Roll Development Basics

Philip Papadopoulos
Normal RedHat Distribution

Parallel Code / WebFarm / Grid / Computer Lab

Message Passing / Communication Layer

- Job Scheduling and Launching
- Cluster Software Management
- Cluster State Management / Monitoring

Linux Environment

- Linux Kernel

HPC Device Drivers (e.g., Interconnect and Storage drivers)

Linux Distribution

HPC Community Software

End User Applications
Distribution Based on Rolls

Parallel Code / WebFarm / Grid / Computer Lab

Message Passing / Communication Layer (MPI, MPD and Sockets)

Job Scheduling and Launching

Cluster Software Management

Cluster State Management / Monitoring

Base

Myrinet

PVFS

CentOS 4.0

Base Roll

Stock CentOS

HPC Roll

SGE Roll

App Roll

© 2008 UC Regents
Treasure Hunting

- The treasure you seek is a fully installed and configured cluster
- What are the things you’ll need
Map
Instruction set
Resources
Translate that to a Roll

- Graph file ⇔ Installation Map ⇔ Map
- Node Files ⇔ Configuration Data ⇔ Instruction set
- RPM/Binaries ⇔ Resources
Available Rolls for Rocks 5.0

- Rolls we provide
  - Core: Base, Kernel, Web Server, OS
  - Area51 - security analysis tools
  - Bio - bioinformatics tools
  - Condor
  - HPC - MPICH and cluster tools
  - Ganglia - cluster monitoring
  - PVFS2 - parallel file system
  - SGE
  - Viz
  - Java
Available Rolls for Rocks 5.0

- Rolls produced by academic community

  - PBS/Maui
    - HPC group at University of Tromso, Norway

  - APBS (Adaptive Poisson-Boltzmann Solver)
    - NBCR group, UCSD

  - Grid Roll
    - ThaiGrid, Thailand
Available Rolls for Rocks 5.0

- Rolls produced by commercial entities
  - Absoft, Cisco OFED, PGI, Intel, Moab
    - ClusterCorp
  - MX/GM - Myrinet
    - Myricom (4.3)
  - CUDA – GPU Programming
    - Nvidia (4.3)
Roll Contents

◆ RPMS
  ✦ Your software.
  ✦ Tasks:
    • Package bits into RPM

◆ Kickstart Graph
  ✦ Your configuration.
  ✦ Tasks:
    • Verify correct files exist after installation
    • Verify correct operation on frontend and compute nodes
    • Test, Test, Test
How do you configure NTP on compute nodes?

ntp-client.xml:

```xml
<post>

<!-- Configure NTP to use an external server -->

<file name="/etc/ntp.conf">
  server <var name="Kickstart_PrivateNTPHost"/>
  authenticate no
  driftfile /var/lib/ntp/drift
</file>

<!-- Force the clock to be set to the server upon reboot -->

/bin/mkdir -p /etc/ntp

<file name="/etc/ntp/step-tickers">
  <var name="Kickstart_PrivateNTPHost"/>
</file>

<!-- Force the clock to be set to the server right now -->

/usr/sbin/ntpdate <var name="Kickstart_PrivateNTPHost"/>
/sbin/hwclock --systohc

</post>
```
Kickstart File

- RedHat’s Kickstart: DNA of a node
  - Monolithic flat ASCII file
    - “Main”: disk partitioning, timezone
    - “Packages”: list of RPM names
    - “Post”: shell scripts for config
  - Limited macro language
  - Requires forking based on site information and node type.
Getting A Kickstart File
Kickstart File

- Rocks XML Kickstart
  - Decompose a kickstart file into nodes and a graph
    - Graph specifies OO framework
    - Each node specifies a service and its configuration
  - SQL Database to help site configuration
  - “Compile” flat kickstart file from a web cgi script
Kickstart Graph for Kgen

Preprocess (kpp)
Kickstart Graph with Roll

© 2007 UC Regents
Full Kickstart Graph
Kickstart XML Language

- Graph contains
  - **Nodes**
    - Rich language to help with configuration tasks
  - **Edges**
    - Simple. Defines node *MEMBERSHIP* in compiled kickstart files
  - **Order**
    - Simple syntax. Defines *POST SECTION ORDER* among nodes.
Example Roll: Sweetroll

