Basic Management and Customization

What We Will Cover

- Adding software to compute nodes
  - Quick and dirty method (read: not scalable!)
  - Rocks method
  - How to package code into an RPM
- Customizing compute node configuration
  - Using bash scripts in “<post>” sections
  - Configuring additional ethernet interfaces
  - Setting kernel boot parameters
- Flashing BIOS with PXE

Adding Software to Compute Nodes

Quick and Dirty

- On frontend, the directory /export/apps is shared on all compute nodes as:

```
/share/apps
```

- All files in /export/apps will be accessible on compute nodes:

```
# cd /export/apps
# touch myapp
# ssh compute-0-0
# cd /share/apps
# ls
myapp
```

Distribute Packages with the Rocks Installer

- If you have an RPM you'd like to install on all compute nodes, put the RPM in:

  `/home/install/contrib/5.0/`*`arch`*`/RPMS`

  - Where *arch* is i386 or x86_64

Distribute Packages with the Rocks Installer

- Create an XML file that ‘extends’ the compute XML file:

```
# cd /home/install/site-profiles/5.0/nodes
# cp skeleton.xml extend-compute.xml
```

Distribute Packages with the Rocks Installer

- Add your package name by changing the line:

```
<package> <!-- insert your package name here --> </package>
```

- To:

```
<package> your package  </package>
```

- Important: The package name must be the base name of the package

Get a Package's Base Name

- Assume you want to install the package:

vino-2.13.5-6.el5.x86_64.rpm

- Get the base name with "rpm –qip"

```
# rpm -qip vino-2.13.5-6.el5.x86_64.rpm
Name        : vino                          Relocations: (not relocatable)
Version     : 2.13.5                            Vendor: CentOS
Release     : 6.el5                          Build Date: Sun 07 Jan 2007 02:52:08 PM PST
Install Date: (not installed)                Build Host: builder3.centos.org
Group       : User Interface/Desktops        Source RPM: vino-2.13.5-6.el5.src.rpm
Size        : 1137432                           License: GPL
Signature   : DSA/SHA1, Tue 03 Apr 2007 05:27:50 PM PDT, Key ID a8a447dce8562897
URL         : http://www.gnome.org
Summary     : A remote desktop system for GNOME
Description :
Vino is a VNC server for GNOME. It allows remote users to
connect to a running GNOME session using VNC.
```

Adding Specific Architecture Packages

- On x86_64 systems, sometimes you want both the x86_64 and i386 versions of an RPM installed
  - "Native" package is installed by default
- Supply *.arch* in package tag:

```
<package>pkgbasename.x86_64</package>
<package>pkgbasename.i386</package>
```

Apply XML Node File to the Distribution

- Rebuild the distribution to apply extend-compute.xml

```
# cd /home/install
# rocks-dist dist
```

Reinstall to Apply the Packages to the Compute Nodes

- Reinstall one compute node:

```
# shoot-node compute-0-0
```

- After that node successfully boots and it has the packages you expect, then reinstall all the compute nodes:

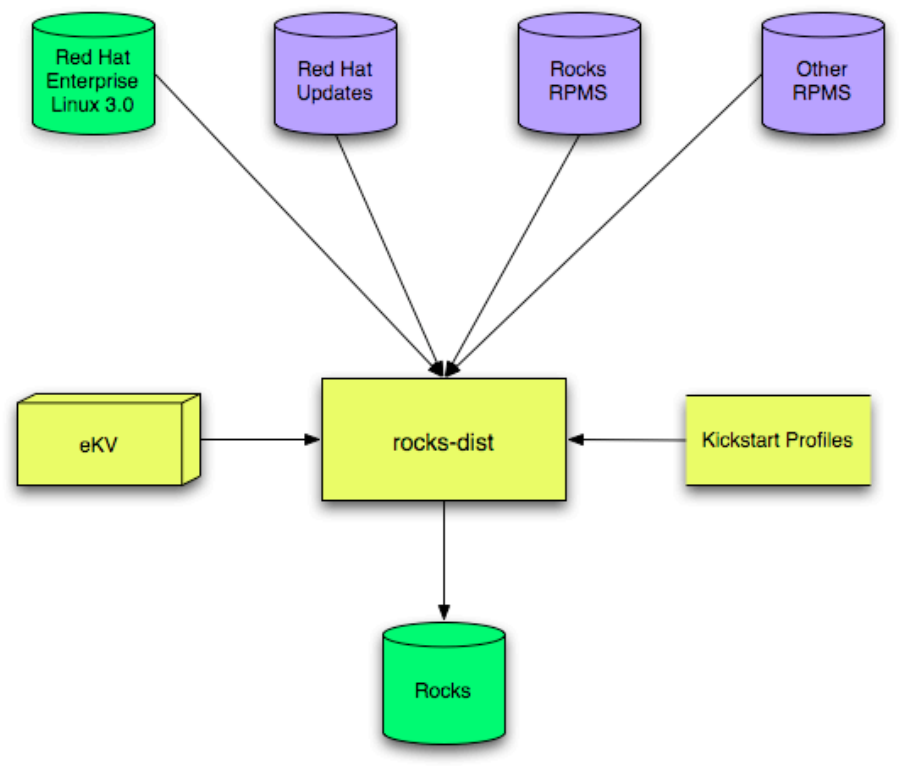
```
# rocks run host compute /boot/kickstart/cluster-kickstart
```

More on the Distro

- Rocks-dist looks for packages in:
  - "/home/install/rolls"
    - RedHat and Rocks packages
  - "/home/install/contrib"
    - Pre-built 3rd party packages
  - "/usr/src/redhat/RPMS"
    - RedHat default location for 'built' packages
    - But, when building packages in Rocks source tree, packages are not placed here
      - The packages are placed local to the roll source code

More on the Distro

- Any time you add a package to the distro, you must re-run “rocks-dist dist”
  - Rocks-dist binds all the discovered packages into a RedHat-compliant distribution



What If My Software Isn't in an RPM?

