm-failover.cfg -d hs
Reading configuration file rhevm-failover.cfg.
Checking to ensure all needed commands are available for the script.
xpath: not found and is needed.
     This command may be provided by the perl-XML-XPath package.
iscsiadm: not found and is needed.
     This command may be provided by the iscsi-initiator-utils package.
Exiting...
```

11. Since the original RHEV Manager was installed using the base configuration, it did not have the **xpath** and **iscsiadm** packages installed. These commands are needed by the script. Install the **perl-XML-XPath** and **iscsi-initiator-utils** packages to provide the needed commands.

```
# yum install perl-XML-XPath iscsi-initiator-utils
Loaded plugins: product-id, rhnplugin, security, subscription-manager,
versionlock
Updating certificate-based repositories.

[ ... OUTPUT REMOVED FOR BREVITY ... ]

Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : iscsi-initiator-utils-6.2.0.872-34.el6.x86_64
1/2
  Installing : perl-XML-XPath-1.13-10.el6.noarch
2/2
Installed products updated.

Installed:
  iscsi-initiator-utils.x86_64 0:6.2.0.872-34.el6
perl-XML-XPath.noarch 0:1.13-10.el6

Complete!
```

12. Once the packages are installed, execute the script again.

```
# ./rhevm-failover.sh -c rhevm-failover.cfg -dhs
Reading configuration file rhevm-failover.cfg.
Checking to ensure all needed commands are available for the script.
Backing up the RHEVM database.
Fixing the RHEV Managers database to fix storage issues.
Stopping jbossas:                                          [  OK  ]



  --- DataDomain
```

```
UPDATE 1
  --- ISODomain
UPDATE 1
  --- ExportDomain
UPDATE 1

Starting jbossas:                                          [  OK  ]
Waiting to let jbossas services finish.

Fixing the RHEV Managers environment.

Retrieving certificate..... Done
Getting Datacenter UIDs...
Getting Cluster UIDs...
Getting Host UIDs...
Getting VM UIDs...
Getting Networks for Datacenter RefArch...
Getting status of Hypervisor Hosts
Approving Host: rdr-rhevh.siteb.example
Giving hosts time to finish approval process.

Detected event code: 519
  --- Attempting to fix network 'public' on rdr-rhevh.siteb.example.
   --- Attaching nic 00:17:A4:77:24:3A to network public

Getting status of Hypervisor Hosts
Attempting to Activate Host: rdr-rhevh.siteb.example
   --- Activating Host
Giving hosts time to finish activating.

Getting status of Hypervisor Hosts
Fencing Host: rdr-rhevh.sitea.example
   --- Fencing Host
Giving environment time to elect new Storage Pool Manager if necessary.

Moving Virtual Machines from cluster SiteA to cluster SiteB.
   --- rdr-devel-vm
   --- rdr-web-vm
   --- rdr-W7
   --- rdr-w2k8
Retrieving certificate..... Done
Starting the virtual machines.
Getting VM UIDs...
   --- rdr-devel-vm (5)

   --- rdr-w2k8 (10)

   --- rdr-W7 (5)

   --- rdr-web-vm (5)
```

The virtual machines are now running in the new environment.

8.4 Failover iSCSI

The following sequence shows a site level failover of an environment using an iSCSI Data Storage Domain. The failover is done using the created script.

1. Place the script and configuration files in the */root/rdr* directory on the SiteA RHEV Manager.
2. Power on the virtual machines at SiteA.
3. Once the virtual machines are completely booted, remove the power from the RHEV Manager, RHEV Hypervisor, and NFS server.
4. Simulate the LUN mirroring process.
5. Present the LUNs to the correct servers and power them on.
6. Fix the networking on the RHEV Manager by executing the script using the **–n** option.

```
# cd /root/rdr

# ./rhevm-failover.sh -c rhevm-failover.cfg -n

Reading configuration file rhevm-failover.cfg.
Fixing the RHEV Managers networking.
Removing old interfaces.
Renaming interfaces:
   --- 00:25:b3:a9:b0:02 to eth2
   --- 00:25:b3:a9:b0:03 to eth3
   --- 00:25:b3:a9:b0:00 to eth1
   --- 00:25:b3:a9:b0:01 to eth0
Removing HWADDR lines from interface configuration files.