- Will use a fictitious roll named “Sweetroll”

```xml
<?xml version="1.0" standalone="no"?>

<kickstart>
  <description>
  The sweet roll.
  </description>
</kickstart>
```
Kickstart Nodes

- Altering Default Nodes
  - Can *replace* or *extend* default nodes in Roll
    - Extend: concatenated to the end of a default node
    - Replace: overwrite default node
  - *Discouraged use: Reserved for end users*

- Extend by name: extend-[node].xml
  - sweetroll/nodes/extend-compute.xml

- Replace by name: replace-[node].xml
  - sweetroll/nodes/replace-compute.xml
Kickstart Nodes

- **Graph**
  - Nodes
    - Rich language to help with configuration tasks

- “Main” section
- “Package” section
- “Post” section
- “Installclass” section
  - Used to modify Anaconda
Nodes XML Tools: <var>

- Get Variables from Database
  - `<var name="Kickstart_PrivateAddress"/>`
  - `<var name="Node_Hostname"/>`

- Can grab any value from the *appGlobals* database table

10.1.1.1
compute-0-0
<var> values from app Globals

<table>
<thead>
<tr>
<th>ID</th>
<th>Membership</th>
<th>Service</th>
<th>Component</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
<td>Info</td>
<td>ClusterLatlong</td>
<td>N32.87 W117.22</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>Info</td>
<td>ClusterName</td>
<td>Onyx</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
<td>Info</td>
<td>CertificateState</td>
<td>California</td>
</tr>
<tr>
<td>34</td>
<td>0</td>
<td>Info</td>
<td>CertificateOrganization</td>
<td>Rocksclusters</td>
</tr>
<tr>
<td>37</td>
<td>0</td>
<td>Info</td>
<td>CertificateLocality</td>
<td>San Diego</td>
</tr>
<tr>
<td>44</td>
<td>0</td>
<td>Info</td>
<td>CertificateCountry</td>
<td>US</td>
</tr>
<tr>
<td>45</td>
<td>0</td>
<td>Info</td>
<td>ClusterURL</td>
<td><a href="http://onyx.rocksclusters.org/">http://onyx.rocksclusters.org/</a></td>
</tr>
<tr>
<td>50</td>
<td>0</td>
<td>Info</td>
<td>RocksRelease</td>
<td>Makalu</td>
</tr>
<tr>
<td>52</td>
<td>0</td>
<td>Info</td>
<td>RocksVersion</td>
<td>3.3.0</td>
</tr>
<tr>
<td>54</td>
<td>0</td>
<td>Info</td>
<td>ClusterContact</td>
<td><a href="mailto:admin@onyx.rocksclusters.org">admin@onyx.rocksclusters.org</a></td>
</tr>
<tr>
<td>58</td>
<td>0</td>
<td>Info</td>
<td>Born</td>
<td>2005-02-23 14:30:13</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Kickstart</td>
<td>PrivateKickstartBasedir</td>
<td>install</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Kickstart</td>
<td>PartsizeRoot</td>
<td>6000</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Kickstart</td>
<td>PublicAddress</td>
<td>198.202.88.74</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>Kickstart</td>
<td>PublicHostname</td>
<td>onyx.rocksclusters.org</td>
</tr>
</tbody>
</table>

◆ Combine “Service” and “Component”

✦ For example, Kickstart_PublicAddress

© 2007 UC Regents
Nodes XML Tools:  <var>

◆  <var> attributes
  
  ➤ name
    • Required. Format is “Service_Component”
    • Service and Component relate to column names in the app_global database table.
  
  ➤ val
    • Optional. Sets the value of this variable
      • <var name=“Info_ClusterName” val=“Seinfeld”/>
  
  ➤ ref
    • Optional. Set this variable equal to another
      • <var name=“Info_Weather” ref=“Info_Forecast”/>
Nodes XML Tools: <eval>

◆ Do processing on the frontend when the kickstart file is generated (by the CGI script):
  <eval shell="bash">
  
◆ To insert the Rocks release info in the kickstart file:

<eval shell="bash">
cat /etc/rocks-release
</eval>

Rocks release 4.2.1 (Cydonia)
Nodes XML Tools: <eval>

◆ <eval> attributes
  ➤ shell
    • Optional. The interpreter to use. Default “sh”
  ➤ mode
    • Optional. Value is quote or xml. Default of quote specifies for kpp to escape any XML characters in output.
    • XML mode allows you to generate other tags:
      • <eval shell="python” mode="xml”>
        • import time
        • now = time.time()
        • print "<var name=’Info_Now’ val=’%s’>” % now
      • </eval>
Nodes XML Tools: `<eval>`

- Inside `<eval>` variables are not accessed with `<var>`: use the environment instead.

```xml
<eval shell="python">
import os
print "My NTP time server is", os.environ['Kickstart_PublicNTPHost']
print "Got it?"
</eval>
```

My NTP time server is time.apple.com
Got it?
Nodes XML Tools `<include>`

- Auto-quote XML characters in a file
  - `<include file="foo.py"/>`

- Quotes and includes file
  ```
  sweetroll/include/foo.py
  ```

- foo.py (native) → foo.py (quoted xml):

```python
#!/usr/bin/python
import sys
def hi(s):
    print >> sys.stderr, s
```

```python
#!/usr/bin/python
import sys
def hi(s):
    print >&; sys.stderr, s
```
Nodes XML Tools: `<include>`

- `<include>` attributes
  - file
    - Required. The file to include (relative to “include/”) dir in roll src.
  - mode
    - Optional. Value is quote or xml. Default of quote specifies for kpp to escape any XML characters in file.
      - `<include file="my-favorite-things" mode="quote"/>`
Nodes XML Tools `<file>`

- Create a file on the system:
  - `<file name="/etc/hi-mom" mode="append">`
    - How are you today?
  - `</file>`

- Used extensively throughout Rocks post sections
  - Keeps track of alterations automatically via RCS.

```xml
<file name="/etc/hi" perms="444">
  How are you today?
  I am fine.
</file>
```

```plaintext
...RCS checkin commands...
cat > /etc/hi << 'EOF'
  How are you today?
  I am fine.
EOF
chmod 444 /etc/hi-mom
...RCS cleanup commands...
```
Nodes XML Tools: <file>

◆ <file> attributes
  ✦ name
    • Required. The full path of the file to write.
  ✦ mode
    • Optional. Value is “create” or “append”. Default is create.
  ✦ owner
    • Optional. Value is “user.group”, can be numbers or names.
      • <file name="/etc/hi" owner="daemon.root"/>
  ✦ perms
    • Optional. The permissions of the file. Can be any valid “chmod” string.
      • <file name="/etc/hi" perms="a+x"/>
Nodes XML Tools: <file>

◆ <file> attributes (continued)

  ✶ vars
    • Optional. Value is “literal” or “expanded”. In literal (default), no variables or backticks in file contents are processed. In expanded, they work normally.
      • <file name="/etc/hi" vars="expanded"/>
        • The current date is `date`
        • </file>

  ✶ expr
    • Optional. Specifies a command (run on the frontend) whose output is placed in the file.
      • <file name="/etc/hi" expr="/opt/rocks/dbreport hi"/>
Fancy <file>: nested tags

<file name="/etc/hi">

Rocks release:
<eval>
date +"%d-%b-%Y"
echo ""
cat /etc/rocks-release
</eval>

</file>

…RCS checkin commands...
cat > /etc/hi << ‘EOF’

Rocks release:
13-May-2005

Rocks release 4.2.1 (Cydonia)

EOF

…RCS cleanup commands…
Nodes Main

- Used to specify basic configuration:
  - timezone
  - mouse, keyboard types
  - install language
- Used more rarely than other tags
- Rocks main tags are usually a straight translation:

```xml
<main>
  <timezone>America/Mission_Beach</timezone>
</main>
```

```plaintext
... timezone America/Mission_Beach ...
rootpw --iscrypted sndk48shdlwis
mouse genericps/2
url --url http://10.1.1.1/install/rocks-dist/..
```
Nodes Main: Partitioning

- `<main>`
  - `<part> / --size 8000 --ondisk hda </part>`
  - `<part> swap --size 1000 --ondisk hda </part>`
  - `<part> /mydata --size 1 --grow --ondisk hda </part>`
- `</main>`

```
part / --size 8000 --ondisk hda
part swap --size 1000 --ondisk hda
part /mydata --size 1 --grow --ondisk hda
```
Nodes Packages

- `<package>java</package>`
  - Specifies an RPM package. Version is automatically determined: take the newest rpm on the system with the name ‘java’.