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 location on frontend that houses rocks development source tree
  - 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’

Package bonnie as an RPM

- Go to the Rocks roll development directory

```
# cd /export/site-roll/rocks/src/roll
```

- Side note: this is where the Restore Roll lives

```
# ls
bin  etc  restore  template
```

Create a Benchmark Roll

- Use the ‘template’ roll to populate a skeleton ‘benchmark’ roll

```
# cd /export/site-roll/rocks/src/roll/
# bin/make-roll-dir.py -n benchmark
```

- Create directory for bonnie

```
# cd benchmark/src
# mkdir bonnie++
```

Create a Bonnie RPM

- Get the source

```
# cd bonnie++
# wget http://www.coker.com.au/bonnie++/bonnie++-1.03a.tgz
```

Create a Bonnie RPM

- Create a version.mk file:

vi version.mk

```
NAME    = bonnie++
VERSION = 1.03a
RELEASE = 1
PKGROOT = /opt/$(NAME)
```

Create a Bonnie RPM

- Create a Makefile:

  # vi Makefile


```
REDHAT.ROOT     = $(CURDIR)/../../
ROCKSROOT       = ../../../../..
-include $(ROCKSROOT)/etc/Rules.mk
include Rules.mk

build:
        tar -zxvf $(NAME)-$(VERSION).tgz
        (                                       \
                cd $(NAME)-$(VERSION) ; \
                ./configure ;                   \
                make                            \
        )


install::
        mkdir -p $(ROOT)/$(PKGROOT)
        (                                                       \
                cd $(NAME)-$(VERSION) ;                         \
                make prefix=$(ROOT)/$(PKGROOT) install \
        )

clean::
        rm -f $(NAME).spec.in
```

Create a Bonnie RPM

- Build the RPM

```
# make rpm
```

- You see lots of output
  - The last line shows you where the resulting binary RPM is:

```
Wrote: /state/partition1/site-roll/rocks/src/roll/benchmark/RPMS/i386/bonnie++-1.03a-1.i386.rpm
```

Create a Bonnie RPM

- View the RPM contents

```
# rpm -qlp /state/partition1/site-roll/rocks/src/roll/benchmark/RPMS/i386/bonnie++-1.03a-1.i386.rpm
```

- Which outputs:

```
/
/opt
/opt/benchmark
/opt/benchmark/bonnie++
/opt/benchmark/bonnie++/bin
/opt/benchmark/bonnie++/bin/bon_csv2html
/opt/benchmark/bonnie++/bin/bon_csv2txt
/opt/benchmark/bonnie++/man
/opt/benchmark/bonnie++/man/man1
/opt/benchmark/bonnie++/man/man1/bon_csv2html.1
/opt/benchmark/bonnie++/man/man1/bon_csv2txt.1
/opt/benchmark/bonnie++/man/man8
/opt/benchmark/bonnie++/man/man8/bonnie++.8
/opt/benchmark/bonnie++/man/man8/zcav.8
/opt/benchmark/bonnie++/sbin
/opt/benchmark/bonnie++/sbin/bonnie++
/opt/benchmark/bonnie++/sbin/zcav
```

Copy the bonnie++ RPM so rocks-dist Can Find It

- All packages are found under '/home/install'
- Put bonnie++ RPM package in /home/install/contrib/5.0/<arch>/RPMS
  - Where <arch> is 'i386' or 'x86_64'

```
# cd /home/install/contrib/5.0/i386/RPMS
# cp /state/partition1/site-roll/rocks/src/roll/benchmark/RPMS/i386/bonnie++-1.03a-1.i386.rpm .
```

Extend the “Compute” XML Configuration File

- To add the package named “bonnie++”

```
$ cd /home/install/site-profiles/5.0/nodes
$ vi extend-compute.xml
```

- In ‘extend-compute.xml’, change the section:

```
<!-- <package> insert 1st package name here and uncomment the line</package> -->
```

- To:

```
<package>bonnie++</package>
```

Extend the “Compute” XML Configuration File

- Rebuild the distro
  - This copies ‘extend-compute.xml’ into /home/install/rocks-dist/…/build/nodes

```
# cd /home/install
# rocks-dist dist
```

- Test the changes
  - Generate a test kickstart file

```
# rocks list host profile compute-0-0 > /tmp/ks.cfg
```

  - You should see ‘bonnie++’ under the ‘%packages’ section

Extend the "Compute" XML Configuration File

- When you are satisfied with the changes, reinstall a compute node

```
# shoot-node compute-0-0
```

  - Or:

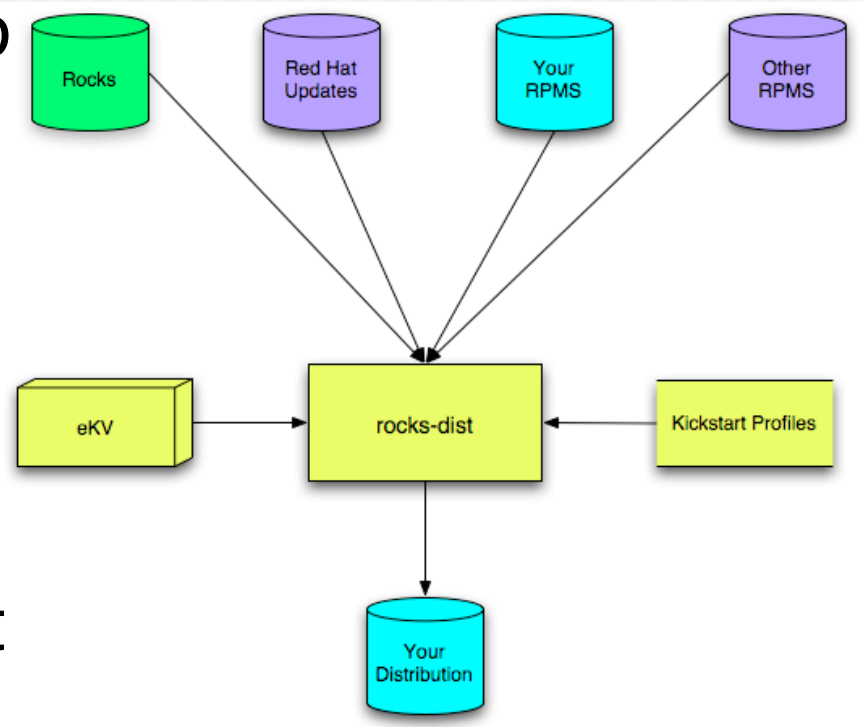
```
# ssh compute-0-0 /boot/kickstart/cluster-kickstart
```

- If you are satisfied with the compute node, shoot 'em all:

```
# rocks run host compute /boot/kickstart/cluster-kickstart
```