The system should now be rebooted to ensure everything comes up correctly.
```

7. Reboot the RHEV Manager.

```
# init 6
```

8. Configure the RHEV Hypervisor. Configure networking, remote ssh access, then join it to the RHEV Manager.
9. Edit the scripts configuration file and verify the configuration for the storage domains.

```
Storage_Domain   DataDomain iscsi 192.168.201.32 iqn.2011-
11.siteb.example:rdr-iscsi.dat a
Storage_Domain   ISODomain       nfs   rdr-nfs.siteb.example:/exports/ISO
Storage_Domain   ExportDomain    nfs   rdr-nfs.siteb.example:/exports/EXPORT
```

10. Install the **perl-XML-XPath** and **iscsi-initiator-utils** packages.

```
# yum install perl-XML-XPath iscsi-initiator-utils
Loaded plugins: product-id, rhnplugin, security, subscription-manager,
versionlock
Updating certificate-based repositories.

[ ... OUTPUT REMOVED FOR BREVITY ... ]

Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : iscsi-initiator-utils-6.2.0.872-34.el6.x86_64
1/2
  Installing : perl-XML-XPath-1.13-10.el6.noarch
2/2
Installed products updated.

Installed:
  iscsi-initiator-utils.x86_64 0:6.2.0.872-34.el6
perl-XML-XPath.noarch 0:1.13-10.el6

Complete!
```

11. Fix the RHEV Managers database, the environment and start the virtual machines. This is done by executing the script with the **-d**, **-h**, and **-s** options.

```
# ./rhevm-failover.sh -c rhevm-failover.cfg -dhs
Reading configuration file rhevm-failover.cfg.
Checking to ensure all needed commands are available for the script.
Backing up the RHEVM database.
Fixing the RHEV Managers database to fix storage issues.
Stopping jbossas:                                          [  OK  ]


  --- DataDomain
Starting iscsid:                                           [  OK  ]
192.168.201.32:3260,1 iqn.2011-11.siteb.example:rdr-iscsi.data
Logging in to [iface: default, target: iqn.2011-11.siteb.example:rdr-
iscsi.data, portal: 192.168.201.32,3260] (multiple)
Login to [iface: default, target: iqn.2011-11.siteb.example:rdr-iscsi.data,
portal: 192.168.201.32,3260] successful.
Logging out of session [sid: 1, target: iqn.2011-11.siteb.example:rdr-
iscsi.data, portal: 192.168.201.32,3260]
Logout of [sid: 1, target: iqn.2011-11.siteb.example:rdr-iscsi.data, portal:
192.168.201.32,3260] successful.
INSERT 34956 1
UPDATE 1
DELETE 1
UPDATE 1

  --- ISODomain
```

```
UPDATE 1
  --- ExportDomain
UPDATE 1

Starting jbossas:                                          [  OK  ]
Waiting to let jbossas services finish.

Fixing the RHEV Managers environment.

Retrieving certificate..... Done
Getting Datacenter UIDs...
Getting Cluster UIDs...
Getting Host UIDs...
Getting VM UIDs...
Getting Networks for Datacenter RefArch...
Getting status of Hypervisor Hosts
Approving Host: rdr-rhevh.siteb.example
Giving hosts time to finish approval process.

Detected event code: 519
  --- Attempting to fix network 'public' on rdr-rhevh.siteb.example.
   --- Attaching nic 00:17:A4:77:24:3A to network public

Getting status of Hypervisor Hosts
Attempting to Activate Host: rdr-rhevh.siteb.example
   --- Activating Host
Giving hosts time to finish activating.

Getting status of Hypervisor Hosts
Fencing Host: rdr-rhevh.sitea.example
   --- Fencing Host
Giving environment time to elect new Storage Pool Manager if necessary.