- `<package arch=”x86_64”>java</package>`
  - Only install this package on x86_64 architectures

- `<package arch=”i386,x86_64”>java</package>`

```
<package>newcastle</package>
<package>stone-pale</package>
<package>guinness</package>

%packages
newcastle
stone-pale
guinness
```
Nodes Packages

- RPMS are installed brute-force: no dependancy checking, always --force
Nodes Packages

- RPM name is a basename (not fullname of RPM)

  For example, RPM name of package below is ‘kernel’

```
# rpm -qip /home/install/rocks-dist/lan/i386/RedHat/RPMS/kernel-2.6.9-22.EL.i686.rpm
Name : kernel                      Relocations: (not relocatable)
Version : 2.6.9                     Vendor: CentOS
Release : 22.EL                     Build Date: Sun 09 Oct 2005 03:01:51 AM WET
Install Date: (not installed)       Build Host: louisa.home.local
Group : System Environment/Kernel   Source RPM: kernel-2.6.9-22.EL.src.rpm
Size : 25589794                     License: GPLv2
Signature : DSA/SHA1, Sun 09 Oct 2005 10:44:40 AM WET, Key ID a53d0bab443e1821
Packager : Johnny Hughes <johnny@centos.org>
Summary : the linux kernel (the core of the linux operating system)
Description :
The kernel package contains the Linux kernel (vmlinuz), the core of any
Linux operating system
```
Nodes Post

- `<post>` for *Post-Install* configuration scripts
- Configuration scripts in `<post>` section run after *all* RPMs have been installed.
  - Useful: you have all your software available
  - Scripts run in “target” environment: /etc in `<post>` will be /etc on the final installed system
- Scripts are always non-interactive
  - No Human is driving
Nodes Post

ntp-client.xml

```xml
<post>
/bin/mkdir -p /etc/ntp
/usr/sbin/ntpdate <var name="Kickstart_PrivateNTPHost"/>
/sbin/hwclock --systohc
</post>
```

```bash
%post
/bin/mkdir -p /etc/ntp
/usr/sbin/ntpd 10.1.1.1
/sbin/hwclock --systohc
```
Nodes Post Section

- Scripts have minimal $PATH (/bin, /usr/bin)
- Error reporting is minimal
  - Write to personal log file if you need debugging
- Not all services are up. Network is however.
  - Order tag is useful to place yourself favorably relative to other services
- Can have multiple <post> sections in a single node
Nodes XML Tools: <post>

◆ <post> attributes
  ➔ arch
    • Optional. Specifies which architectures to apply package.
  ➔ arg
    • Optional. Anaconda arguments to %post
      • --nochroot (rare): operate script in install environment, not target disk.
      • --interpreter: specifies script language

• <post arg="--nochroot --interpreter /usr/bin/python">
Post Example: PXE config

for an x86_64 machine:

```xml
<post arch="x86_64,i386">
  <file name="/tftpboot/pxe../default">
    default ks
    prompt 0
    label ks
    kernel vmlinuz
    append ks inird=initrd.img......
  </file>
</post>

<post arch="ia64">

  <!-- Itaniums do PXE differently -->
  ...
</post>
```

```bash
cat >> /root/install.log << 'EOF'
./nodes/pxe.xml: begin post section
EOF
mkdir -p /tftpboot/pxelinux/pxelinux.cfg
...
RCS...
cat > /tftpboot/pxe../default << EOF
default ks
prompt 0
...
EOF
..RCS...
```
<kickstart>
  <description>
  Enable SSH
  </description>

  <package>openssh/package>
  <package>openssh-clients</package>
  <package>openssh-server</package>
  <package>openssh-askpass</package>

  <post>

    <file name="/etc/ssh/ssh_config">
      Host *
      CheckHostIP       no
      ForwardX11         yes
      ForwardAgent       yes
      StrictHostKeyChecking       no
      UsePrivilegedPort no
      FallBackToRsh        no
      Protocol            1,2
    </file>

    chmod o+rx /root
    mkdir /root/.ssh
    chmod o+rx /root/.ssh

  </post>
</kickstart>
Graph Edges

- `<edge>`
- Specifies *membership* in a kickstart file
  - To make a kickstart file for a compute node:
    1. Take contents of “compute” xml node
    2. Follow all outgoing edges from “compute”
    3. Take all contents of child node
    4. Follow all its outgoing edges, etc, etc, etc

