Simplified Virtualization in a HEP/NP Environment with Condor

RHIC/ATLAS Computing Facility at Brookhaven National Laboratory

William Strecker-Kellogg <willsk@bnl.gov>
Costin Caramarcu <caramarc@bnl.gov>
Christopher Hollowell <hollowec@bnl.gov>
Tony Wong <tony@bnl.gov>

Introduction

Cloud computing—otherwise known as infrastructure as a service (IaaS), or on-demand virtual-machine instantiation, provides some exciting opportunities for systems administrators and physicists alike in the HEP community.

Faced with the problem of supporting legacy OS's after a refresh, we aimed to examine what it would take to implement a cloud-computing solution with a minimum of extra expense and software. This involved integration with Condor (our batch system), ensuring security, access to data, and adequate performance.

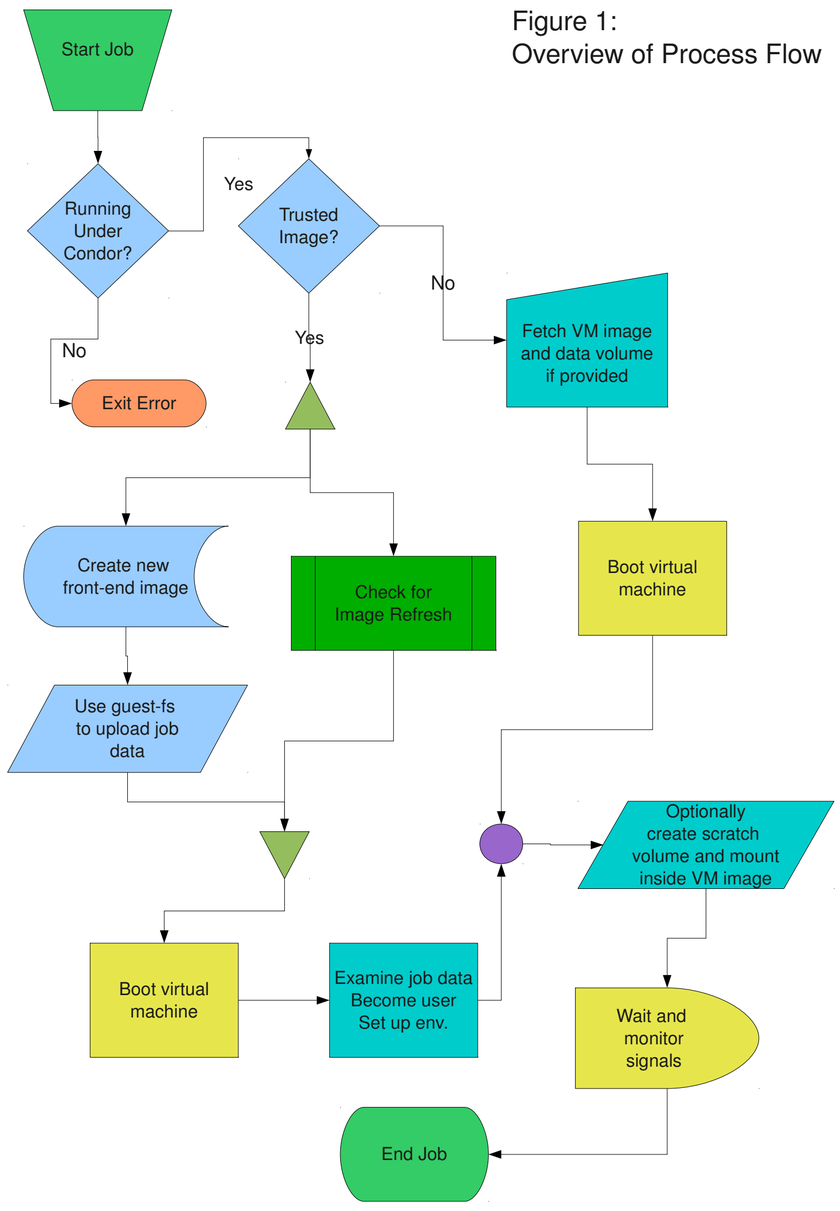

Figure 1:
Overview of Process Flow

Image Distribution

Images are stored on a webserver and distributed over HTTP. This enables us to leverage well understood tools like SQUID for performance and scalability. Images are cached on the local node, and updates are pulled each time a job starts.