Your Distro - Extending Rocks

- You can use “rocks-dist” to build and distribute your own distribution
  - Merges RPMS
    - When two RPMS have the same basename, rocks-dist selects the one with the newest timestamp
- Final distribution looks just like Rocks
  - And, Rocks looks just like RedHat



Customizing Configuration of Compute Nodes

Extend an XML Node File

- Create an XML file that ‘extends’ the compute XML file:

```
# cd /home/install/site-profiles/5.0/nodes
# cp skeleton.xml extend-compute.xml
```

Code Your Configuration Changes

- Code your configuration changes in bash and put them in a “<post>” section:

```
<post>
        < !-- insert your scripts here -->
</post>
```

Apply XML Node File to the Distribution

- Rebuild the distribution to apply extend-compute.xml

```
# cd /home/install
# rocks-dist dist
```

Reinstall to Apply the Packages to the Compute Nodes

- Reinstall one compute node:

```
# shoot-node compute-0-0
```

- After that node successfully boots and it has the packages you expect, then reinstall all the compute nodes:

```
# rocks run host compute /boot/kickstart/cluster-kickstart
```

Configuring Additional Ethernet Interfaces

Configuring eth1

- If your compute nodes have more than 1 NIC, you can configure the other NICs with the Rocks command line
- Example:

```
# rocks list host interface compute-1-1
SUBNET  IFACE MAC               IP              NETMASK   GATEWAY MODULE NAME
private eth0  00:0e:0c:5d:7e:5e 10.255.255.251 255.0.0.0 ------- e1000  compute-1-1
------- eth1  00:30:1b:b2:ea:61 --------------- --------- ------- tg3    -----------
```

Configuring eth1

- We want to configure eth1 like:
  - IP: 192.168.1.1
  - Gateway: 192.168.1.254
  - Name: fast-1-1

```
# rocks set host interface ip compute-1-1 eth1 192.168.1.1
# rocks set host interface gateway compute-1-1 eth1 192.168.1.254
# rocks set host interface name compute-1-1 eth1 fast-1-1
# rocks set host interface subnet compute-1-1 eth1 public
```

Configuring eth1

- Check our work

```
# rocks list host interface compute-1-1
SUBNET  IFACE MAC               IP              NETMASK       GATEWAY         MODULE NAME
private eth0  00:0e:0c:5d:7e:5e 10.255.255.251 255.0.0.0     ----------      e1000  compute-1-1
public  eth1  00:30:1b:b2:ea:61 192.168.1.1    255.255.255.0 192.168.1.254 tg3    fast-1-1
```

- Reinstall to apply the changes:

```
# rocks run host compute-1-1 /boot/kickstart/cluster-kickstart
```

Configuring eth2

- Need to add a “network” to the database
  - Rocks automatically defines two networks:

```
# rocks list network
NETWORK  SUBNET        NETMASK
private: 10.0.0.0      255.0.0.0
public:  198.202.88.0 255.255.255.0
```

- Add a network for eth2

```
# rocks add network newnet 172.16.1.0 255.255.255.0

# rocks list network
NETWORK  SUBNET        NETMASK
private: 10.0.0.0      255.0.0.0
public:  198.202.88.0 255.255.255.0
newnet:  172.16.1.0    255.255.255.0
```

Configuring eth2

- Add network configuration like you did for eth1

```
# rocks set host interface ip compute-0-6 eth2 172.16.1.254
# rocks set host interface gateway compute-0-6 eth2 172.16.1.1
# rocks set host interface name compute-0-6 eth2 new-0-6
# rocks set host interface subnet compute-0-6 eth2 newnet
```

```
# rocks list host interface compute-0-6
SUBNET  IFACE MAC               IP              NETMASK       GATEWAY    MODULE NAME
private eth0  00:12:3f:20:e6:28 10.255.255.248 255.0.0.0     ---------- e1000  compute-0-6
------- eth1  00:12:3f:20:e6:29 -------------- ------------- ---------- e1000  -----------
newnet  eth2  00:01:02:03:04:05 172.16.1.254   255.255.255.0 172.16.1.1 e1000  new-0-6
```

Configuring eth2

- Check your work:

```
# rocks list host interface compute-0-6
SUBNET  IFACE MAC               IP              NETMASK       GATEWAY    MODULE NAME
private eth0  00:12:3f:20:e6:28 10.255.255.248 255.0.0.0     ---------- e1000  compute-0-6
------- eth1  00:12:3f:20:e6:29 --------------- ------------- ---------- e1000  -----------
newnet  eth2  00:01:02:03:04:05 172.16.1.254    255.255.255.0 172.16.1.1 e1000  new-0-6
```

- Look at the kickstart file:

```
# rocks list host profile compute-0-6 > /tmp/ks.cfg
```

- Inside /tmp/ks.cfg, you'll see:

```
cat > /etc/sysconfig/network-scripts/ifcfg-eth2 << 'EOF'
DEVICE=eth2
HWADDR=00:01:02:03:04:05
IPADDR=172.16.1.254
NETMASK=255.255.255.0
BOOTPROTO=static
GATEWAY=172.16.1.1
ONBOOT=yes
EOF
```

Setting Kernel Boot Parameters

Installation Boot Parameters

- Example, we'll add "ucsd=rocks" to compute-0-0 boot parameters
- The boot "action" of compute nodes is controlled by the Rocks command line:

```
# rocks list host pxeboot
HOST           ACTION
olympic:       ------
compute-0-0: os
```

  - "os" = boot the OS off local disk
  - "install" = on next boot, install

Installation Boot Parameters

- List all boot actions:

```
# rocks list host pxeaction compute-0-0
ACTION           COMMAND                ARGS
install          kernel vmlinuz         append ks initrd=initrd.img ramdisk_size=150000
                                                lang= devfs=nomount pxe kssendmac selinux=0 noipv6
install headless kernel vmlinuz         append ks initrd=initrd.img ramdisk_size=150000
                                                lang= devfs=nomount pxe kssendmac selinux=0 noipv6 headless vnc
memtest          kernel memtest         ------------------------------------------------------------------
os               localboot 0            ------------------------------------------------------------------
pxeflash         kernel memdisk bigraw  append initrd=pxeflash.img keeppxe
rescue           kernel vmlinuz         append ks initrd=initrd.img ramdisk_size=150000
                                                lang= devfs=nomount pxe kssendmac selinux=0 noipv6 rescue
```

Installation Boot Parameters

- Change boot action:

```
# rocks set host pxeboot compute-0-0 action="install"
```