Moving Virtual Machines from cluster SiteA to cluster SiteB.
   --- rdr-devel-vm
   --- rdr-web-vm
   --- rdr-W7
   --- rdr-w2k8
Retrieving certificate..... Done
Starting the virtual machines.
Getting VM UIDs...
   --- rdr-devel-vm (5)

   --- rdr-w2k8 (10)

   --- rdr-W7 (5)

   --- rdr-web-vm (5)
```

The virtual machines are now running in the new environment.

8.5 Failover Fibre

The following shows the process of performing a site level failover of an environment using an iSCSI Data Storage Domain. The failover is done using the created script.

1. Place the script and configuration files in the */root/rdr* directory on the SiteA RHEV Manager.
2. Power on the virtual machines at SiteA.
3. Once the virtual machines are completely booted, remove the power from the RHEV Manager, RHEV Hypervisor, and NFS server.
4. Simulate the LUN mirroring process.
5. Present the LUNs to the correct servers and power them on.
6. Fix the networking on the RHEV Manager by executing the script using the **–n** option.

```
# cd /root/rdr

# ./rhevm-failover.sh -c rhevm-failover.cfg -n

Reading configuration file rhevm-failover.cfg.
Fixing the RHEV Managers networking.
Removing old interfaces.
Renaming interfaces:
   --- 00:25:b3:a9:b0:02 to eth2
   --- 00:25:b3:a9:b0:03 to eth3
   --- 00:25:b3:a9:b0:00 to eth1
   --- 00:25:b3:a9:b0:01 to eth0
Removing HWADDR lines from interface configuration files.




The system should now be rebooted to ensure everything comes up correctly.
```

7. Reboot the RHEV Manager.

```
# init 6
```

8. Generate an ssh key pair for root on the RHEV Manager.

```
# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
60:fa:d2:47:cd:3c:54:c8:15:4c:02:2a:0e:04:c8:18 root@rdr-rhevm.sitea.example
The key's randomart image is:
```

```
+--[ RSA 2048]----+
|Eo.     .o.==.   |
|oo     .  oo.    |
|  . . +   .      |
|   o + . =       |
|    o   S =      |
|     o .   .     |
|    . o .        |
|     . .         |
|                 |
+-----------------+
```

9. Configure the RHEV Hypervisor. Configure the networking, remote ssh access, and join it to the RHEV Manager.
10. Place the public key of the RHEV Managers root user in the */root/.ssh/authorized keys* file on the RHEV Hypervisor.

```
# ssh rdr-rhevm.sitea.example "cat /root/.ssh/id_rsa.pub" >> authorized_keys
root@rdr-rhevm.sitea.example's password: [PASSWORD]
```

11. Edit the scripts configuration file and verify the configuration for the storage domains.

```
Storage_Domain   DataDomain  fibre rdr-rhevh.siteb.example
00c0ffd7e69d000085b89d4e01000000
Storage_Domain   ISODomain        nfs   rdr-nfs.siteb.example:/exports/ISO
Storage_Domain   ExportDomain     nfs   rdr-nfs.siteb.example:/exports/EXPORT
```

12. Install the **perl-XML-XPath** and **iscsi-initiator-utils** packages.

```
# yum install perl-XML-XPath iscsi-initiator-utils
Loaded plugins: product-id, rhnplugin, security, subscription-manager,
versionlock
Updating certificate-based repositories.

[ ... OUTPUT REMOVED FOR BREVITY ... ]

Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : iscsi-initiator-utils-6.2.0.872-34.el6.x86_64
1/2
  Installing : perl-XML-XPath-1.13-10.el6.noarch
2/2
Installed products updated.

Installed:
  iscsi-initiator-utils.x86_64 0:6.2.0.872-34.el6
perl-XML-XPath.noarch 0:1.13-10.el6

Complete!
```

13. Fix the RHEV Managers database, the environment and start the virtual machines. This is done by executing the script with the **-d**, **-h**, and **-s** options.

```
# ./rhevm-failover.sh -c rhevm-failover.cfg -dhs
Reading configuration file rhevm-failover.cfg.
Checking to ensure all needed commands are available for the script.
Backing up the RHEVM database.
Fixing the RHEV Managers database to fix storage issues.
Stopping jbossas:                                          [  OK  ]



  --- ISODomain
UPDATE 1
  --- ExportDomain
UPDATE 1

  --- DataDomain
The authenticity of host 'rdr-rhevh.siteb.example (192.168.201.42)' can't be
established.
RSA key fingerprint is 2b:d7:37:7b:97:2f:8a:c5:ec:78:46:95:19:50:1e:02.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'rdr-rhevh.siteb.example,192.168.201.42' (RSA) to
the list of known hosts.
UPDATE 1
Starting jbossas:                                          [  OK  ]
Waiting to let jbossas services finish.