Getting Images and Updates

A checksum file is maintained at the URL where images are fetched from. Each job gets this file and compares it to a copy on the local node. Images whose checksums differ are re-downloaded. In pseudo-python the refresh cycle looks like this:

```
import racf

# Get checksum file from webserver if not present
racf.download_file(web_checksum, web_cached_sums)

present_images = set(racf.present_images(image_dir))

local_sums = racf.ChecksumList(local_checksum)
web_sums = racf.ChecksumList(web_cached_sums)

# Expired local images present but not in remote checksum file
racf.rm_filelist([image_dir + x for x in present_images if x not in web_sums])

need_to_get = set([x for x in local_sums if x not in present_images])

outdated = racf.compare_sums(web_sums, local_sums)
if outdated:
    need_to_get |= set([x for x in outdated])

if image not in present_images:
    need_to_get.add(image)

for img in need_to_get:
    racf.download_file(url + img, image_dir + img, buf=32 * 1024, unzip=True)

new_sums = racf.get_checksums(need_to_get, image_dir)
for x in new_sums:
    local_sums.add(x, new_sums[x])

bad_sums = local_sums.diff(web_sums)

if len(bad_sums) > 0:
    log.critical("BAD checksums found, terminate immediately")
    sys.exit(1)
```

Images are compressed on the webserver and are uncompressed as they are downloaded to the node. The checksums are of the uncompressed images since verification is done on the node once a download/unzip is complete.

Image Maintenance

Images will be kept patched by periodically booting the master copies, running a "`yum update`", recomputing the checksums, compressing them, and uploading the new image and checksum file to the master webserver.

Security Concerns

We allow trusted images only for now; this solves two problems—security and image contextualization.

Untrusted images can be run only behind a NAT that remaps ports below 1024 to high ports. NFS and other services on our network use the fact that only root can bind to "low" ports to authenticate clients.

Users run a setuid-binary to instantiate the images, but this can only be run underneath each Condor job so other users and jobs cannot invoke it (parent process must be `condor_starter`). This is secure under Linux as we check that the parent exe-symlink is correct: `/proc/<ppid>/exe -> /usr/bin/condor_starter`.

Security cont.

Inside trusted VM's we inject a custom `rc.local` that inspects the job information uploaded in the image, becomes the appropriate UID and executes the indicated code.

Inside untrusted VM's, perhaps paradoxically, users can claim-to-be-root since ports are re-mapped and they can do no harm on the network.

Machine Instantiation

Done through a combination of directly using the QEMU/KVM toolset and the libvirt bindings, as well as libguestfs for contextualization.

Networking

For security reasons we cannot allow untrusted images to run on our network with access to low ports (see Security section). The best solution that affords optional port remapping is using a NAT. This involves creating a virtual bridge and a tun/tap device behind this bridge using `iptables` to do the port remapping if necessary. `libvirt` will configure this as well.

Storage

A new empty qcow2 image is created for each job in the scratch area with the appropriate backing image behind it.

CPU and Memory Specs

Will match what is available per condor-slot (2Gb/core currently). Inspection is done by the instantiation script of the current condor job settings and these are set accordingly.

Contextualization

Condor job information is uploaded to a specified location inside trusted images using `libguestfs`. If necessary other context can be added (someday perhaps puppet certificates to allow untrusted VM's to point to their own configuration management servers)

I/O Methods

Job Input/Output

Most users will utilize their already-existing NFS home-directories for their code and will be analyzing data there as well.

Images can be optionally provided by users and mounted inside virtual machines as a `/data` partition, likewise a `/scratch` partition can be exported after a the job is complete.

Job Scratch Area

Because the performance of qcow2 with an unallocated backing store is not optimal, jobs that plan on using large amounts of scratch space can optionally request a blank image be created for them and mounted as a directory specified by a flag in the job

Condor Tie-In

The batch system can be used in two ways, one where it just executes a wrapper to boot the VM and another where it boots the VM itself using a feature known as the Condor "VM Universe"

Running Jobs

First, condor is used to manage the deployment of VM's. The user sees as an "API" just their regular condor job with an additional few flags:

```
Command = /usr/local/racf/vm
Args = -i sl_53
+VM_Command = "/phenix/u/chris/cornfop/exe/job.sh"
+VM_Args = "/phenix/data33/run10/data_file $(ClusterID)"
```

The user modifies the command and arguments to be our wrapper and provides extra flags determining what runs inside the virtual machine.

Job Parameters and Setup

Condor provides a number of environment variables and overwrites, we provide these inside the VM by taking the environment of the wrapper and injecting it to the VM. The working directory and temp areas are created inside the VM by the setup script and the environment is loaded prior to user job execution.

Condor Instantiation

Condor has a concept called the "VM Universe" that allows the actual instantiation to be handled by the condor daemons. The ability to control exactly how this was done was added in recent versions. In versions of condor >7.6 there is a flag called LIBVIRT_XML_SCRIPT that allows the administrator to control the logic of how images are instantiated. This script takes the job-description as input and outputs the XML description used by `libvirt` to instantiate the VM.

This can replace the custom instantiation scripts because it allows us to validate whether the image is trusted and therefore whether or not to do the port-remapping.

US ATLAS

RHIC
Brookhaven National Laboratory's Relativistic Heavy Ion Collider

DEPARTMENT OF ENERGY • UNITED STATES OF AMERICA •

BROOKHAVEN NATIONAL LABORATORY