- Check our work

```
# rocks list host pxeboot
HOST          ACTION
olympic:      -------
compute-0-0: install
```

Add a New PXE Action

- Add global action:

```
# rocks add host pxeaction action="install ucsd" command="kernel vmlinuz" \
  args="append ks initrd=initrd.img ramdisk_size=150000 lang= devfs=nomount \
  pxe kssendmac selinux=0 noipv6 ucsd=rocks"
```

- Check our work

```
# rocks list host pxeaction compute-0-0
ACTION           COMMAND                 ARGS
install          kernel vmlinuz          append ks initrd=initrd.img ramdisk_size=150000
                                         lang= devfs=nomount pxe kssendmac selinux=0 noipv6
install headless kernel vmlinuz          append ks initrd=initrd.img ramdisk_size=150000
                                         lang= devfs=nomount pxe kssendmac selinux=0 noipv6 headless vnc
install ucsd     kernel vmlinuz          append ks initrd=initrd.img ramdisk_size=150000
                                         lang= devfs=nomount pxe kssendmac selinux=0 noipv6 ucsd=rocks
memtest          kernel memtest          ------------------------------------------------------------
os               localboot 0             ------------------------------------------------------------
pxeflash         kernel memdisk bigraw   append initrd=pxeflash.img keeppxe
rescue           kernel vmlinuz          append ks initrd=initrd.img ramdisk_size=150000
                                         lang= devfs=nomount pxe kssendmac selinux=0 noipv6 rescue
```

Add a New PXE Action

- Add compute-0-0 only action:

```
# rocks add host pxeaction compute-0-0 action="install ucsd" \
  command="kernel vmlinuz" \
  args="append ks initrd=initrd.img ramdisk_size=150000 lang= devfs=nomount \
  pxe kssendmac selinux=0 noipv6 ucsd=rocks"
```

- Override global action

```
# rocks add host pxeaction compute-0-0 action="install" \
  command="kernel vmlinuz" \
  args="append ks initrd=initrd.img ramdisk_size=150000 lang= devfs=nomount \
  pxe kssendmac selinux=0 noipv6 ucsd=rocks"
```

Running Boot Parameters

- Get the current boot flags

```
# rocks report host bootflags
rocks-168: dom0_mem=1024M
compute-0-0: dom0_mem=1024M
```

- Add a boot flag

```
# rocks set host bootflags compute-0-0 flags="dom0_mem=1024M ucsd=rocks”
```

- Check

```
# rocks report host bootflags
rocks-168: dom0_mem=1024M
compute-0-0: dom0_mem=1024M ucsd=rocks
```

Running Boot Parameters

- Reinstall to apply boot flags
- After the node installs, check

```
# cat /proc/cmdline
ro root=LABEL=/ dom0_mem=1024M ucsd=rocks
```

Flashing BIOS with PXE

No More CDs or Floppies!

- Download BIOS file
  - Put in:
    - /opt/pxeflash/addon
- In /opt/pxeflash, execute:
  - make build
  - make install
- Set boot action

```
# rocks set host pxeboot compute-0-0 action=pxeflash
```

Boot and Flash

- PXE boot the compute node
  - You'll get a DOS prompt
- On frontend, reset boot action

```
# rocks set host pxeboot compute-0-0 action=os
```

- Execute the flash program
- Reboot the compute node
- Done!

The RedHat Installer

Anaconda: RedHat's Installer

- Open-source python-based installer
- Developed by RedHat
- (Somewhat) object-oriented
  - We extend when we can and insert "shims" when we can't

Anaconda: RedHat's Installer

- Key tasks:
  - Probe hardware
  - Ask users for site-specific values
    - E.g., IP addresses and passwords
  - Insert network and storage drivers
    - For network-based installations and to write packages down onto local disk
  - Install packages
    - RPMs
  - Configure services
    - Via shell scripts

Scripted Installation

- Anaconda achieves “lights-out” installation via kickstart mechanism
- It reads a “kickstart file”
  - Description of how to install a node
- One file composed of three key sections:
  - Main: general parameters
  - Packages: list of RPMs to install
  - Post: scripts to configure services

Kickstart File

- Main section

```
rootpw --iscrypted loijgoij5478fj2i9a
zerombr yes
bootloader --location=mbr
lang en_US
langsupport --default en_US
keyboard us
mouse genericps/2
install
reboot
timezone --utc America/Los_Angeles
part
```

Kickstart File

- Packages section

```
%packages --ignoredeps --ignoremissing
@Base
PyXML
atlas
autofs
bc
chkrootkit
contrib-pexpect
contrib-pvfs-config
contrib-python-openssl
```