Fixing the RHEV Managers environment.

Retrieving certificate..... Done
Getting Datacenter UIDs...
Getting Cluster UIDs...
Getting Host UIDs...
Getting VM UIDs...
Getting Networks for Datacenter RefArch...
Getting status of Hypervisor Hosts
Approving Host: rdr-rhevh.siteb.example
Giving hosts time to finish approval process.

Detected event code: 519
  --- Attempting to fix network 'public' on rdr-rhevh.siteb.example.
   --- Attaching nic 00:17:A4:77:24:3A to network public

Getting status of Hypervisor Hosts
Attempting to Activate Host: rdr-rhevh.siteb.example
   --- Activating Host
Giving hosts time to finish activating.

Getting status of Hypervisor Hosts
Fencing Host: rdr-rhevh.sitea.example
   --- Fencing Host
Giving environment time to elect new Storage Pool Manager if necessary.

Moving Virtual Machines from cluster SiteA to cluster SiteB.
```

```
   --- rdr-devel-vm
   --- rdr-web-vm
   --- rdr-W7
   --- rdr-w2k8
Retrieving certificate..... Done
Starting the virtual machines.
Getting VM UIDs...
   --- rdr-devel-vm (5)

   --- rdr-w2k8 (10)

   --- rdr-W7 (5)

   --- rdr-web-vm (5)
```

The virtual machines are now running in the new environment.

9 Conclusion

This paper demonstrated the site wide failover and recovery of three separate RHEV environments.

- NFS Data Storage Domain
- iSCSI Data Storage Domain
- Fibre Data Storage Domain

Simulated LUN mirroring was used to replicate data volumes across sites. Each site consisted of individual data domains and network infrastructures.

DHCP and DNS were used to reduce site recovery times. Name resolution issues were discussed and solutions provided.

The use of simple scripting and the REST API to assist in the site recovery process was shown.

The goals achieved in this paper include, failing a RHEV environment over to a designated backup site during a site level disaster. A quick recovery of the failed over environment can be performed easily through the use of scripts. The scripts automate the tasks involved and minimize the time involved to restore the environment to a working state.

Appendix A: DNS Configuration Files

A.1 SiteA

/etc/named.conf

```
options {
      listen-on port 53 { 127.0.0.1; 192.168.200.34; 10.16.136.0/21; };
      listen-on-v6 port 53 { ::1; };
      directory  "/var/named";
      dump-file  "/var/named/data/cache_dump.db";
        statistics-file "/var/named/data/named_stats.txt";
        memstatistics-file "/var/named/data/named_mem_stats.txt";
      recursion yes;

      forwarders { 10.16.143.247; 10.16.143.248; };
};

logging {
        channel default_debug {
                file "data/named.run";
                severity dynamic;
        };
};

zone "." IN {
      type hint;
      file "named.ca";
};

zone "sitea.example" IN {
      type master;
      file "named.sitea.example";
      allow-update { none; };
};

zone "200.168.192.in-addr.arpa" IN {
      type master;
      file "reverse.sitea.example";
      allow-update { none; };
};

include "/etc/named.rfc1912.zones";
```

/var/named/named.sitea.example

```
$ORIGIN sitea.example.
$TTL 3H
@     IN SOA      rdr-dns.sitea.example. root.sitea.example. (
                              1     ; serial
                              1D    ; refresh
                              1H    ; retry
                              1W    ; expire
                              3H )  ; minimum

      IN    NS    rdr-dns.sitea.example.

rdr-gateway IN    A     192.168.200.1
rdr-nfs           IN    A     192.168.200.30
rdr-iscsi   IN    A     192.168.200.32
rdr-dns           IN    A     192.168.200.34
rdr-rhevm   IN    A     192.168.200.40
rdr-rhevh   IN    A     192.168.200.42
rdr-ie            IN    A     192.168.200.36
rdr-dc-vm   IN    A     192.168.200.50
rdr-w7-vm   IN    A     192.168.200.52
rdr-web-vm  IN    A     192.168.200.54
rdr-devel-vm      IN    A     192.168.200.56
```

/var/named/reverse.sitea.example

```
$ORIGIN 200.168.192.in-addr.arpa.
$TTL 3H
@     IN SOA      rdr-dns.sitea.example. root.sitea.example. (
                              1     ; serial
                              1D    ; refresh
                              1H    ; retry
                              1W    ; expire
                              3H )  ; minimum

      IN    NS    rdr-dns.sitea.example.

