Kvm Users Guide

7.0 Edition

Kvm Users Guide :

7.0 Edition
Published Dec 01 2017

Table of Contents

List of Tables

Preface

The KVM Roll installs and configures Virtual Machines (VMs) on Rocks Clusters.

A physical frontend can configure VMs on client nodes (*VM container* appliances). A VM container is a physical machine that houses and runs VMs.

The KVM Roll also supports building virtual clusters. The frontend can be installed as a *VM server* appliance and the client nodes can be installed as VM containers. Then a virtual frontend can be installed on the VM server while virtual compute nodes can be installed on the VM containers. All network traffic is encapsulated within a unique VLAN, that is, each virtual cluster has its own VLAN.

Please visit the KVM site[1] to learn more about their release and the individual software components.

Notes

1. http://www.linux-kvm.org

Chapter 1. Overview

Table 1-1. Summary

Name	kvm
Version	7.0
Maintained By	Rocks Group
Architecture	x86_64
Compatible with Rocks®	7.0

The kvm roll has the following requirements of other rolls. Compatability with all known rolls is assured, and all known conflicts are listed. There is no assurance of compatiblity with third-party rolls.

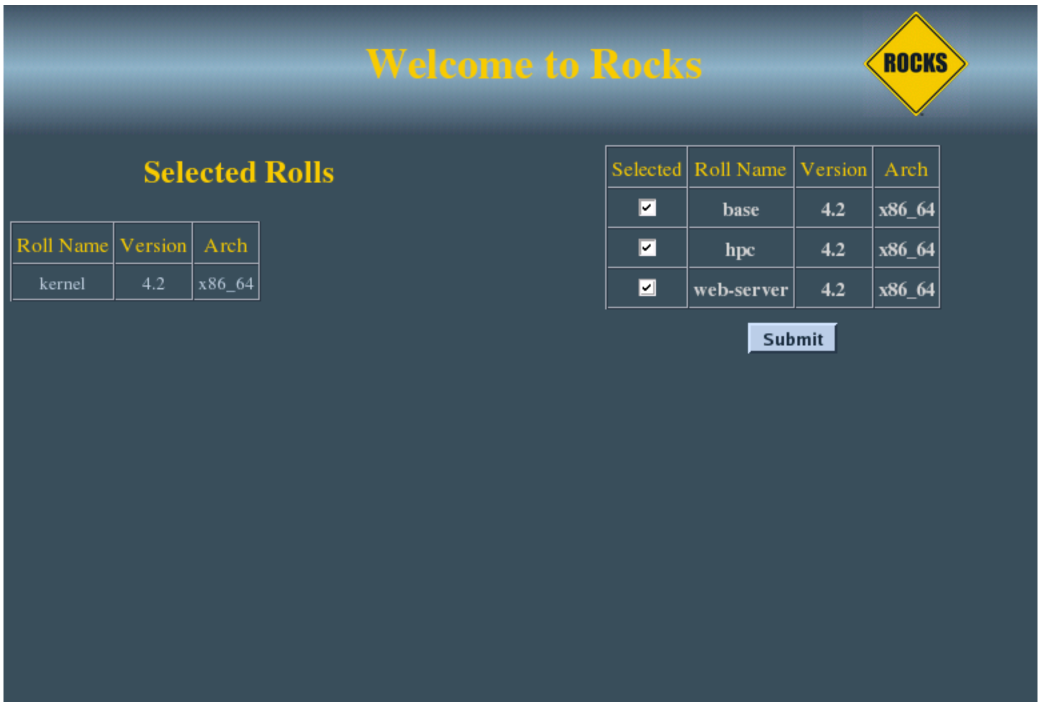
Table 1-2. Compatibility

Requires	Conflicts
Base	
Kernel	
OS	
	Xen

Chapter 2. Installing

2.1. On a New Server

The kvm roll should be installed during the initial installation of your server (or cluster). This procedure is documented in section 3.2 of the Rocks® usersguide. You should select the kvm roll from the list of available rolls when you see a screen that is similar to the one below.



Welcome to Rocks

Selected Rolls

Roll Name	Version	Arch
kernel	4.2	x86_64

Selected	Roll Name	Version	Arch
✓	base	4.2	x86_64
✓	hpc	4.2	x86_64
✓	web-server	4.2	x86_64

2.2. On an Existing Server

The kvm Roll may also be added onto an existing server (or frontend). For sake of discussion, assume that you have an iso image of the roll called `kvm.iso`. The following procedure will install the Roll, and after the server reboots the Roll should be fully installed and configured.

```
$ su - root
# rocks add roll kvm.iso
# rocks enable roll kvm
# cd /export/rocks/install
# rocks create distro
# rocks run roll kvm | bash
# init 6
```

Chapter 3. Using the KVM Roll

3.1. Installing a VM Server

A VM Server is a machine that can house virtual frontend appliances. It is required to build a VM Server if you wish to build virtual clusters.

Building a VM Server is just like building a traditional frontend, except that you *must have installed properly the KVM Roll*. Follow the procedure Install and Configure Your Frontend[1] and be sure to supply the KVM Roll.

After you build the VM Server, you'll need to install VM Containers (see the next section).

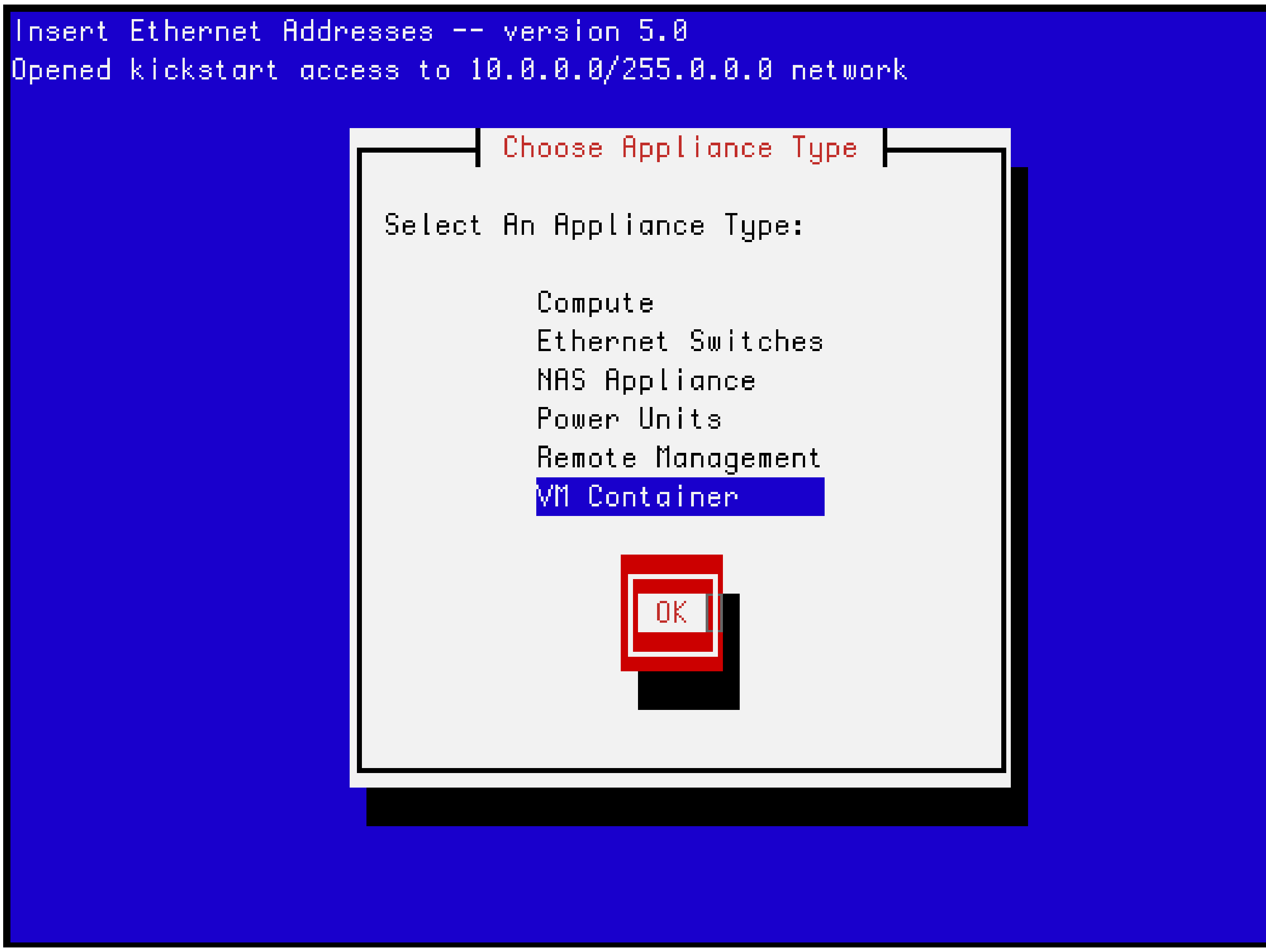
3.2. Installing VM Containers

You will need to install a physical machine that will act as the "container" for your VMs. This method is very similar to the method for installing compute nodes.

On the frontend, execute:

```
# insert-ethers
```

You will see a screen that looks like:



```
Insert Ethernet Addresses -- version 5.0
Opened kickstart access to 10.0.0.0/255.0.0.0 network
```

Select the ’VM Container’ appliance, then hit ’OK’.

Now PXE boot the physical machine that will be your VM container. Just like a compute node, the VM container will be recognized by insert-ethers and installed. The default name of the node will be `vm-container-X-Y`.