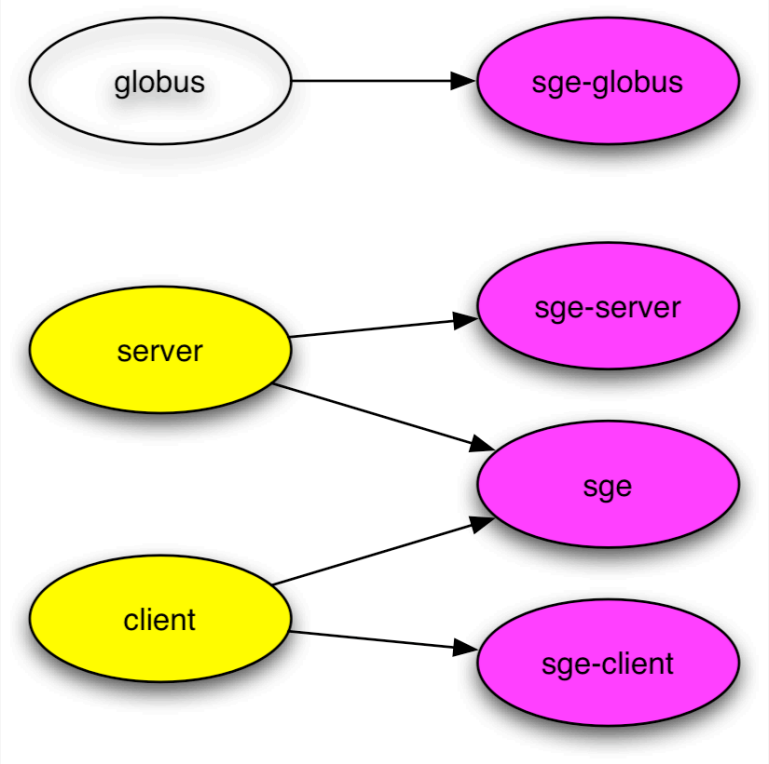
Kickstart File

- Post section

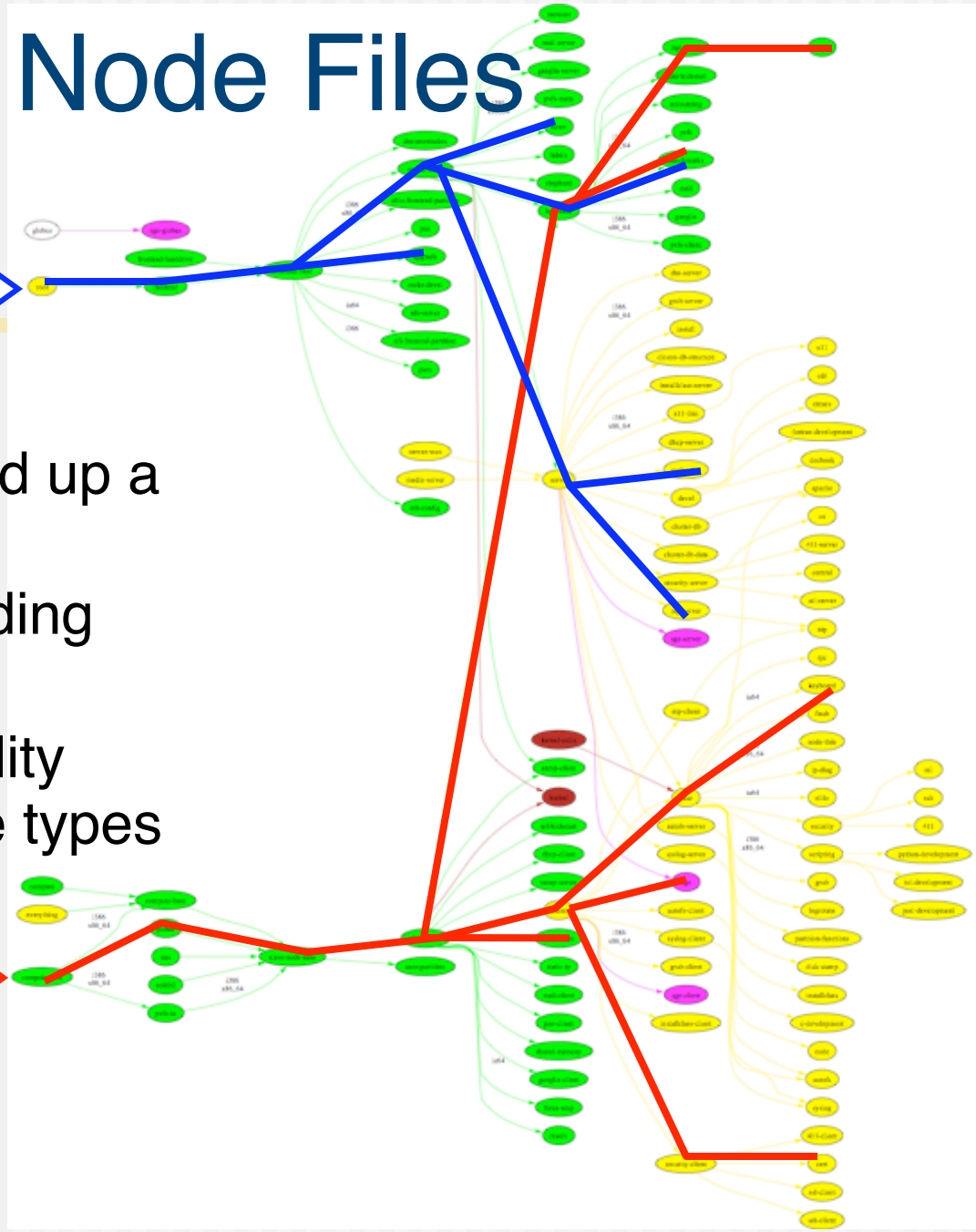
```
%post

cat > /etc/motd << 'EOF'
Rocks Compute Node
EOF
```

Use Graph Structure to Dissect Distribution

- Use ‘nodes’ and ‘edges’ to build a customized kickstart file
- Nodes contain portion of kickstart file
  - Can have a ‘main’, ‘package’ and ‘post’ section in node file
- Edges used to coalesce node files into one kickstart file