1     IN    PTR   rdr-gateway.sitea.example.
30    IN    PTR   rdr-nfs.sitea.example.
32    IN    PTR   rdr-iscsi.sitea.example.
34    IN    PTR   rdr-dns.sitea.example.
40    IN    PTR   rdr-rhevm.sitea.example.
42    IN    PTR   rdr-rhevh.sitea.example.
36    IN    PTR   rdr-ie.sitea.example.
50    IN    PTR   rdr-dc-vm.sitea.example.
52    IN    PTR   rdr-w7-vm.sitea.example.
54    IN    PTR   rdr-web-vm.sitea.example.
56    IN    PTR   rdr-devel-vm.sitea.example.
```

A.2 SiteB

/etc/named.conf

```
options {
      listen-on port 53 { 127.0.0.1; 192.168.201.34; 10.16.136.0/21; };
      listen-on-v6 port 53 { ::1; };
      directory  "/var/named";
      dump-file  "/var/named/data/cache_dump.db";
        statistics-file "/var/named/data/named_stats.txt";
        memstatistics-file "/var/named/data/named_mem_stats.txt";
      recursion yes;

      forwarders { 10.16.143.247; 10.16.143.248; };
};

logging {
        channel default_debug {
                file "data/named.run";
                severity dynamic;
        };
};

zone "." IN {
      type hint;
      file "named.ca";
};

zone "sitea.example" IN {
      type master;
      file "named.sitea.example";
      allow-update { none; };
};

zone "siteb.example" IN {
      type master;
      file "named.siteb.example";
      allow-update { none; };
};

zone "201.168.192.in-addr.arpa" IN {
      type master;
      file "reverse.siteb.example";
      allow-update { none; };
};

include "/etc/named.rfc1912.zones";
```

/var/named/named.sitea.example

```
$ORIGIN sitea.example.
$TTL 3H
@     IN SOA      rdr-dns.sitea.example. root.sitea.example. (
                              5     ; serial
                              1D    ; refresh
                              1H    ; retry
                              1W    ; expire
                              3H )  ; minimum

      IN    NS    rdr-dns.sitea.example.

rdr-rhevm   IN    A     192.168.201.40
rdr-dns           IN    A     192.168.201.34
```

/var/named/named.siteb.example

```
$ORIGIN siteb.example.
$TTL 3H
@     IN SOA      rdr-dns.siteb.example. root.siteb.example. (
                              5     ; serial
                              1D    ; refresh
                              1H    ; retry
                              1W    ; expire
                              3H )  ; minimum

      IN    NS    rdr-dns.siteb.example.

rdr-gateway IN    A     192.168.201.1
rdr-nfs           IN    A     192.168.201.30
rdr-iscsi   IN    A     192.168.201.32
rdr-dns           IN    A     192.168.201.34
rdr-rhevm   IN    A     192.168.201.40
rdr-rhevh   IN    A     192.168.201.42
rdr-ie            IN    A     192.168.201.36
rdr-dc-vm   IN    A     192.168.201.50
rdr-w7-vm   IN    A     192.168.201.52
rdr-web-vm  IN    A     192.168.201.54
rdr-devel-vm      IN    A     192.168.201.56
```

/var/named/reverse.siteb.example

```
$ORIGIN 201.168.192.in-addr.arpa.
$TTL 3H
@     IN SOA     rdr-dns.siteb.example. root.siteb.example. (
                              5     ; serial
                              1D    ; refresh
                              1H    ; retry
                              1W    ; expire
                              3H )  ; minimum

      IN    NS    rdr-dns.siteb.example.

1     IN    PTR   rdr-gateway.siteb.example.
30    IN    PTR   rdr-nfs.siteb.example.
32    IN    PTR   rdr-iscsi.siteb.example.
34    IN    PTR   rdr-dns.siteb.example.
42    IN    PTR   rdr-rhevh.siteb.example.
36    IN    PTR   rdr-ie.siteb.example.
50    IN    PTR   rdr-dc-vm.siteb.example.
52    IN    PTR   rdr-w7-vm.siteb.example.
54    IN    PTR   rdr-web-vm.siteb.example.
56    IN    PTR   rdr-devel-vm.siteb.example.