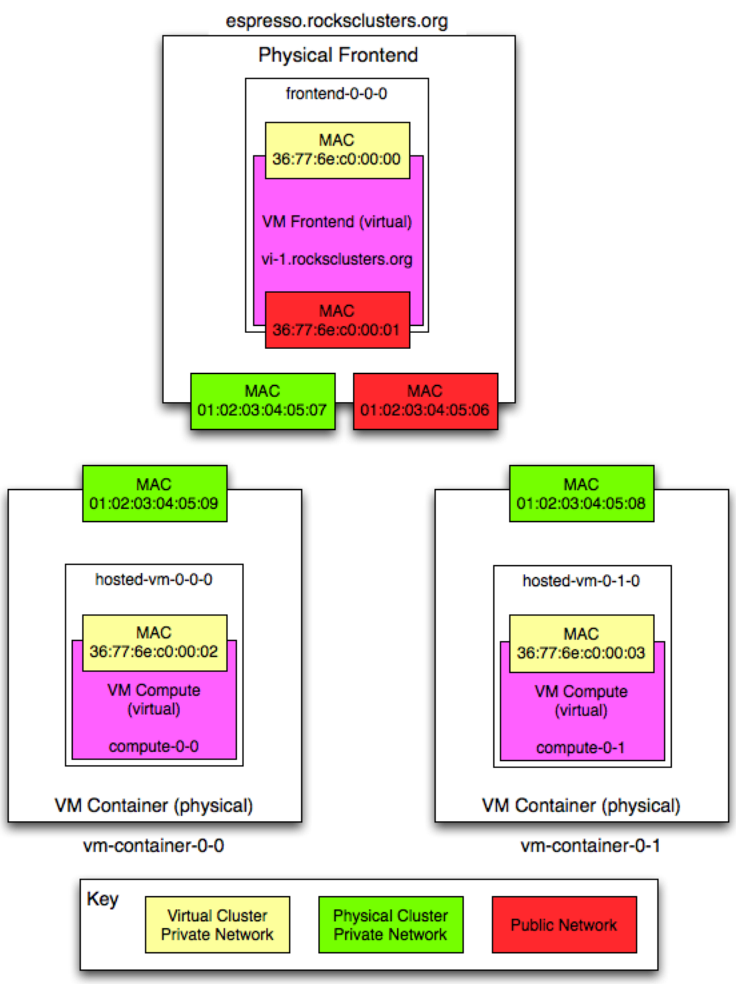
You can install as many VM containers as you like.

3.3. Installing Virtual Clusters

3.3.1. Provisioning a Virtual Cluster

After you install a VM Server and at least one VM Container, you are ready to provision a virtual cluster.

We’ll use the following illustration as a guide to help keep track of the names of the physical machines and the virtual machines.



In the above picture, "espresso.rocksclusters.org" is a physical machine. Also, "vm-container-0-0" and "vm-container-0-1" are physical machines that were kickstarted by "espresso". The machine "frontend-0-0-0" is a virtual machine that is hosted by "espresso". The machines "hosted-vm-0-0-0" and "hosted-vm-0-1-0" are VMs that are associated with "frontend-0-0-0" (they are all in the same VLAN).

Depending on your perspective, the virtual machines have different names. Dom0 is a physical machine that hosts (multiple) virtual systems. DomU are guests and generally refer to names by usual convention. The equivalence is:

Table 3-1. Virtual Machine Names

Host	Dom0 Name (physical)	DomU Name (virtual)
37:77:6e:c0:00:00	frontend-0-0-0	vi-1.rocksclusters.org
37:77:6e:c0:00:01	hosted-vm-0-0-0	compute-0-0
37:77:6e:c0:00:02	hosted-vm-0-1-0	compute-0-1

An important point is that the only common thing between the physical side and the virtual side is the MAC address (in yellow). We will use the MAC address of a virtual machine to control it (e.g., to initially power it on).

The names in the virtual cluster look like the names in a traditional cluster -- the frontend is named "vi-1.rocksclusters.org" and its compute nodes are named "compute-0-0" and "compute-0-1". If you login to "vi-1.rocksclusters.org", you would be hard pressed to tell the difference between this virtual cluster and a traditional physical cluster.

You must select your own IP address for your virtual frontend. The IP address "137.110.119.118" is managed by UCSD and should not be used by you.

They are only used here to show you a concrete example.

First, we'll add a virtual cluster to the VM Server's database. In this example, we'll add a frontend with the IP of "137.110.119.118" and we'll associate 2 compute nodes with it:

```
# rocks add cluster ip="137.110.119.118" num-computes=2
```

The above command will take some time and then output something similar to:

```
created frontend VM named: frontend-0-0-0
 created compute VM named: hosted-vm-0-0-0
 created compute VM named: hosted-vm-0-1-0
```

The command adds entries to the database for the above nodes and establishes a VLAN that will be used for the private network (eth0 inside the VM).

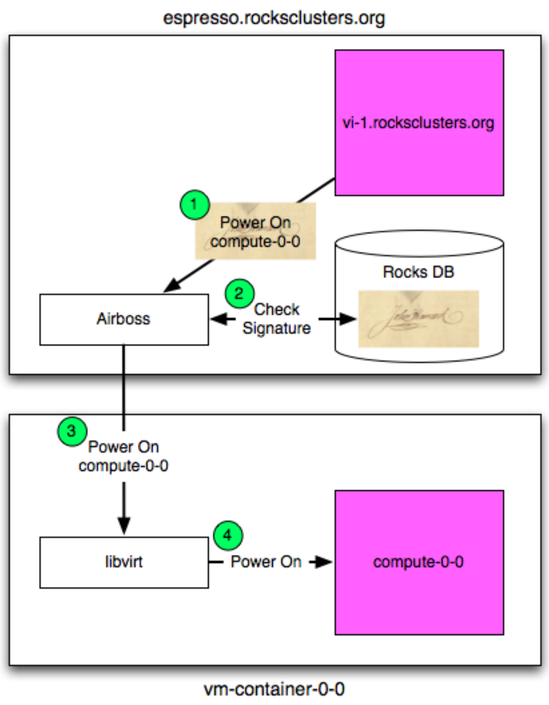
Info about all the defined clusters on the VM Server (including the physical cluster) can be obtained with the command: **rocks list cluster**:

```
# rocks list cluster
FRONTEND                    CLIENT NODES     TYPE
espresso.rocksclusters.org: ---------------- physical
:                           vm-container-0-0 physical
:                           vm-container-0-1 physical
frontend-0-0-0-public:      ---------------- VM
:                           hosted-vm-0-0-0  VM
:                           hosted-vm-0-1-0  VM
```

3.3.2. The Airboss

In Rocks, we've developed a service known as the "Airboss" that resides on the physical frontend (in Dom0) and it allows non-root users to control their VMs. The motivation for this service is that libvirt (a virtualization API written by RedHat that can control several different virtualization implementations) assumes "root" access to control and monitor VMs.

The Airboss in Rocks is a small service that uses digitally signed messages to give non-root users access to their virtual cluster (and only their virtual cluster). The Airboss relies upon public/private key pairs to validate messages. The administrator of the physical hosting cluster must issue a single command to associate a public key with a particular virtual cluster. At that point, the full process of booting and installing a virtual cluster can be controlled by the (authorized) non-root user.



In the above picture, a user that is logged in to vi-1.rocksclusters.org wants to power on compute-0-0 (one of the VMs associated with the virtual cluster). The user executes the "power on" command. The command creates a "power on" message, signs it with a private key, then sends it to the Airboss that is running on espresso.rocksclusters.org. The Airboss verifies the message signature. If the signature is valid, then the Airboss instructs libvirt on vm-container-0-0 to start ("power on") compute-0-0.

3.3.3. Creating an RSA Key Pair

Before we can install a VM, we must create an RSA key pair. These keys will be used to authenticate Airboss commands. To create a key pair, execute:

```
# rocks create keys key=private.key
```

The above command will ask for a pass phrase for the private key. If you would like a "passphraseless" private key, execute:

```
# rocks create keys key=private.key passphrase=no
```

The above command will place your private key into the file private.key and it will output the public key for your private key:

```
# rocks create keys key=private.key
Generating RSA private key, 1024 bit long modulus
............++++++
.......++++++
e is 65537 (0x10001)
Enter pass phrase for private.key:
Verifying - Enter pass phrase for private.key:
Enter pass phrase for private.key:
writing RSA key
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDMoCPmR/Kev64znRBxvtsniXIF
```

```
dyQMxR/bBFKNDmvmzPuPUim5jmD3TLilnH75/KidtJCwlb+Lhr5Cs6/9sRzX6rX2
ExVUZsgo4A+O+XMk8KeowO/c2rPc+YdXaBir3Aesm/MCfCZaidZae8QLmVKW7Va5
qErl9gyhhR7uDX+hgwIDAQAB
-----END PUBLIC KEY-----
```

Now save the public key to file, that is, copy the above public key:

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDMoCPmR/Kev64znRBxvtsniXIF
dyQMxR/bBFKNDmvmzPuPUim5jmD3TLilnH75/KidtJCwlb+Lhr5Cs6/9sRzX6rX2
ExVUZsgo4A+O+XMk8KeowO/c2rPc+YdXaBir3Aesm/MCfCZaidZae8QLmVKW7Va5
qErl9gyhhR7uDX+hgwIDAQAB
-----END PUBLIC KEY-----
```

And save your public key into a file (e.g., $HOME/public.key).

We now want to associate your public key with the virtual cluster you provisioned. This will allow you to use your private key to send authenticated commands to control your cluster. To associate your public key with your virtual cluster, execute:

```
# rocks add host key frontend-0-0-0 key=public.key
```

We can see the relationship by executing:

```
# rocks list host key
HOST            ID PUBLIC KEY
frontend-0-0-0: 7  -----BEGIN PUBLIC KEY-----
:                  MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDMoCPmR/Kev64znRBxvtsniXIF
:                  dyQMxR/bBFKNDmvmzPuPUim5jmD3TLilnH75/KidtJCwlb+Lhr5Cs6/9sRzX6rX2
:                  ExVUZsgo4A+O+XMk8KeowO/c2rPc+YdXaBir3Aesm/MCfCZaidZae8QLmVKW7Va5
:                  qErl9gyhhR7uDX+hgwIDAQAB
:                  -----END PUBLIC KEY-----
:                  ----------------------------------------------------------------
```

We see that the public key is associated with "frontend-0-0-0" (the name of the VM in Dom0).

3.3.4. Installing a VM Frontend

Now, we'll want to install the virtual frontend. First, login to the physical frontend (e.g., espresso). To start the VM frontend install, we'll need to power on and install the VM frontend:

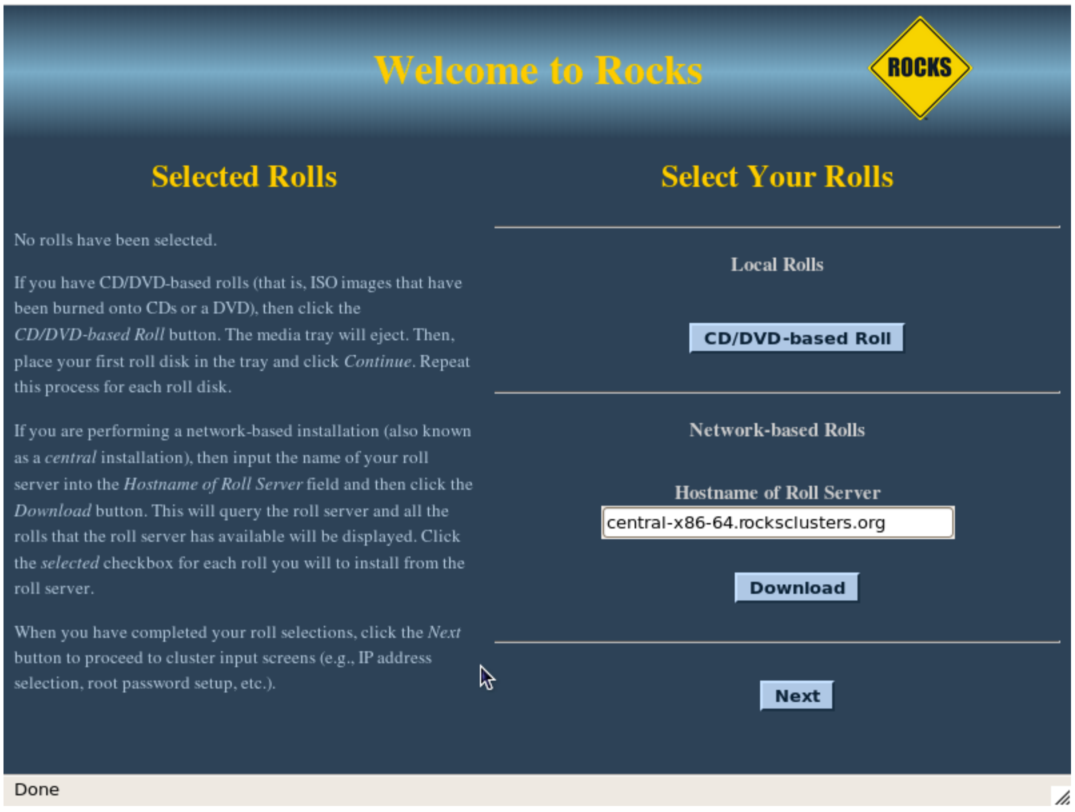
```
# rocks set host power frontend-0-0-0 action=install key=private.key
```

The action of "install" ensures that the VM will be put into install mode, then it will be powered on.

Then, to connect to the VM's console, execute:

```
# rocks open host console frontend-0-0-0 key=private.key
```

Soon you will see the familiar frontend installation screen:



Selected Rolls

No rolls have been selected.

If you have CD/DVD-based rolls (that is, ISO images that have been burned onto CDs or a DVD), then click the *CD/DVD-based Roll* button. The media tray will eject. Then, place your first roll disk in the tray and click *Continue*. Repeat this process for each roll disk.

If you are performing a network-based installation (also known as a *central* installation), then input the name of your roll server into the *Hostname of Roll Server* field and then click the *Download* button. This will query the roll server and all the rolls that the roll server has available will be displayed. Click the *selected* checkbox for each roll you will to install from the roll server.

When you have completed your roll selections, click the *Next* button to proceed to cluster input screens (e.g., IP address selection, root password setup, etc.).

Select Your Rolls

Local Rolls

Network-based Rolls

Hostname of Roll Server: central-x86-64.rocksclusters.org

In the "Hostname of Roll Server" field, insert the FQDN of your VM Server (the name of the physical machine that is hosting the VM frontend). Then click "Download".

From here, you want to follow the standard procedure for bringing up a frontend[2] starting at Step 8.

After the VM frontend installs, it will reboot. After it reboots, login and then we'll begin installing VM compute nodes.

3.3.5. Installing VM Compute Nodes

Login to the VM frontend (the virtual machine named "vi-1.rocksclusters.org" in the example picture at the top of this page), and execute:

```
# insert-ethers
```

Select "Compute" as the appliance type.

In another terminal session on vi-1.rocksclusters.org, we'll need to set up the environment to send commands to the Airboss on the physical frontend. We'll do this by putting the RSA private key that we created in section Creating an RSA Key Pair (e.g., private.key) on vi-1.rocksclusters.org.

Prior to sending commands to the Airboss, we need to establish a ssh tunnel between the virtual frontend (e.g., vi-1) and the physical frontend (e.g., espresso, where the Airboss runs). This tunnel is used to securely pass Airboss messages. On the virtual frontend (e.g., vi-1), execute:

```
# ssh -f -N -L 8677:localhost:8677 espresso.rocksclusters.org
```

Now we can securely send messages to the Airboss.

Now, we're ready to install compute nodes. But, there's a problem - when we first login to vi-1.rocksclusters.org, the only machine we know about is ourself (i.e., vi-1.rocksclusters.org). There are no other nodes in the virtual frontend's database. But the physical machine knows about the MAC addresses of the virtual compute nodes (e.g., hosted-vm-0-0-0 and hosted-vm-0-1-0) that are associated with this virtual cluster. The good news is, we can ask the Airboss on the physical frontend for a list of MAC addresses that are assigned to our virtual cluster:

```
# rocks list host macs vi-1.rocksclusters.org key=private.key
```

Which outputs:

```
MACS IN CLUSTER
36:77:6e:c0:00:02
36:77:6e:c0:00:00
36:77:6e:c0:00:03
```

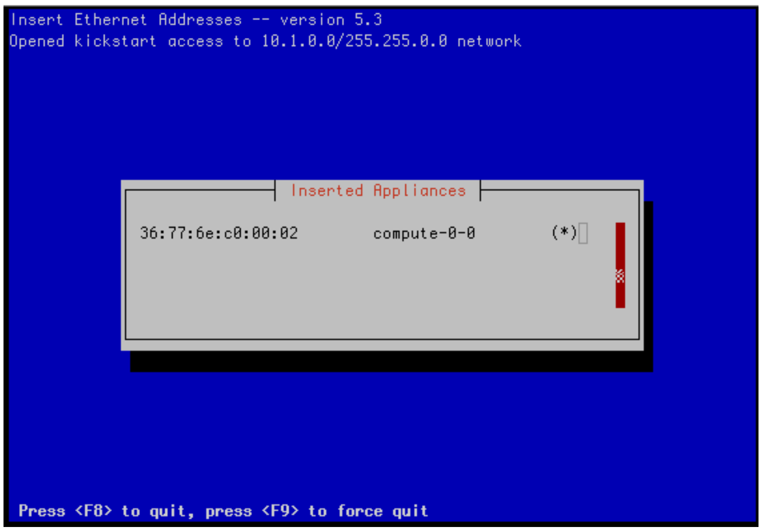
The MAC address 36:77:6e:c0:00:00 is ourself (the VM frontend) and the other two MACs (36:77:6e:c0:00:02 and 36:77:6e:c0:00:03) are the VM compute nodes that are associated with our VM frontend.

We can use the MAC address of the VM compute nodes to power up and install our compute nodes:

```
# rocks set host power 36:77:6e:c0:00:02 key=private.key action=install
```

The action of "install" ensures that the VM will be put into install mode, then it will be powered on.

Soon, you should see insert-ethers discover the VM compute node:



```
Insert Ethernet Addresses -- version 5.3
Opened kickstart access to 10.1.0.0/255.255.0.0 network

                    Inserted Appliances

  36:77:6e:c0:00:02          compute-0-0          (*)

Press <F8> to quit, press <F9> to force quit
```

After the virtual compute node is discovered by insert-ethers, we can open a console to the node by executing:

```
# rocks open host console compute-0-0 key=private.key
```

Lastly, to power off a virtual compute node (e.g., compute-0-0), execute:

```
# rocks set host power compute-0-0 key=private.key action=off
```

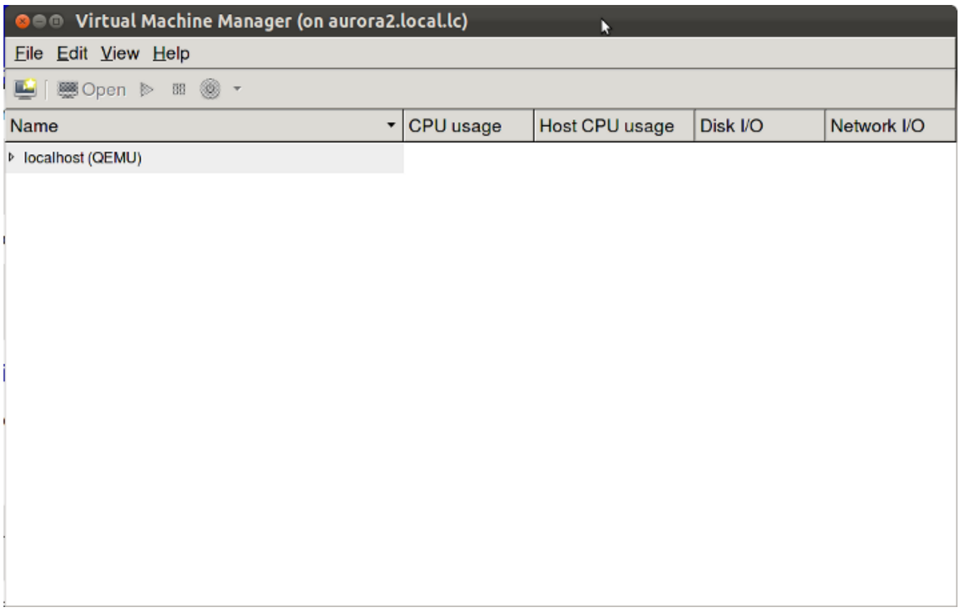
3.3.6. Using Virtual Machine Manager (Root Users Only)

The **virt-manager** program included in CentOS is a desktop user interface for managing virtual machines. This section describes how to use **virt-manager** functionality to control and monitor VMs on a Rocks cluster.

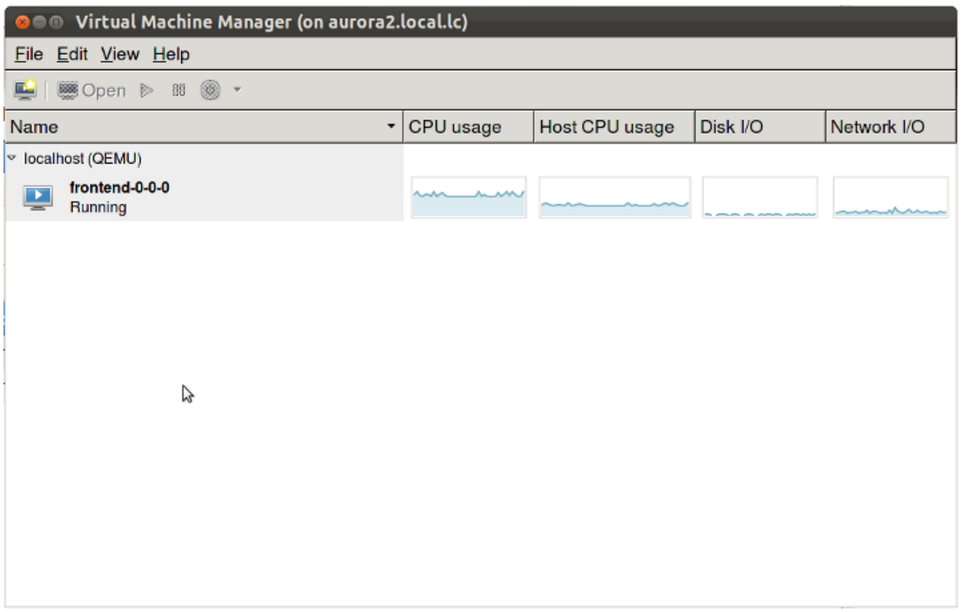
To interact with the VM frontend's console, on the physical frontend, you need to start "virt-manager":