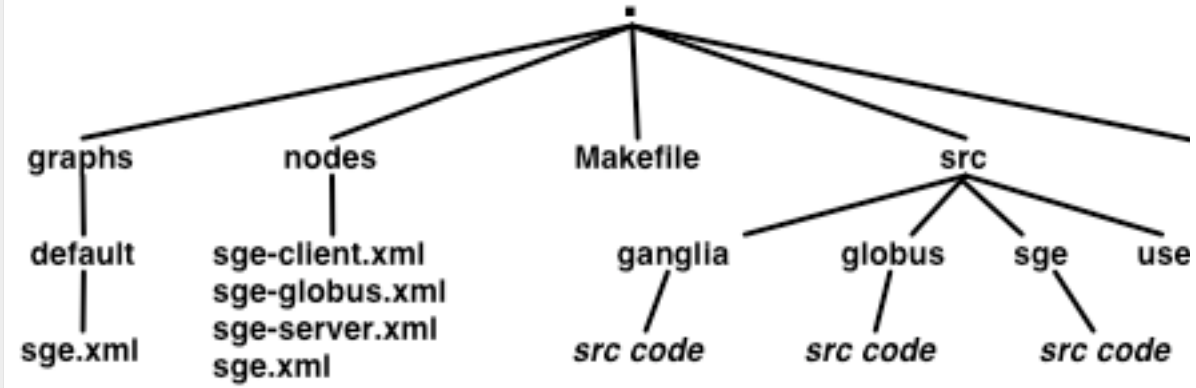
Coalescing Node Files

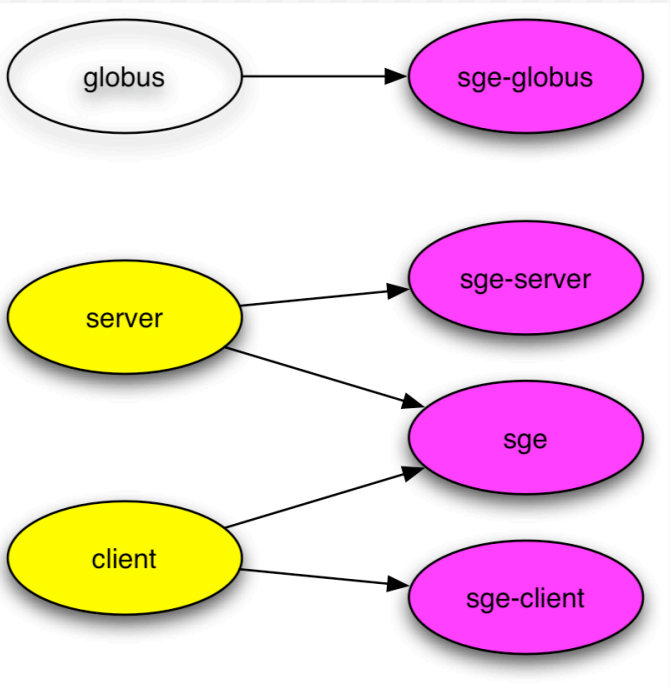


- Traverse a graph to build up a kickstart file
- Makes kickstart file building flexible
- Easy to share functionality between disparate node types

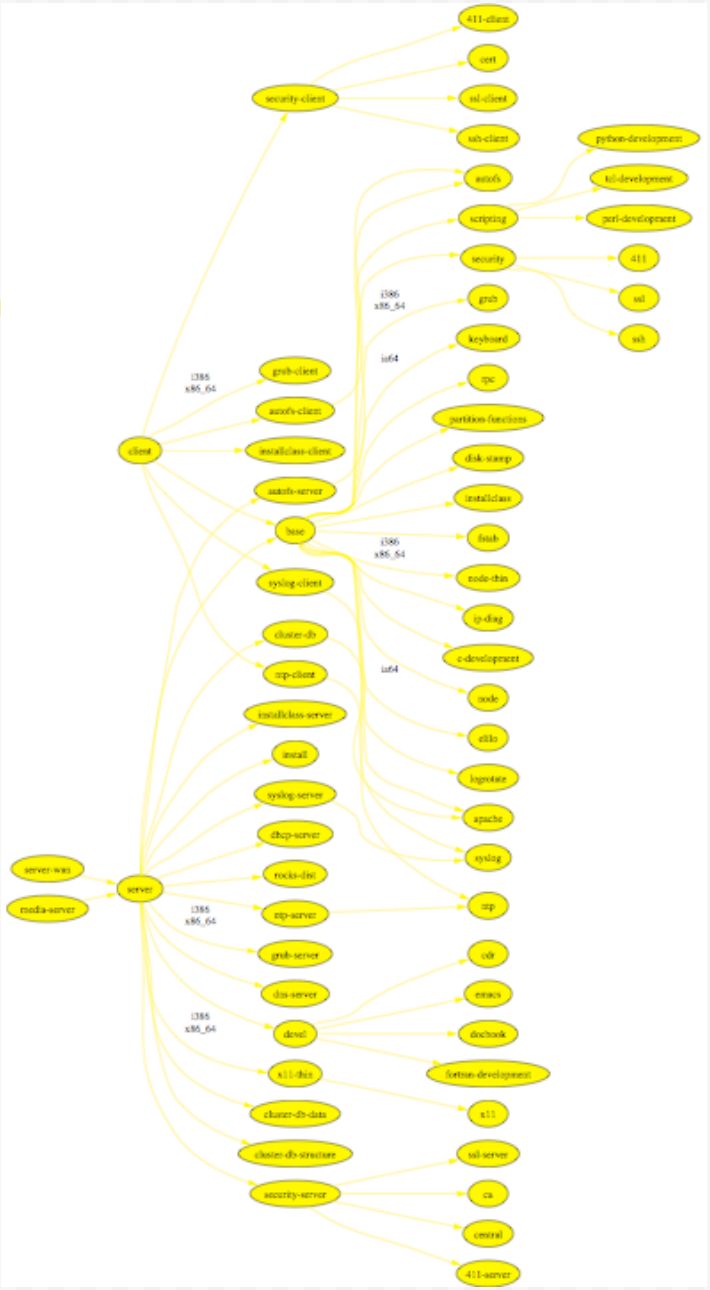
Why We Use A Graph

- A graph makes it easy to ‘splice’ in new nodes
- Each Roll contains its own nodes and splices them into the system graph file