- Edges between nodes listed in a “graph” file
  - sweetroll/graphs/default/sweetroll.xml

- All graph files concatenated together
  - E.g., base.xml, hpc.xml, sweetroll.xml, etc. all concatenated
Graph Edges: <edge>

- **<edge> attributes**
  - **from**
    - Required. The name of a node at end of the edge
      - `<edge from="base" to="autofs"/>
  - **to**
    - Required. The name of a node at the head of an edge
  - **arch**
    - Optional. Which architecture should follow this edge. Default is all.
  - **gen**
    - Optional. Which generator should follow this edge. Default is “kgen”
Graph Edges

<edge from="security-server" to="central"/>

<edge from="client">
  <to arch="i386,x86_64">grub-client</to>
  <to>autofs-client</to>
  <to>installclass-client</to>
</edge>
Graph Ordering

- Added recently to give us control over when node `<post>` sections are run
  - `<order head="database">`
    - `<tail>database-schema</tail>`
  - `</order>`

- `database` node appears before `database-schema` in all kickstart files.

- Special HEAD and TAIL nodes represent “first” and “last” (post sections that you want to run first/last)
  - `<order head="installclass" tail="HEAD">` BEFORE HEAD
  - `<order head="TAIL" tail="postshell">` AFTER TAIL
Graph Ordering: `<order>`

- `<order>` attributes
  - head
    - Required. The name of a node whose `<post>` section will appear BEFORE in the kickstart file.
  - tail
    - Required. The name of a node whose `<post>` section will appear AFTER in the kickstart file.
      - `<order head="grub" tail="grub-server"/>
  - arch
    - Optional. Which architecture should follow this edge. Default is all.
  - gen
    - Optional. Which generator should follow this edge. Default is “kgen”
When Things Go Wrong

◆ Test your Kickstart Graph
  ✐ Check XML syntax: xmllint
  ✐ Make a kickstart file
    • Make kickstart file as a node will see it
      # rocks list host profile compute-0-0
When Things Go Wrong

◆ Test your Kickstart Graph
  ✐ Check XML syntax: xmllint
    • # cd sweetroll/nodes
    • # xmllint --noout sweetroll.xml

```xml
<?xml version="1.0" standalone="no"?>
<kickstart>
  <description>
  The sweet roll. This roll is just sweet!
  </description>
</kickstart>
```

```bash
# xmllint --noout sweetroll.xml
sweetroll.xml:7: parser error : Opening and ending tag mismatch: description line 6 and kickstart
</kickstart>
```
When Things Go Wrong

- Test your Kickstart Graph
  - Make a kickstart file

- First, install Sweetroll on the frontend “on-the-fly”:
  - # make roll;
  - # rocks add roll sweetroll-*.*.iso
  - # rocks enable sweetroll
  - # cd /home/install; rocks-dist dist
  - # kroll sweetroll > /tmp/install-sweetroll.sh
  - # sh /tmp/install-sweetroll.sh
When Things Go Wrong

- Test your Kickstart Graph
  - With Sweetroll XML in place:
    
    ```
    # dbreport kickstart compute-0-0 > /tmp/ks.cfg
    ```

  - Open `/tmp/ks.cfg` and look for the section:

    ```
    cat >> /root/install.log << 'EOF'
    ./.nodes/sweetroll.xml: begin post section
    ```

  - (We do this 10 times a day during release phase)
  - **Exactly the same as what a compute node actually sees during installation**
When Things Go Wrong

- Test your Kickstart Graph
  - Low level functionality test: kpp
    - Run the kickstart compilers by hand
      - For more difficult to diagnose problems
  - KPP is Kickstart Pre Processor: runs <eval>, <var>
  - KGEN is generator: turns XML into kickstart
    - # cd /home/install/rocks-dist/lan/x86_64/build
    - # kpp sweetroll
    - # kpp sweetroll | kgen
RPM Building
Building an RPM

- Generic RPMs are built with ‘spec’ file and ‘rpmbuild’
- It takes time to learn how to write a spec file
- Can use Rocks development source tree to create RPMs without having to make a spec file
Building an RPM