```
# virt-manager
```

This will display a screen similar to:



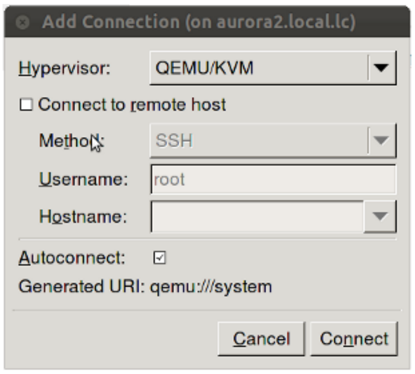
Double click on the "localhost" entry and then you'll see:



To bring the up the console for the VM frontend, double click on "frontend-0-0-0".

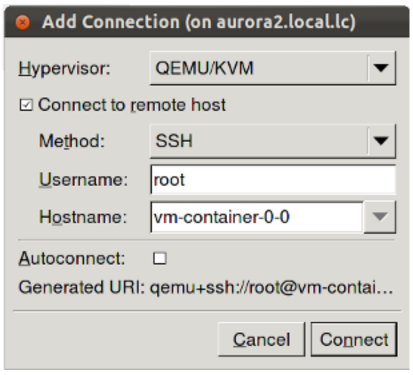
Now we'll describe how to connect to the console for the virtual compute node "compute-0-0". In the example configuration described at the top of this page, the VM "compute-0-0" is hosted on the physical machine named "vm-container-0-0" so we'll need to tell "virt-manager" to open a connection to "vm-container-0-0".

Inside "virt-manager", click on "File" then "Open connection...". This brings up a window that looks like:


Add Connection (on aurora2.local.lc)

Hypervisor: QEMU/KVM
☐ Connect to remote host
Method: SSH
Username: root
Hostname:
Autoconnect: ☑
Generated URI: qemu:///system

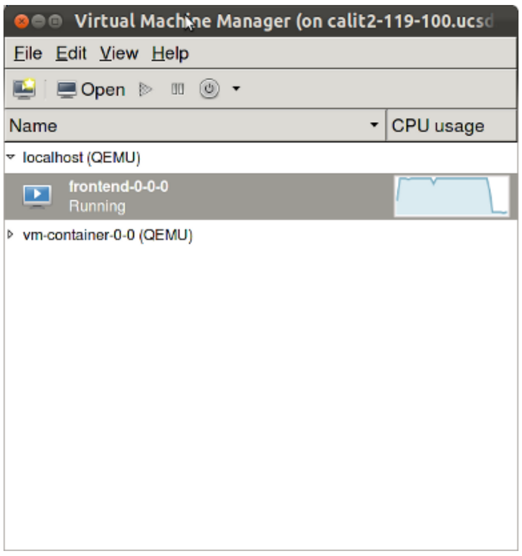
Now change the "Connection:" field to "Remote tunnel over SSH" and enter "vm-container-0-0" for the "Hostname:" field:


Add Connection (on aurora2.local.lc)

Hypervisor: QEMU/KVM
☑ Connect to remote host
Method: SSH
Username: root
Hostname: vm-container-0-0
Autoconnect: ☐
Generated URI: qemu+ssh://root@vm-contai...

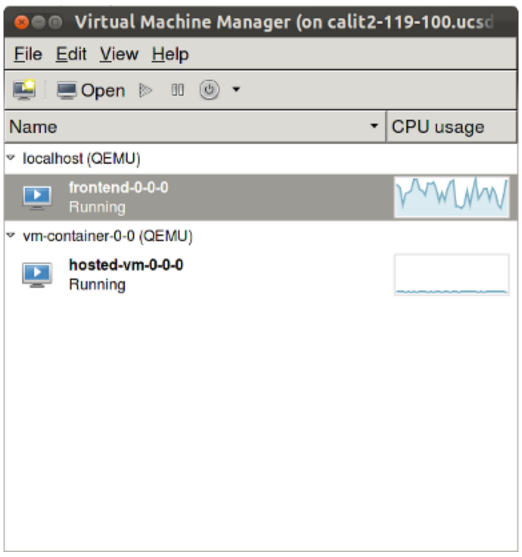
Then click "Connect".

In the "virt-manager" window, you should see something similar to:


Virtual Machine Manager (on calit2-119-100.ucsd

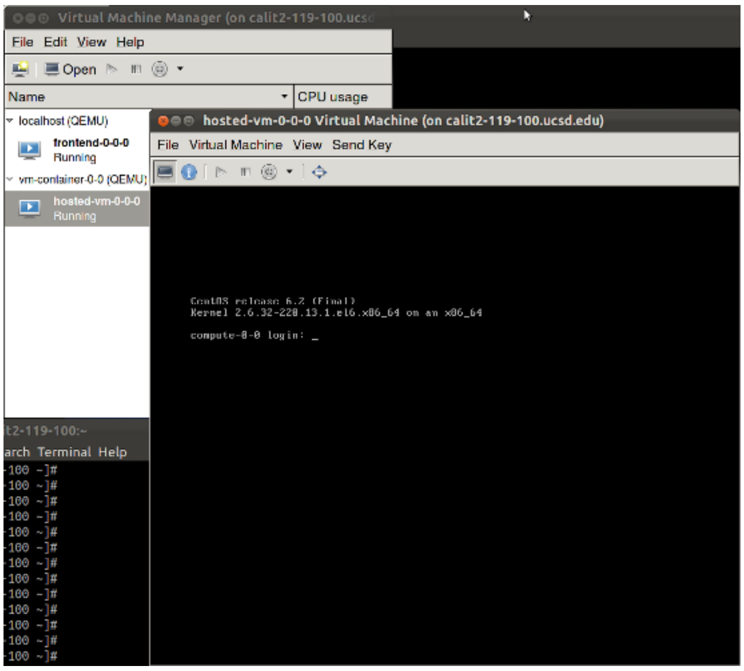
Name	CPU usage
localhost (QEMU)	
frontend-0-0-0 Running	
vm-container-0-0 (QEMU)	

Double click on "vm-container-0-0" and then you'll see:



Now to connect to the compute node's console, double click on "hosted-vm-0-0-0". Recall that from the perspective of the physical frontend (the VM Server), "hosted-vm-0-0-0" is the name for the VM "compute-0-0" (again, see the figure at the top of this page).

You should now see the console for compute-0-0:



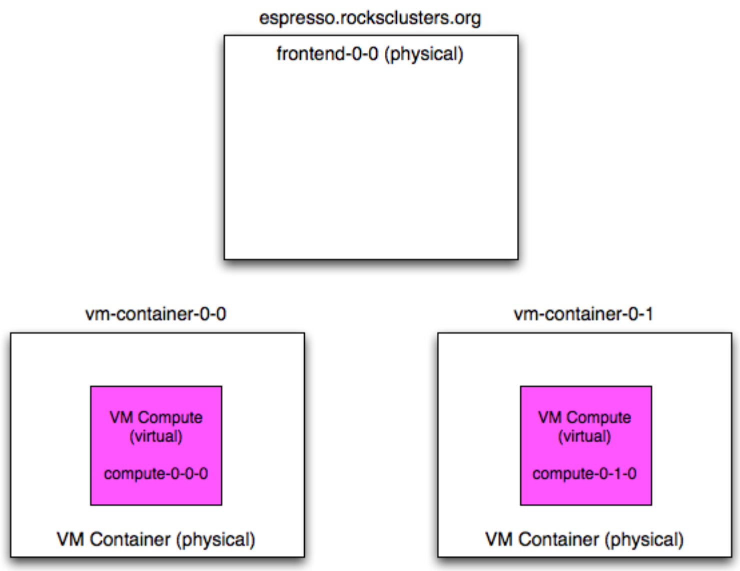
```
CentOS release 6.2 (Final)
Kernel 2.6.32-220.13.1.el6.x86_64 on an x86_64

compute-0-0 login: _
```

```
-100 ~]#
-100 ~]#
-100 ~]#
-100 ~]#
-100 ~]#
-100 ~]#
-100 ~]#
-100 ~]#
-100 ~]#
-100 ~]#
-100 ~]#
-100 ~]#
-100 ~]#
```

3.4. Physical Frontend with Virtual Compute Nodes

In this scenario, the frontend is a physical machine (not a VM) and the compute nodes are virtual machines.



In the above picture, "frontend-0-0" is a physical machine (with the public name of "espresso.rocksclusters.org". The physical machine "frontend-0-0" controls two VM compute nodes named "compute-0-0-0" and "compute-0-1-0". This means that "compute-0-0-0" and "compute-0-1-0" are configured by "frontend-0-0". This is opposed to the "virtual cluster scenario" (Installing Virtual Clusters), where the virtual frontend ("frontend-0-0-0") configured the VM compute nodes, and the physical machine that housed "frontend-0-0-0" only started and stopped the virtual compute nodes.

3.4.1. Adding, Installing and Booting VMs with a Physical Frontend

In the common case, you will execute three Rocks commands over the lifetime of your VMs: add (to add VM info to the database), start (to boot or install a VM) and stop (to shutdown an installed VM).

To add a VM to the system, you need to associate a VM with a physical machine (i.e., a VM container) and you need to assign an appliance type to the VM. Here's an example:

```
# rocks add host vm vm-container-0-0 membership="Compute"
```

The above command will output a message similar to:

```
added VM compute-0-0-0 on physical node vm-container-0-0
```

This tells us that, in the database, the compute VM named "compute-0-0-0" has been assigned to the physical machine "vm-container-0-0".

The next step is to install the VM.

VMs are installed with the `start`. Here's how to install the VM that was added above:

```
# rocks start host vm compute-0-0-0
```

After running the command above, you may see the following message:

```
libvir: KVM Daemon error : POST operation failed: (kvm.err "Error creating domain: Disk isn't accessibl
```

This is not a problem. The above means the file that will be used for the VMs disk space was not present when the VM was started. The **rocks start host vm** command eventually creates it and starts the VM. That is, while you may see the error message above, the VM is actually running and installing.

The above command will start the standard Rocks installation process for the VM named "compute-0-0-0". After the installation process initializes the network inside the VM, you can monitor the installation just like a physical machine installation by executing:

```
# rocks-console compute-0-0-0
```

After the installation completes, the VM will reboot. After the VM boots, you can interact with the VM just like any other physical machine.

3.5. Remotely Controlling VMs with Pilot

3.5.1. Pilot Overview

"Pilot" is a program that can be used on Windows, Mac OS X and Linux systems to remotely and securely control VMs that are housed on Rocks Clusters. Pilot communicates to the "Airboss" (see section The Airboss for more info) on a Rocks frontend. Pilot can power off, power on, install and connect to the console of VMs.

3.5.2. Installing Pilot

3.5.2.1. Installing Pilot on Mac OS X

Open a "terminal" session and execute:

```
$ curl -o pilot.py http://www.rocksclusters.org/ftp-site/pub/rocks/extra/pilot/pilot.py
$ chmod a+x pilot.py
$ curl -o TightVncViewer.jar http://www.rocksclusters.org/ftp-site/pub/rocks/extra/pilot/osx/Tigh
$ curl -o foundation-python-extras.dmg http://www.rocksclusters.org/ftp-site/pub/rocks/extra/pilo
```

Install the "foundation-python-extras.dmg" package:

```
$ open foundation-python-extras.dmg
```

This will open a window with a picture of an open box icon with the label "foundation-python-extras.pkg" under it. Double click on the open box icon.

This will open an installation window. Click on the default settings to properly install this package.

3.5.2.2. Installing Pilot on Windows

The following procedure has been tested on Windows 7, 32-bit.

First, you'll need to install "curl" on your machine. Download and unzip the following file:

```
http://am.net/lib/TOOLS/curl/curl-7.21.2-ssl-sspi-zlib-static-bin-w32.zip
```

Then, from the Rocks web site, download:

```
http://www.rocksclusters.org/ftp-site/pub/rocks/extra/pilot/windows/DLSupport.bat
```

Open a Windows Command Prompt and execute:

```
PATH=%PATH%;<directory for unzipped curl>
DLsupport
```

The above will download: Pilot, VCredist, Python v2.7, OpenSSL and M2Crypto.

Install VCredist, Python v2.7, OpenSSL and M2Crypto by executing:

```
vcredist_x86.exe
python-2.7.msi
Win32OpenSSL_Light-1_0_0a.exe
M2Crytpo-0.20.2.win32-py2.7.exe
```

If you don't already have an SSH client, you'll need to download one. We suggest Putty:

```
http://the.earth.li/~sgtatham/putty/latest/x86/putty-0.60-installer.exe
```

Most Windows PCs already have Java installed. Pilot has been tested with versions available from:

```
http://www.java.com/en/download/
```

3.5.3. Create a Public/Private Key Pair

Pilot uses a private/public key pair to authenticate its messages with the Airboss. If you haven't already created a key pair and associated the public key with the virtual frontend of the cluster you wish to remotely control, then login to the physical frontend that houses your virtual cluster and execute the procedure found here: Creating an RSA Key Pair.

Then copy the private key to the same directory on your machine where you have installed "pilot.py".

3.5.4. Open an SSH Tunnel from Your Machine to the Physical Frontend

Pilot sends messages to the Airboss on port 8677. We'll need to open a secure tunnel and forward port 8677 from your local machine to the Rocks frontend where the Airboss is running.

3.5.4.1. Mac OS X SSH Tunneling

Open a new terminal session and execute:

```
$ ssh -L 8677:localhost:8677 root@beopen.rocksclusters.org
```

Be sure to replace "beopen.rocksclusters.org" with the FQDN of your frontend.

3.5.4.2. Windows SSH Tunneling

If using PUTTY, a guide to setting up tunnels can be found here:

```
http://docs.cs.byu.edu/general/ssh_tunnels.html#use-putty-to-set-up-a-tunnel
```

3.5.5. Using Pilot for Remote VM Power Control

Before you get started, open a new terminal session on your machine. Pilot is a command-line utility.

To power on a VM, you must know the MAC address of the VM you wish to power on. Assuming you want to power on the VM named "frontend-0-0-0" on your physical frontend, execute:

```
# rocks list host interface frontend-0-0-0
SUBNET  IFACE MAC               IP              NETMASK       MODULE NAME                  VLAN O
private eth0  76:77:6e:40:00:00 10.1.255.251    255.255.0.0   ------ frontend-0-0-0        2    -
public  eth1  76:77:6e:40:00:01 137.110.119.118 255.255.255.0 ------ frontend-0-0-0-public 0    -
```

The VM frontend-0-0-0 has the MAC addresses of 76:77:6e:40:00:00 and 76:77:6e:40:00:01. You can use either with pilot.

To power on frontend-0-0-0, execute:

```
$ ./pilot.py set host power host=76:77:6e:40:00:00 key=private.key action=on
```

To power off frontend-0-0-0, execute:

```
$ ./pilot.py set host power host=76:77:6e:40:00:00 key=private.key action=off
```

To power on and to force frontend-0-0-0 to install, execute:

```
$ ./pilot.py set host power host=76:77:6e:40:00:00 key=private.key action=install
```

3.5.6. Using Pilot to Connect to a VM's Console

You can also connect to a VMs console with pilot by executing:

```
$ ./pilot.py open host console host=76:77:6e:40:00:00 key=private.key
```

When you connect to a VM's console, you will see two mouse pointers, that is, the mouse pointers are not in "sync". You will have to experiment to find which pointer is the true pointer for the console.

3.6. KVM Networking (Advanced)

In order to support KVM Virtual Machines it is necessary to have in place a particular network configuration on the hosting servers (which can be frontends or vm-containers). In particular bridges must be set up to provide Virtual Machine with network connectivity (KVM networking differ substantially from XEN networking). In this paragraph we will explain in detail what are the differences from a standard Rocks Cluster network configuration.

3.6.1. VM Network Bridging to Physical Devices

When a VM is bridged to the physical device, it must be assigned in the same subnet as the physical device with a compatible IP address (this case is exactly what is described in Physical Frontend with Virtual Compute Nodes).

For this reason every Ethernet interfaces is renamed with a 'p' (for physical) in front of its original name. The interface is also put into promiscuous mode and then attached to a bridge named with the original name of the interface. The IP address and netmask are finally configured on the bridge device.

For example if you have an eth0 which is supposed to be configured with IP address 1.2.3.4 and netmask 255.255.255.0 you will end up with the physical interface named peth0 in promiscuous mode attached to a bridge named eth0 with IP address 1.2.3.4 and netmask 255.255.255.0. The creation of the bridge device, the renaming of the physical interface and the assignment of the IP address is all managed by Rocks through the Red Hat init scripts. This means that **rocks sync host network** creates the proper configuration files in `/etc/sysconfig/network-scripts/`.

If you start a virtual compute on a VM Container, with the VLAN 0 (the virtual machine belongs to the same cluster as the physical frontend), you will see a virtual device belonging to the virtual machine attached to the eth0 bridge.