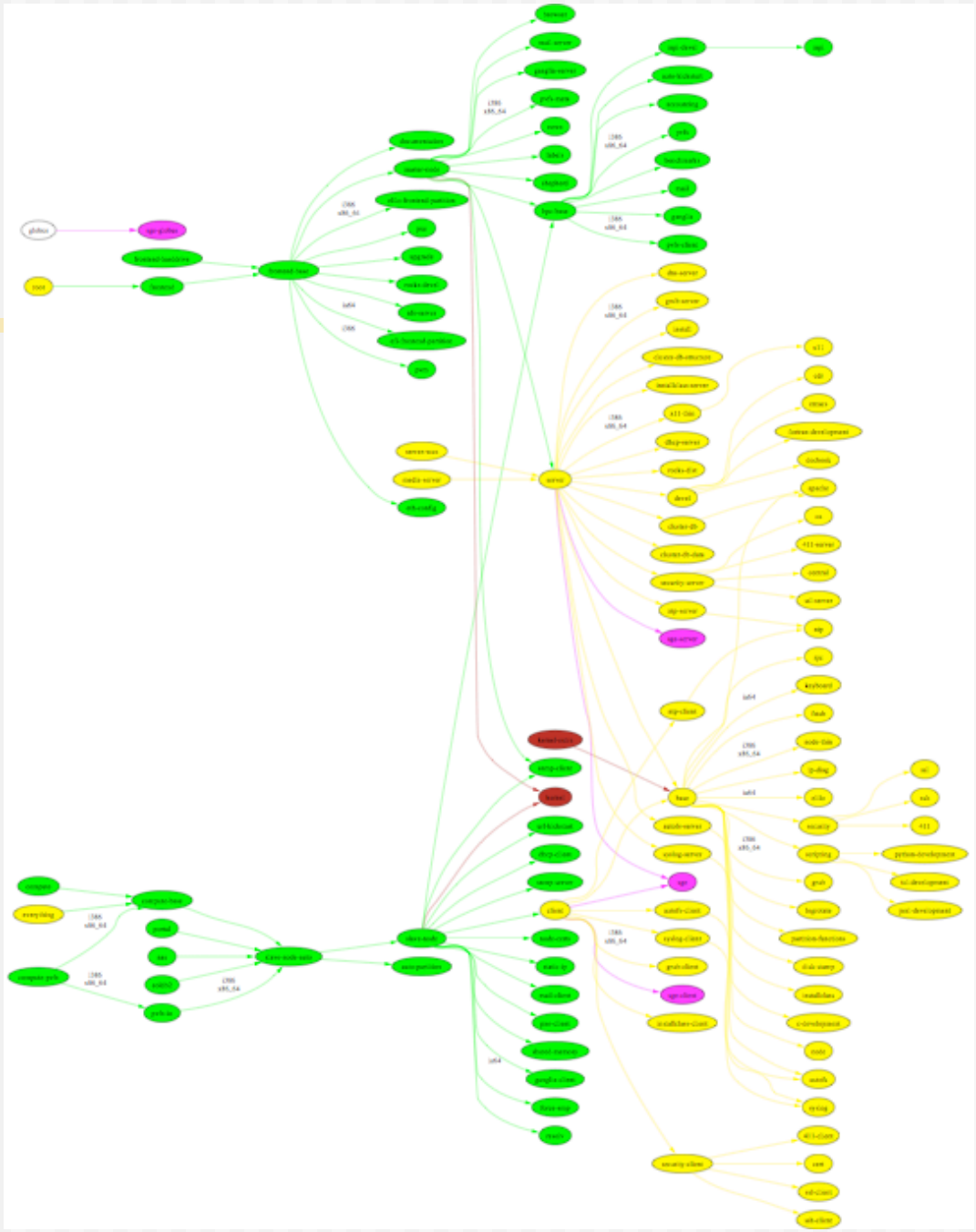




Install Rocks Base Graph

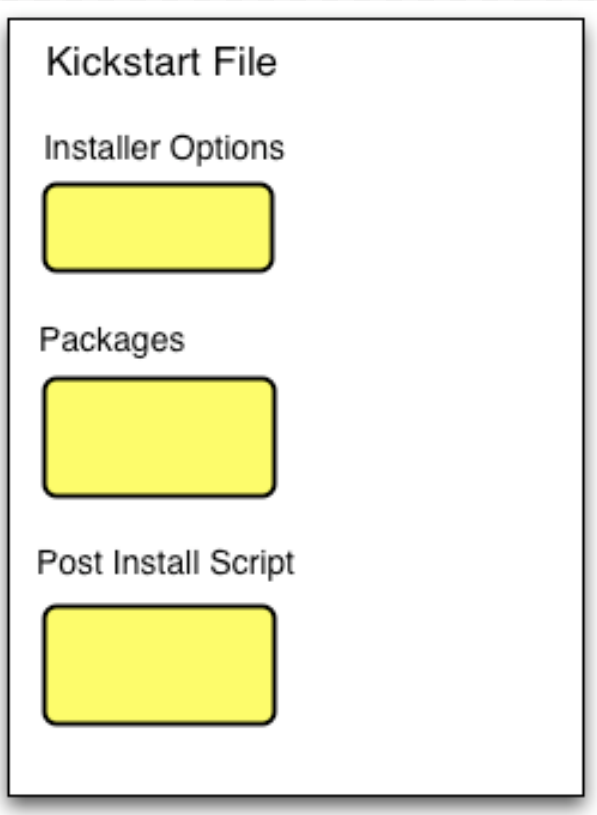


Base + All Rolls

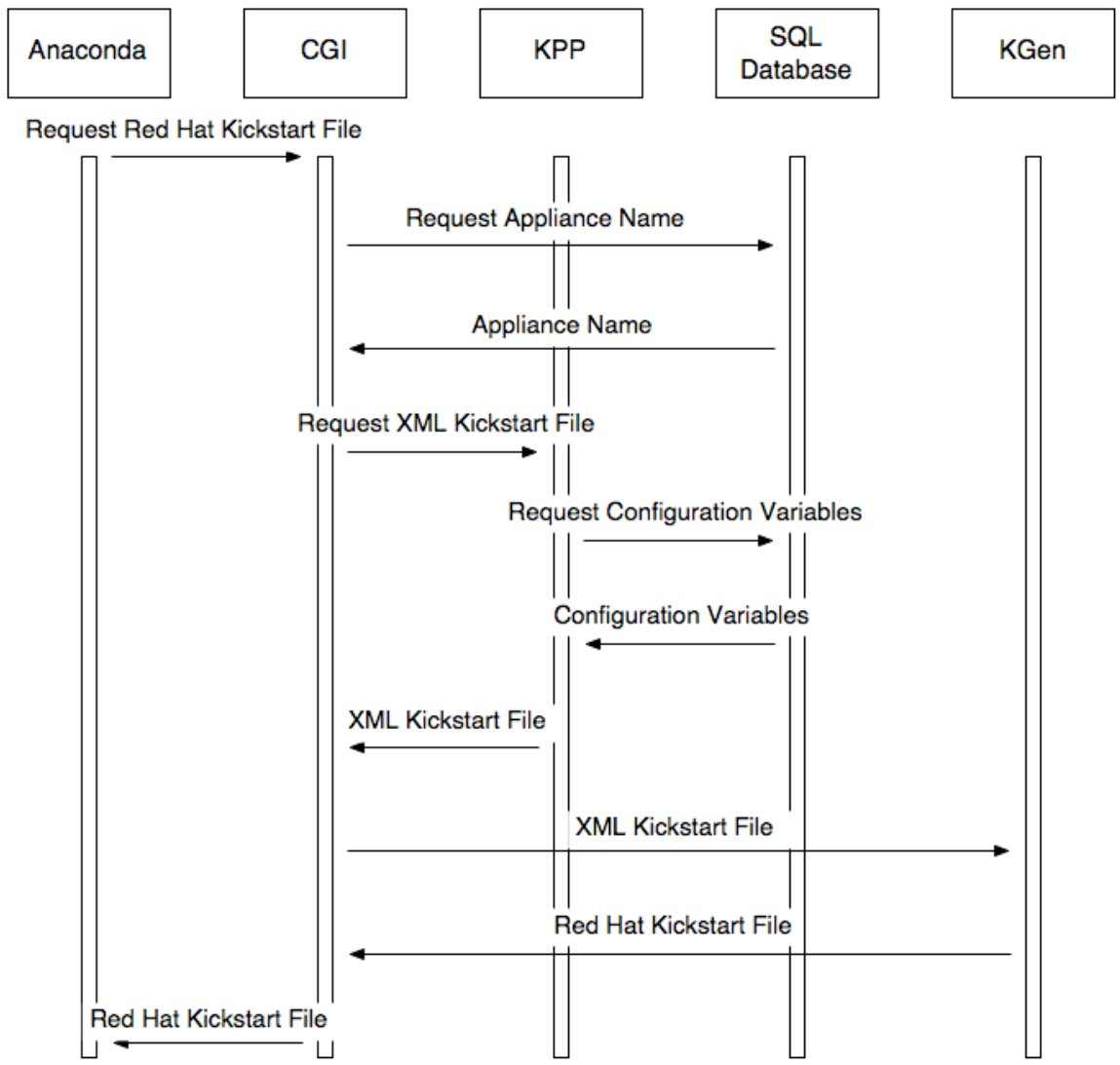


Kickstart File

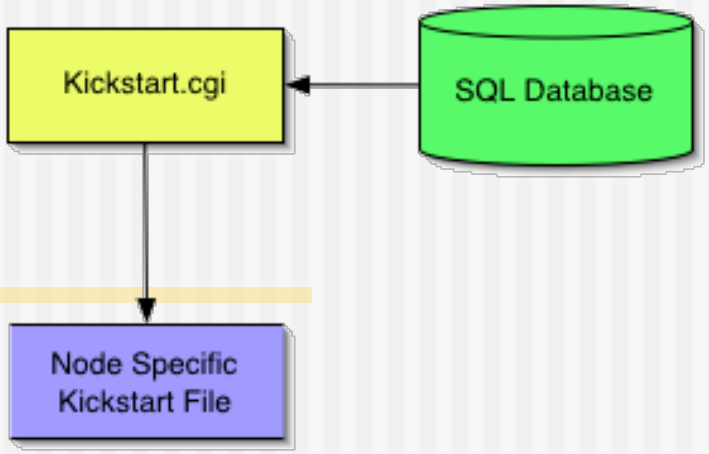
- RedHat's Kickstart: DNA of a node
  - Monolithic flat ASCII file
    - "Main": disk partitioning, timezone
    - "Packages": list of RPM names
    - "Post": shell scripts for config
  - No macro language
  - Requires forking based on site information and node type.



Getting A Kickstart File