◆ Short story
  ➢ Go to /export/site-roll/rocks/src/roll on a Rocks Frontend
  ➢ Make a new roll from a ‘template’ roll
  ➢ Download the source tarball
  ➢ Update a description file (version.mk)
  ➢ Execute: make rpm
    • Assumes tarball adheres to ‘configure, make, make install’
Using Rocks Make Environment

- Rocks frontend has the tooling to build rolls
- cd /export/site-roll/rocks/src/roll/
- Let’s Make an RPM ---
- First, make a template for a new roll
  # ./bin/make-roll-dir.py --name valgrind --version 3.3.0
  # ls valgrind
  graphs Makefile nodes src version.mk

- valgrind/src/valgrind has what you need to make an rpm
src/valgrind – a working example

```bash
# cd valgrind/src/valgrind
# wget http://valgrind.org/downloads/valgrind-3.3.0.tar.bz2
# bunzip2 valgrind.*.bz2; gzip valgrind.tar
# rm *.spec.in
# edit version.mk so that
TARBALL_POSTFIX = tar.gz
# make rpm
# ls ../../RPMS/x86_64/valgrind-3.3.0-1.x86_64.rpm
../../RPMS/x86_64/valgrind-3.3.0-1.x86_64.rpm
../ ../../RPMS/x86_64/valgrind-3.3.0-1.x86_64.rpm

That’s it…. Works because valgrind is built using
“./configure; make; make install”
```
There is “magic” here

◆ We use RPM as a transport
  ✐ rpmbuild as the “package builder”
    • Needs an rpm spec file to drive it
    • We build a generic spec file automatically

◆ Standard RPM file tree needs the following directories to work properly

BUILD SOURCES SPECs
Step 1 of Magic – Create a Source File to go in SOURCES

Builds a tarball of your current directory called `<name>-<version>.tar.gz`

Copies this file into the SOURCES Directory
* contains this complete directory including the “real” software tarball
Step 2 of Magic

- Create a standard Redhat Spec file

Source: valgrind-3.3.0.tar.gz
Buildroot: `pwd`/valgrind.buildroot

%prep
  (unpack the tarball created in step 1)
%build
  (calls make build)
%install
  (calls make install)
ls of ../../BUILD ../../SOURCES

# ls ../../SOURCES/
valgrind-3.3.0.tar.gz ← Rocks-created tarball
# ls ../../BUILD/valgrind-3.3.0/ ← unpacked of above
_arch python.mk Rules-linux.mk
_distribution rocks-version.mk Rules.mk
Makefile Rules-install.mk Rules-rcfiles.mk
_os Rules-linux-centos.mk Rules-scripts.mk
valgrind-3.3.0.tar.gz version.mk
# ls ../../SPECS
valgrind.spec
Rpmbuild used with SPEC Files as input

- Just follows directions in the Makefile eg.

Build:

```
$(TAR) -zx $(NAME)-$(VERSION).$(TARBALL_POSTFIX)  # unpack actual source code

(  # actually configure and make
    cd $(NAME)-$(VERSION);
    ./configure --prefix=$(PKGROOT)/$(NAME);
    make;

    Equivalent logic for install
```
Okay … RPM Builds. Make the roll

```bash
# cd /export/site-roll/rocks/src/roll/valgrind
# make clean; make roll
# ls -l valgrind*iso
-rw-r--r-- 1 root root 16433152 May 14 22:15 valgrind-5.0-0.x86_64.disk1.iso
```

(note: have to edit graph file to define which appliance will get this rpm)
Connecting into the graph

# vi graphs/default/valgrind.xml (and add):

```xml
<edge from="base">
  <to>valgrind</to>
</edge>
```

◆ (now cheat, and shortcut a full roll rebuild)

# make profile; make reroll
Roll is complete

- Can use it as a roll to build frontends
- A straightforward test if you have a compute node

```
# rocks add roll valgrind-5.0-0.x86_64.disk1.iso
Copying valgrind to Rolls.....31340 blocks
#rocks enable roll valgrind
#(cd /home/install; rocks-dist dist)
.... rocks output...
# rocks list host profile compute-0-0 | grep valgrind
# ./nodes/valgrind.xml (valgrind)
roll-valgrind-usersguide
valgrind
```

© 2007 UC Regents