40    IN    PTR   rdr-rhevm.sitea.example.
```

Appendix B: DHCP Configuration Files

B.1 SiteA

/etc/dhcp/dhcpd.conf

```
option domain-name "sitea.example";
option domain-name-servers 192.168.200.34;
option routers 192.168.200.1;

default-lease-time 600;
max-lease-time 7200;

log-facility local7;

subnet 192.168.200.0 netmask 255.255.255.0 {
  range 192.168.200.100 192.168.200.120;
}

host rdr-gateway {
  hardware ethernet 52:54:00:48:bb:69;
  fixed-address 192.168.200.1;
}

host rdr-nfs {
  hardware ethernet 52:54:00:ee:2b:52;
  fixed-address 192.168.200.30;
}

host rdr-iscsi {
  hardware ethernet 52:54:00:0b:90:38;
  fixed-address 192.168.200.32;
}

host rdr-dns {
  hardware ethernet 52:54:00:0b:90:38;
  fixed-address 192.168.200.34;
}

host rdr-rhevm {
  hardware ethernet 00:1e:0b:ce:42:78;
  fixed-address 192.168.200.40;
}

host rhevh {
  hardware ethernet 00:17:a4:77:24:34;
#  hardware ethernet 00:17:a4:77:24:38;
  fixed-address 192.168.200.42;
}

host rdr-ie {
  hardware ethernet 52:54:00:00:8c:ed;
  fixed-address 192.168.200.36;
```

```
}

host rdr-dc-vm {
  hardware ethernet 00:16:3e:1e:0a:09;
  fixed-address 192.168.200.50;
}

host rdr-w7-vm {
  hardware ethernet 00:16:3e:7f:53:40;
  fixed-address 192.168.200.52;
}

host rdr-web-vm {
  hardware ethernet 00:16:3e:60:51:86;
  fixed-address 192.168.200.54;
}

host rdr-devel-vm {
  hardware ethernet 00:16:3e:69:70:0a;
  fixed-address 192.168.200.56;
}
```

B.2 SiteB

/etc/dhcp/dhcpd.conf

```
option domain-name "siteb.example";
option domain-name-servers 192.168.201.34;
option routers 192.168.201.1;

default-lease-time 600;
max-lease-time 7200;

log-facility local7;

subnet 192.168.201.0 netmask 255.255.255.0 {
  range 192.168.201.100 192.168.201.120;
}

host rdr-gateway {
  hardware ethernet 52:54:00:8c:95:5a;
  fixed-address 192.168.201.1;
}

host rdr-nfs {
  hardware ethernet 52:54:00:f8:df:5e;
  fixed-address 192.168.201.30;
}

host rdr-iscsi {
  hardware ethernet 52:54:00:aa:d5:7e;
  fixed-address 192.168.201.32;
}
```

```
host rdr-dns {
  hardware ethernet 52:54:00:ea:8e:6d;
  fixed-address 192.168.201.34;
}

host rdr-rhevm {
  hardware ethernet 00:25:b3:a9:b0:00;
  fixed-address 192.168.201.40;
}

host rdr-rhevh {
  hardware ethernet 00:17:a4:77:24:38;
  fixed-address 192.168.201.42;
}

host rdr-ie {
  hardware ethernet 52:54:00:f9:3b:11;
  fixed-address 192.168.201.36;
}

host rdr-dc-vm {
  hardware ethernet 00:16:3e:1e:0a:09;
  fixed-address 192.168.201.50;
}

host rdr-w7-vm {
  hardware ethernet 00:16:3e:7f:53:40;
  fixed-address 192.168.201.52;
}

host rdr-web-vm {
  hardware ethernet 00:16:3e:60:51:86;
  fixed-address 192.168.201.54;
}

host rdr-devel-vm {
  hardware ethernet 00:16:3e:69:70:0a;
  fixed-address 192.168.201.56;
}
```

Appendix C: Revision History

Revision 1.0	Tuesday January 17, 2012	John Herr
Initial Release		