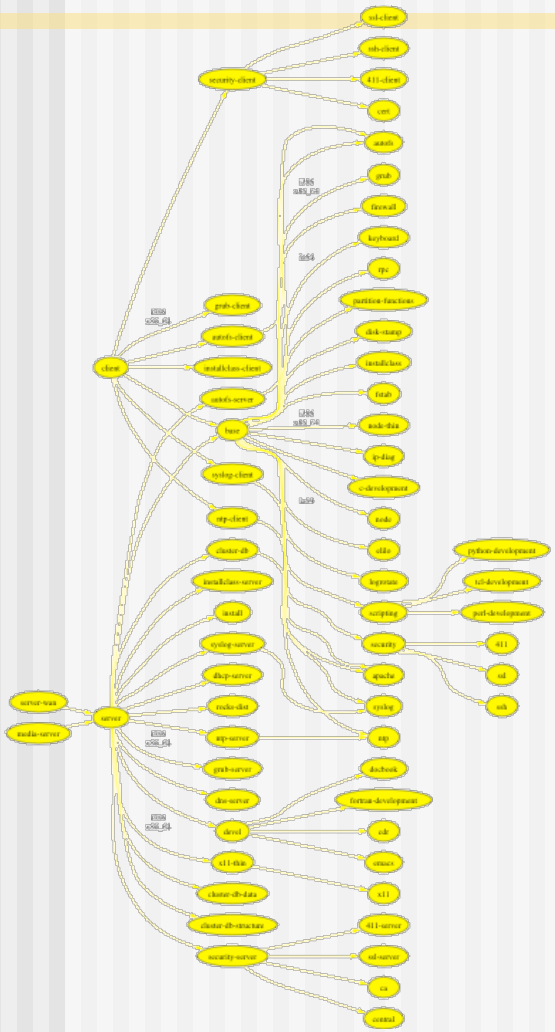


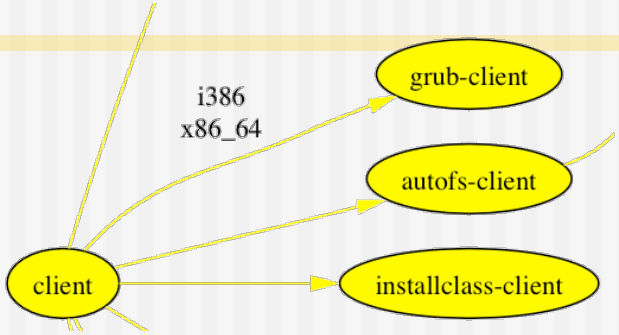
Kickstart File



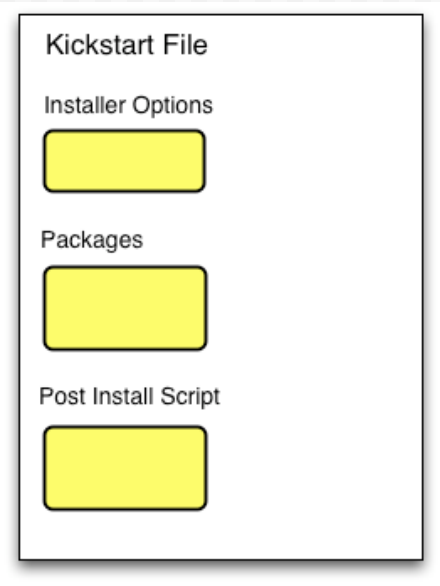
- Rocks XML Kickstart
  - Decompose a kickstart file into nodes and a graph
    - Graph specifies OO framework
    - Each node specifies a service and its configuration
  - SQL Database to help site configuration
  - “Compile” flat kickstart file from a web cgi script

Kickstart Graph for Kgen

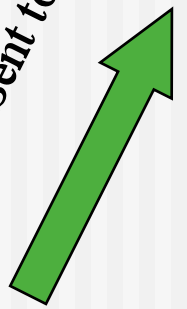




Preprocess (kpp)



Sent to node (http)



Kickstart Graph with Roll