```
[root@vm-container-0-0 ~]# brctl show
bridge name bridge id  STP enabled interfaces
eth0  8000.00144f80de00 no  peth0
       vnet0
virbr0  8000.5254007953c7 yes  virbr0-nic
```

As you can see from the previous example bridge eth0 is connected to peth0 (the physical interface) and vnet0 (the virtual interface of the virtual machine). virbr0 is a default bridge started by libvirt which you can ignore.

3.6.2. Logical VLAN Devices

In this scenario, The guest (hosted-vm-0-0-0) and the host (vm-container-0-0) are not in the same logical network (this is the scenario described in Provisioning a Virtual Cluster).

The virtual machine hosted-vm-0-0-0 will be connected to the network through a Logical VLAN (for more information consult the Base Roll documentation on networking[3]). Logical VLAN interfaces don't need bridge device and they are dynamically loaded only when the virtual machine is started. They are called after the physical name interface simply adding the number of the VLAN (e.g. VLAN 2 on peth0 will be called peth0.2). Virtual machines are attached to the VLAN interface using a MacVTap[4] driver which does not require any bridging infrastructure. After you run a **rocks start host vm hosted-vm-0-0-0** if you go on the physical node you will see an interface called peth0.2 (the physical peth0 with the VLAN tag set to 2) and the macvtap0 which is the interface used by the virtual host.

3.7. Advanced Configuration

In this section we present several enhancements introduced with Rocks 6.2 which allow greater control of the virtualization functionalities.

3.7.1. Startup and Shutdown Plugins

The KVM roll allows users to define custom plugins which are invoked at each VM startup and shutdown. The plugins can be used to re-allocate the VM on a different physical container or to mount and unmount a storage system for the VM images, or to change some hardware feature of the VM before it is booted.

There are two different startup plugins which are triggered in different stages of the startup process. They both reside in `/opt/rocks/lib/python2.6/site-packages/rocks/commands/start/host/vm` and they must be called as indicated below.

`plugin_allocate.py`

> This script is invoked before the virtual machine is started and at this stage it is still possible to make modifications to the Rocks database. This means that it is still possible to modify the physical container of the virtual machine or its networking configuration.

`plugin_preboot.py`

> This script is invoked before the virtual machine is started but at this stage it is not possible to make modifications to the Rocks database. The virtual machine can not be relocated anymore to a different VM container. If this script returns a string it will be used as the libvirt XML startup. This means that it is still possible to make temporary changes to the hardware of the virtual machine.

There is one shutdown plugin which should be placed in: `/opt/rocks/lib/python2.6/site-packages/rocks/commands/stop/host/vm` The shutdown plugin must be called as indicated below.

`plugin_disallocate.py`

This script is invoked after the virtual machine is shutdown and it should release the resources allocated by the two previous scripts.

3.7.2. Fine tuning of virtual hardware

Several attributes can be used to customize the virtual hardware of the VM.

Since attribute values will be used inside an XML file during the kickstart generation they need to be properly escaped following XML escaping rules (only the characters >, < and & will be escaped). For this reason the attribute value might appear different when running **rocks list host attr**.

3.7.2.1. Defining Virtual CPU Types

Using the attribute `cpu_mode` it is now possible to configure a guest CPU to be as close to host CPU as possible. The attribute value can have two values (which are taken from Libvirt Documentation[5]):

- host-model: The host-model mode is essentially a shortcut to copying host CPU definition from capabilities XML into domain XML. Since the CPU definition is copied just before starting a domain, exactly the same XML can be used on different hosts while still providing the best guest CPU each host supports. Use this mode if you need to migrate virtual machine.
- host-passthrough: With this mode, the CPU visible to the guest should be exactly the same as the host CPU even in the aspects that libvirt does not understand. Though the downside of this mode is that the guest environment cannot be reproduced on different hardware. This is the default mode, if you don't need migration capabilities but just speed use this mode.

The attribute `cpu_mode` can also be used to specify a specific topology or model type for the cpu. If the value of `cpu_mode` attribute has a colon then the second part of the value (the one after the colon) will be used as a string to be inserted between the cpu tag. To better understand what tags can be used inside the cpu tag see the Libvirt Documentation[6]. So for example if the `cpu_mode` attribute value is:

```
exact: <model fallback='allow'>core2duo</model><vendor>Intel</vendor><topology sockets='1' cores=
```

Then the xml used for libvirt will be:

```
<cpu mode='exact'>
  <model fallback='allow'>core2duo</model><vendor>Intel</vendor><topology sockets='1' cores='2'/>
</cpu>
```

3.7.2.2. Pinning CPUs

Using the attribute `kvm_cpu_pinning` it is now possible to pin virtual CPUs to the physical CPUs. If the value of the attribute is `pin_all` each virtual cpu will be automatically pinned to the corresponding physical CPUs. This mode is good only if you have 1 VM for each physical machine, since the pinning will always start from physical core 0. Every other value used in this attribute will be dumped in the final libvirt xml as a child of the <domain> root tag.

For example the following command will pin virtual CPU 0 to physical CPU 4, virtual CPU 1 to physical CPU 5, virtual CPU 2 to physical CPU 6 and virtual CPU 3 to physical CPU 7 on VM called hosted-vm-0-2-0.

```
rocks set host attr hosted-vm-0-2-0 kvm_cpu_pinning value='<cputune>\
    <vcpupin vcpu="0" cpuset="4"/>\
    <vcpupin vcpu="1" cpuset="5"/>\
    <vcpupin vcpu="2" cpuset="6"/>\
    <vcpupin vcpu="3" cpuset="7"/>\
  </cputune>'
```

3.7.2.3. Defining Hardware Devices

Using the attributes called `kvm_device_%d` where the %d can be an integer going from 0 up, it is possible to add >devices< lines to a VM to fine tune the hardware devices which will be presented to the VM (for more info on the syntax which can be used please refer to the Libvirt Documentation[7])

For example the following command will assign the PCI slot 2 bus 6 and function 0 to the VM hosted-vm-0-2-0.

```
rocks set host attr hosted-vm-0-2-0 kvm_device_0 value='<hostdev mode='subsystem' type='pci' mana
  <source>\
    <address bus='0x06' slot='0x02' function='0x0'/>\
  </source>\
</hostdev>'
```

3.7.2.4. Mounting CDROM

Using the command **rocks set host vm cdrom** it is now possible attach CDROM to VM. The path specified in the cdrom attribute must exist on the physical container of the Virtual Machine. When a CDROM is attached the boot order of the machine is changed so that the CDROM will be first then it will try the network and then the hard disk. After the CDROM path has been changed with **rocks set host vm cdrom** the virtual machine has to be powered off and restarted with **rocks start host vm** in order to make effective the changes.

3.7.2.5. Host Autorestart

If a virtual machine has the attribute `kvm_autostart` defined with a value equal to true it will be automatically restart if the physical container is rebooted. If the physical container is properly shut down the Virtual Machine will be paused and saved to disk, and when the physical container restarts the VM will be properly restored automatically. If the physical container is unplugged from the power (hard shutdown) the virtual machine will crash like the physical container and it will be automatically restarted through a normal boot when the physical container is restarted.

3.7.2.6. Running VM on different appliances

Starting with Rocks 6.2 it is possible to run virtual machine on every type of node, for example you can have compute nodes which runs virtual machines. VM Containers are automatically enabled but if the user wants to enable a generic node to run virtual machines it must set the attribute `kvm` equal to true, and then re-install the node. After the re-installation the node will be able to host virtual machines.

3.8. Tutorial on Advanced Configuration

Since Rocks version 6.2 the kvm roll support plugins which are called at VMs startup and shutdown. Thanks to this users can customize several aspects of VM storage and networking. In the following example we show how to use a central NAS as a repository for VM images using Network Block Device.

The two example plugin scripts included in this tutorial are called each time a VM is started and stopped. Using this plugin script functionality it is even possible to relocate a VM to another physical host right before the virtual machine is booted.

This tutorial is only meant as an example for showing how to build more advanced system using the KVM roll. There are more advanced protocol which should be considered to build such a system (e.g. ISCSI protocol).

With this tutorial it is possible to host only 1 virtual machine for each physical host.

3.8.1. Preparing clients

This tutorial is using compute node to run the Virtual Machine (this is simply to demonstrate the functionality of the kvm attribute). To run NBD client is necessary to update the kernel (the CentOS kernel does not have the NBD module compiled in). To this extend run the following:

```
yumdownloader --enablerepo elrepo-kernel  kernel-lt.x86_64
cp kernel-lt-3.10.25-1.el6.elrepo.x86_64.rpm /export/rocks/install/contrib/6.1/x86_64/RPMS/
yumdownloader --enablerepo epel  nbd
cp nbd-2.9.20-7.el6.x86_64.rpm /export/rocks/install/contrib/6.1/x86_64/RPMS/
cd /export/rocks/install/site-profiles/6.1/nodes/
cp skeleton.xml extend-compute.xml
vi extend-compute.xml
```

and add the following lines to the package section:

```
<package>kernel-lt.x86_64</package>
<package>nbd</package>
```

rebuild the distribution set the attribute (so that compute node will be able to host VMs) and reinstall the compute nodes:

```
cd /export/rocks/install/
rocks create distro
rocks set appliance attr compute kvm true
rocks set appliance attr compute kvm_autostart true
rocks run host compute "/boot/kickstart/cluster-kickstart-pxe"
```

3.8.2. Creating the virtual cluster and allocating storage

We create a virtual cluster with two compute node and then we change the default disk setting of all the nodes. The virtual cluster internal names are rocks-216, vm-rocks-216-0 and vm-rocks-216-1.

```
 rocks add cluster 123.123.123.216 2 cluster-naming=true container-hosts="compute-0-0 compute-0-1
Getting Free VLAN -->
<-- Done
Creating Virtual Frontend on Physical Host rocks-152 -->
        created frontend VM named: rocks-216
<-- Done.
Creating 2 Virtual Cluster nodes  -->
        created compute VM named: vm-rocks-216-0
        created compute VM named: vm-rocks-216-1
<-- Done.
Syncing Network Configuration -->
<-- Done.

 rocks set host vm rocks-216 disk="phy:/dev/nbd0,vda,virtio"
 rocks set host vm vm-rocks-216-0 disk="phy:/dev/nbd0,vda,virtio"
 rocks set host vm vm-rocks-216-1 disk="phy:/dev/nbd0,vda,virtio"
```

For this tutorial we assume there is a nas-0-0 with preinstalled ZFS with a pool with enough capacity called /testpool. The script expects to find a preallocated zvol called /testpool/test-vm/hostname. To preallocate all the required space for the tree virtual machine we run the commands:

```
 ssh nas-0-0
 zfs create testpool/test-vm
 zfs create -V 35G testpool/test-vm/rocks-216
 zfs create -V 35G testpool/test-vm/vm-rocks-216-0
 zfs create -V 35G testpool/test-vm/vm-rocks-216-1
 exit
```

Rename the following files on your frontend:

```
 cd /opt/rocks/lib/python2.6/site-packages/rocks/commands/start/host/vm
 mv exampleplugin_allocate.py plugin_allocate.py
 cd /opt/rocks/lib/python2.6/site-packages/rocks/commands/stop/host/vm
 mv exampleplugin_disallocate.py plugin_disallocate.py
```

And put the following content into plugin_allocate.py:

```
#
#
import rocks.commands


class Plugin(rocks.commands.Plugin):


 nas_server = 'nas-0-0'
 device_base_path = '/dev/zvol/testpool/test-vm/'

 def provides(self):
  return 'allocate'

 def run(self, host):

  #
  # we need to find a free port on the server for this nbd-server
  #
  ports = []
  output = self.owner.command('run.host', [self.nas_server, """netstat -lnt""", "collate=true"])
  for line in output.split('\n'):
   tokens = line.split()
   if len(tokens) > 3 and tokens[0].startswith('tcp') \
     and len(tokens[3].split(':')) > 1:
    ports.append(tokens[3].split(':')[1])
  port = 0
  for i in range(2000,2200):
   if '%s' % i not in ports:
    port = i
    break
  if port == 0:
   self.owner.abort('unable to find a free port')
  port = str(port)

  if not host.vm_defs or not host.vm_defs.physNode:
   self.owner.abort("Unable to find the container for host %s" % host.name)


  #
  # let's start the server
  cmd = 'nbd-server ' + self.nas_server + '.local@' + port + ' ' + self.device_base_path + host.n
  self.owner.command('run.host', [self.nas_server, cmd, "collate=true"])
  # and the client
  cmd = 'modprobe nbd; nbd-client ' + self.nas_server + '.local ' + port + ' /dev/nbd0'
  self.owner.command('run.host', [host.vm_defs.physNode.name, cmd, "collate=true"])

  return
```

And put the following content into plugin_disallocate.py:

```
#

import rocks.commands

class Plugin(rocks.commands.Plugin):

 nas_server = 'nas-0-0'
 device_base_path = '/dev/zvol/testpool/test-vm/'

 def provides(self):
  return 'disallocate'

 def run(self, host):
  # here you can disallovate the resource used by your VM
  # in rocks DB

  # first let kill the clinet
  self.owner.command('run.host', [host.vm_defs.physNode.name, 'nbd-client -d /dev/nbd0'])
  # then the server
  self.owner.command('run.host', [self.nas_server, 'pkill -f "nbd-server.*' \
    + self.device_base_path + host.name + '"'])

  return
```

3.8.3. Using the Virtual Cluster

To start the cluster you can use the standard Rocks command

```
rocks start host vm rocks-216
```

Install the frontend and then run **insert-ethers** on the virtual frontend and turn on the compute nodes.

```
rocks start host vm vm-rocks-216-0
rocks start host vm vm-rocks-216-1
```

If the virtual compute node are shut down, the user can relocate them on a different physical host using the following command:

```
rocks set host vm vm-rocks-216-1 physnode=compute-0-0
```

When vm-rocks-216-1 will be restarted (with **rocks start host vm**) it will be automatically migrated on the new physical host compute-0-0

Notes

1. /roll-documentation/base/7.0/install-frontend.html
2. /roll-documentation/base/7.0/install-frontend.html
3. http://www.rocksclusters.org/roll-documentation/base/6.0/x1051.html#NETWORKING-LOGICAL-VLAN
4. http://virt.kernelnewbies.org/MacVTap
5. http://libvirt.org/formatdomain.html#elementsCPU
6. http://libvirt.org/formatdomain.html#elementsCPU
7. http://libvirt.org/formatdomain.html#elementsDevices

Chapter 4. Command Reference

4.1. add

4.1.1. add cluster

rocks add cluster {ip} {num-computes} [cluster-naming=*bool*] [container-hosts=*string*] [cpus-per-compute=*string*] [disk-per-compute=*string*] [disk-per-frontend=*string*] [fe-container=*string*] [fe-name=*string*] [ip=*string*] [mem-per-compute=*string*] [num-computes=*string*] [virt-type=*string*] [vlan=*string*]

Add a VM-based cluster to an existing physical cluster.

arguments

ip

The IP address for the virtual frontend.

num-computes

The number of compute nodes VMs to associate with the frontend.

parameters

[cluster-naming=*bool*]

If true it will name the compute nodes not based on the physical node they are allocated but based on the fe-name. If the frontend is called cluster the compute nodes will be named: vm-cluster-0 vm-cluster-1 vm-cluster-2 etc.

[container-hosts=*string*]

A list of VM container hosts that will be used to hold the VM compute nodes. This must be a space-separated list (e.g., container-hosts="vm-container-0-0 vm-container-0-1"). The default is to allocate the compute nodes in a round robin fashion across all the VM containers.

[cpus-per-compute=*string*]

The number of CPUs to allocate to each VM compute node. The default is 1.

[disk-per-compute=*string*]

The size of the disk (in gigabytes) to allocate to each VM compute node. The default is 36.

[disk-per-frontend=*string*]

The size of the disk (in gigabytes) to allocate to the VM frontend node. The default is 36.

[fe-container=*string*]

Hosting machine for virtual frontend. Defaults to the physical frontend

[fe-name=*string*]

name to for labeling the frontend. defaults to frontend-0-0-n, where n is assigned

[ip=*string*]

Can be used in place of the ip argument.

[mem-per-compute=*string*]

The amount of memory (in megabytes) to allocate to each VM compute node. The default is 1024.

[num-computes=*string*]

Can be used in place of the num-computes argument.

[virt-type=*string*]

Defines the virtualization type as either paravirtualized (para) or Hardware Virtualized (hvm). KVM supports only hvm if you try to specify something else it will abort.

[vlan=*string*]

The VLAN ID to assign to this cluster. All network communication between the nodes of the virtual cluster will be encapsulated within this VLAN. The default is the next free VLAN ID.

examples

# rocks add cluster 1.2.3.4 2

Create one frontend VM, assign it the IP address '1.2.3.4', and create 2 compute node VMs.

4.1.2. add host vm

rocks add host vm {host...} {membership} [cpus=*string*] [disk=*string*] [disk-size=*string*] [ip=*string*] [mac=*string*] [mem=*string*] [membership=*string*] [name=*string*] [num-macs=*string*] [slice=*string*] [subnet=*string*] [sync-config=*bool*] [virt-type=*string*] [vlan=*string*]

Add a VM specification to the database.

arguments

host

One or more physical host names.

membership

The membership to assign to the VM.

parameters

[cpus=*string*]

The number of CPUs to assign to this VM. The default is: 1.

[disk=*string*]

A disk specification for this VM. The fist part of the disk is a string used to specify the format of the disk on the domain 0: file is used for raw file format, qcow2 for the qcow2 file fomat, and phy for a physical device The second part of the disk string is a path to the location of the disk on the system up to the first comma (,). The third part is used to indicate the name under which the disk is exposed to the guest OS. The actual device name specified is not guaranteed to map to the device name in the guest OS. Treat it as a device ordering hint. The forth part indicate the type of disk device to emulate valid values are "virtio" (default), "ide", "scsi". The default is:
file:/<largest-partition-on-physical-node>/kvm/disks/<vm-name>.vda,vda,virtio

[disksize=*string*]

The amount of disk space in gigabytes to assign to the disk specification. The default is: 36.

[ip=*string*]

The IP address to assign to the VM. If no IP address is provided, then one will be automatically assigned.

[mac=*string*]

A MAC address to assign to this VM. If no MAC address is specified, the next free MAC address will be selected.

[mem=*string*]

The amount of memory in megabytes to assign to this VM. The default is: 2048.

[membership=*string*]

Can be used in place of the membership argument.

[name=*string*]

The name to assign to the VM (e.g., 'compute-0-0-0').

[num-macs=*string*]

The number of MAC addresses to automatically assign to this VM. The default is 1.

[slice=*string*]

The 'slice' id on the physical node. Each VM on a physical node has a unique slice number The default is the next available free slice number.

[subnet=*string*]

The subnet to associate to this VM. The default is: private.

[sync-config=*bool*]

Decides if 'rocks sync config' should be run after the VM is added. The default is: yes.

[virt-type=*string*]

Virtualization Type. With the kvm roll only hvm is a valid value.

[vlan=*string*]

The vlan ID to set for each interface. If you supply multiple MACs (e.g., 'num-macs' > 1), you can specify multiple vlan IDs by a comma separated list (e.g., vlan="3,4,5"). To not specify a vlan for a MAC, use the keyword 'none'. For example, if you want to specify a vlan ID for interface 1 and 3, but not interface 2, type: vlan="3,none,5". The default is to not assign a vlan ID.

examples

# rocks add host vm

Create a default VM.

# rocks add host vm mem=4096

Create a VM and allocate 4 GB of memory to it.

4.2. create

4.2.1. create host vm

rocks create host vm {host...}

Create a VM slice on a physical node. This command will configure a VM and install it. This can be used for the initial setup of a VM or to reconfigure an existing VM.

arguments

host

A list of one or more VM host names.

examples

# rocks create host vm compute-0-0-0

Create VM host compute-0-0-0.

4.3. dump

4.3.1. dump host vm

rocks dump host vm [host...]

Dump host VM information as Rocks commands.

arguments

[host]

Zero, one or more host names. If no host names are supplied, information for all hosts will be listed.

examples

$ rocks dump host vm compute-0-0-0

Dump VM info for compute-0-0-0.

$ rocks dump host vm

Dump VM info for all configured virtual machines.

related commands

add host vm

4.4. list

4.4.1. list cluster

rocks list cluster [cluster...] [status=*bool*]

Lists a cluster, that is, for each frontend, all nodes that are associated with that frontend are listed.

arguments

[cluster]

Zero, one or more frontend names. If no frontend names are supplied, information for all clusters will be listed.

parameters

[status=*bool*]

If true, then for each VM-based cluster node, output the VM's status (e.g., 'active', 'paused', etc.).

examples

$ rocks list cluster frontend-0-0

List the cluster associated with the frontend named 'frontend-0-0'.

$ rocks list cluster

List all clusters.

4.4.2. list host interface disablekvm

rocks list host interface disablekvm [host...]

Lists all the host interfaces with their respective disable_kvm flag

arguments

[host]

Zero, one or more host names. If no host names are supplied, information for all hosts will be listed.

examples

$ rocks list host interface disablekvm

List all the hosts with their corrispective disable KVM flag

$ rocks list host vm compute-0-0 compute-0-1

List the VM configuration for compute-0-0 and compute-0-1.

4.4.3. list host vm

rocks list host vm [host...] [showdisks=*bool*] [status=*bool*]

Lists the VM configuration for hosts.

arguments

[host]

Zero, one or more host names. If no host names are supplied, information for all hosts will be listed.

parameters

[showdisks=*bool*]

If true, then output VM disk configuration. The default is 'false'.

[status=*bool*]

If true, then output each VM's status (e.g., 'active', 'paused', etc.).

examples

$ rocks list host vm compute-0-0

List the VM configuration for compute-0-0.

$ rocks list host vm compute-0-0 compute-0-1

List the VM configuration for compute-0-0 and compute-0-1.

4.5. move

4.5.1. move host vm

rocks move host vm {host} {physhost} {file}

Move a VM from its current physical node to another.

arguments

host

The name of the VM host to move.

physhost

The name of the physical host in which to move the VM.

file

The name of the file that stores the running VM's state.

examples

# rocks move host vm compute-0-0-0 vm-container-1-0

Move VM host compute-0-0-0 to physical host vm-container-1-0.

4.6. pause

4.6.1. pause host vm

rocks pause host vm {host...}

Pauses a VM slice on a physical node.

arguments

host

A list of one or more VM host names.

examples

# rocks pause host vm compute-0-0-0

Pause VM host compute-0-0-0.

4.7. remove

4.7.1. remove cluster

rocks remove cluster [cluster...]

Remove a virtual cluster.

arguments

[cluster]

One or more virtual frontend names.

examples

# rocks rmeove cluster frontend-0-0-0

Remove the cluster associated with the frontend named ’frontend-0-0’.

4.7.2. remove host vm

rocks remove host vm {host...}

Remove only some information (regarding a virtual machine) from the database for the supplied hosts. This command should not be used to delete a virtual machine from the rocks DB (it is only for internal use). Use rocks remove host instead.

arguments

host

A list of one or more VM host names.

examples

# rocks remove host compute-0-0-0

To remove the host compute-0-0-0 from the database. Do not use rocks remove host vm (this command) to remove a host.

4.8. report

4.8.1. report host vm virt_type

rocks report host vm virt_type {host}

Output the type of virtualization used for a VM (with KVM it is always equal to hvm).

arguments

host

One VM host name (e.g., compute-0-0-0).

examples

$ rocks report host vm virt-type compute-0-0-0

Report the vitualization type used.

4.8.2. report vm nextmac

rocks report vm nextmac

Outputs the next free MAC address that can be used for a VM.

examples

$ rocks report vm nextmac

4.9. restore

4.9.1. restore host vm

rocks restore host vm {host...} [file=*string*]

Restore a VM on a physical node. This command restores a previously saved VM.

arguments

host

A list of one or more VM host names.

parameters

[file=*string*]

The file name the saved VM state is stored in. If you don't supply this parameter, then the default file name is: /<largest-partition-on-physical-host>/kvm/disks/<vm-name>.saved. For example, on a physical node with the default partitioning, the file that contains the state for VM compute-0-0-0 is: /state/partition1/kvm/disks/compute-0-0-0.saved

examples

# rocks restore host vm compute-0-0-0

Restore VM host compute-0-0-0.

4.10. resume

4.10.1. resume host vm

rocks resume host vm {host...}

Resume a paused VM slice on a physical node.

arguments

host

A list of one or more VM host names.

examples

# rocks resume host vm compute-0-0-0

Resume paused VM host compute-0-0-0.

4.11. save

4.11.1. save host vm

rocks save host vm {host...} {file}

Save a VM on a physical node. This command saves a currently running VM, then halts the VM. This saved state can be used to restart the VM with the command ’rocks restore host vm’.

arguments

host

A list of one or more VM host names.

file

The file name the saved VM state will be stored in. If you don’t supply this parameter, then the default file name will be: /<largest-partition-on-physical-host>/kvm/disks/<vm-name>.saved. For example, on a physical node with the default partitioning, the saved file for VM compute-0-0-0 will be named: /state/partition1/kvm/disks/compute-0-0-0.saved

examples

# rocks save host vm compute-0-0-0

Save VM host compute-0-0-0.

4.12. set

4.12.1. set cluster power

rocks set cluster power {host} [action=`string`] [delay=`string`] {key=`string`}

Turn the power on or off for each client host in a virtual cluster. This command will *not* affect a virtual frontend.

arguments

host

The host name of a virtual frontend.

parameters

[action=`string`]

The power setting. This must be one of 'on', 'off' or 'install'. The 'install' action will turn the power on and force the host to install.

[delay=`string`]

Sets the time (in seconds) to delay between each power command. Default is '1'.

key=`string`

A private key that will be used to authenticate the request. This should be a file name that contains the private key.

examples

$ rocks set cluster power frontend-0-0-0 action=on

Turn on the power for each client node that is associated with frontend-0-0-0.

4.12.2. set host interface disablekvm

rocks set host interface disablekvm {host} {iface} [disablekvm=`bool`] [disablekvm=`bool`] [iface=`string`]

Change the disablekvm interface flag. This flag is used to disable the automatic generation of KVM bridged configuration when starting a viratul machine. If a given interface has this flag true it will not appear on the host at the first restart. This functionality can be used in conjunction with the plugins of rocks report host vm config

arguments

host

One or more VM host names.

iface

Interface that should be updated. This may be a logical interface or the mac address of the interface.

parameters

[disablekvm=*bool*]

Can be used in place of the disablekvm argument

[disablekvm=*bool*]

If true it will disable kvm bridging

[iface=*string*]

Can be used in place of the iface argument.

examples

# rocks set host interface disablekvm compute-0-0-0 ib0 true

Disable the automatic generation of ib0 for compute-0-0-0 in KVM

4.12.3. set host vm

rocks set host vm
{host} [disk=*string*] [disksize=*string*] [mem=*string*] [physnode=*string*] [slice=*string*] [virt-type=*string*]

Change the VM configuration for a specific VM.

arguments

host

One or more VM host names.

parameters

[disk=*string*]

A VM disk specification. More than one disk can be supplied. Each disk specification must separated by a space. For more information on the disk format please look at the: rocks add host vm help

[disksize=*string*]

The size of the VM disk in gigabytes. If more than one disk is supplied the sizes should be separated by space.

[mem=*string*]

The amount of memory in megabytes to assign to this VM.

[physnode=*string*]

The physical machine this VM should run on.

[slice=*string*]

The slice ID for this VM.

[virt-type=*string*]

Set the virtualization type for this VM. This can be 'para' or 'hvm'.

examples

# rocks set host vm compute-0-0-0 mem=4096

Change the memory allocation for VM compute-0-0-0 to 4 GB.

4.12.4. set host vm cdrom

rocks set host vm cdrom {host} [cdrom=*string*]

Add a CDROM image to the virtual machine specified. The machine must be restarted to make this effective

arguments

host

One or more VM host names.

parameters

[cdrom=*string*]

The path to the ISO image to use as a CDROM of the physical device path (/dev/cdrom) if it is cdrom=none, it will remove the cdrom image from the current disks

examples

# rocks set host vm cdrom compute-0-0-0 cdrom=/root/kernel-6.1-0.x86_64.disk1.iso

Mount the ISO /root/kernel-6.1-0.x86_64.disk1.iso as a CDROM

# rocks set host vm cdrom compute-0-0-0 cdrom=none

Remove the CDROM from node compute-0-0-0

4.13. start

4.13.1. start host vm

rocks start host vm {host...}

Boots a VM slice on a physical node. If kvm_autostart == true make the VM start automatically when the physical container boot.

arguments

host

A list of one or more VM host names.

examples

# rocks start host vm compute-0-0-0

Start VM host compute-0-0-0.

# rocks start host vm compute-0-0-0

Start VM host compute-0-0-0.

4.13.2. start service airboss

rocks start service airboss [foreground=*boolean*]

Starts the VM Control service. This service validates commands from remote hosts and, if the command is accepted, the command is parsed and applied to VMs that are managed by this host.

parameters

[foreground=*boolean*]

If set to to 'yes', this service will stay in the foreground. Default is 'no'.

4.14. stop

4.14.1. stop host vm

rocks stop host vm {host...} [action=*string*] [terminate=*bool*]

Destroy a VM slice on a physical node.

arguments

host

A list of one or more VM host names.

parameters

[action=*string*]

poweroff will shut down the VM immediately (default) shutdown will send the ACPI shutdown signal to the guest OS reset will poweroff and poweron the VM reboot will send the ACPI reboot signal to the guest OS

[terminate=*bool*]

If this is true the command will run only the plugin since it will assume that the host is already down. This is used by to invoke the plugin when the machine is powered so external allocated resource can be freed (e.g. iSCSI connection) NB: if terminate is not equal to true the plugin will not be called. Basically the command either stops the VM or calls the plugins (this is to account for the fact that the libvirtd hooks are called both when the host is shutdown and when the host is forcefully stopped with rocks stop host vm)

examples

# rocks stop host vm compute-0-0-0

Stop VM host compute-0-0-0. This is equivalent to a ’hard power off’, (i.e., pulling the power cord from a node).

4.14.2. stop service airboss

rocks stop service airboss

4.15. sync

4.15.1. sync host vlan

rocks sync host vlan

Start the vlan interface needed by virtual machines on a vm-containers and on frontends. VLan used by virtual machines now are not anymore under Red Hat network manager control, but they are started automatically by this command. If invoked against a VM it will start the VM vlan. If invoked against a VM-container it will update the file /etc/libvirt/networking/vlan.conf

examples

# rocks sync host vlan compute-0-0-0

Start the vlan needed by the virtual host compute-0-0-0 If the vrtual host does not use any vlan the command does nothing

Appendix A. Rocks® Copyright

Appendix B. Third Party Copyrights and Licenses

This section enumerates the licenses from all the third party software components of this Roll. A "best effort" attempt has been made to insure the complete and current licenses are listed. In the case of errors or ommisions please contact the maintainer of this Roll. For more information on the licenses of any components please consult with the original author(s) or see the Rocks® GIT repository[1].

Notes

1. http://git.rocksclusters.org

Appendix C. Known issues

This is the list of known issue with the KVM roll:

1. Ethernet channel bonding does not work with the KVM roll (rocks add host bonded).
2. Nvidia MCP77 integrated Ethernet Controller does not handle properly vlan tagged packages, so virtual clusters will not work on this hardware.
3. We have found that when a virtual frontend and a virutal compute node run on the same physical node it is necessary to disable tx checksumming offload on the eth0 in order to properly install the compute node. If during PXE boot compute node fails with "Connection timed out" error message run the following command on the frontend:

   ```
   ethtool -K eth0 tx off
   ```

   To make this change persistent after a reboot add that line in /etc/rc.d/rc.local of your fronend.
4. If you have a Frontend node and a compute node connected directly with a cable (and no switch in between), you have to change the STP parameters of the private network bridge otherwise the installation of virtual cluster nodes will fail. To do this run:

   ```
    brctl setfd eth0 1.00
    brctl sethello eth0 0.5
   ```