# HEPIX Virtualisation Working Group

**2011-04-01**

# HEPIX Virtualisation Working Group: 2011-04-01

Owen Synge
Virtualisation Working Group HEPiX

**Abstract**

# Table of Contents

# Preface

## Table of Contents



HEPIX is a twice yearly conferance made up from sites providing compute and storage services for High Energy Physics. HEPIX focusses on the compute, network and storage infrastucture required for high energy physics sites.

# HEPIX Virtulisation Working Group Observations

The spring 2009 HEPIX conferance made the following observations:

- High Energy Physics will want consistent images across sites.
- The Grid is a homogenious OS enviorment so requiring High Energy Physics to corrdinate OS upgrades across multiple experiments.
- The Grid will need to be shared with non High Energy Physics comunitys.
- High Energy Physics experiments are very coupled to OS.
- Hardware is coupled to OS version.
- Sites belive `Virtual Machine Image`'s are going to be the execution enviroemnt for High Energy Physics jobs in the future.
- Change Root, and Xen based `Vritualisaed Execution Enviroments` where already present at HEPIX sites.

With these observations the HEPIX Virtulisation Working Group was formed.

A new generation of `Vritualisaed Execution Enviroments` such as Open Nebular and Nimbus where already being developed.

# HEPIX Virtulisation Working Group Objectives

- Produce a framework to securly run `Virtual Machine Image`'s across multiple sites supporting High Energy Physics.
- Sites need control over `Virtual Machine Image` selection.

With these objectives the HEPIX Virtulisation Working Group was created with Tony Cass as its leader.

# Part I. HEPIX Virtulisation Working Group Organisation

## Table of Contents



This part of the book sumarises the organsiation of the HEPIX Virtulisation Working Group.

# Chapter 1. Organisation Introduction

Owen Synge

# Introduction

HEPIX generated documentation includes many details on:

- How to make virtual machines images platform neutral.
- How to deploy site neutral images and contextualise them to a site such as batch queue integration.
- Meta data (and structure) required to describe an image for HEPIX needs, developed from multiple virtual machine management software.
- Software to mange images at a site.
- Requirements for image distribution across sites.
- Most of the software to securely transfer images between sites.

# Part II. Image Trust

## Table of Contents



This part of the document covers secuirty requirements.

# Chapter 2. Policy on the Endorsement of Virtual Machine Images

David Kelsey

The master copy of this document can be found here https://edms.cern.ch/document/1080777

This copy of the document was taken from Draft v1.4b writern on 21 May 2010

# Introduction

This document describes the security-related policy requirements for the generation and endorsement of trusted virtual machine (VM) images for use on the Grid.

The aim is to enable Grid Sites to trust and instantiate endorsed VM images that have been generated elsewhere.

The virtualisation model addressed here is the use of virtual Grid worker nodes that act in a similar way to real worker nodes. Virtualisation provides an efficient way of managing different configurations of worker node, e.g. the operating system used, and importantly different pre-configured application environments for the VOs. The model addressed here, therefore, simply provides a different way of running authorized VO work, transparent to the end user, exactly the same as if the user payload was running on a real worker node. There should be no need to place more restrictions on virtual worker nodes running endorsed images as defined by this policy, than on real worker nodes in terms of access to trusted local services at the site.

This policy does not compel Sites to instantiate images endorsed in accordance with this policy nor limit the rights of a Site to decide to instantiate a VM image generated by any other non-compliant procedures, should they so desire. The Site is still bound by all applicable Grid security policies and is required to consider the security implications of such an action on other Grid participants.

# Definitions

The following terms are defined.

- *VM base image:* A VM image, including a complete operating system and all general middleware, libraries, compilers, programmes and utilities. All kernel and root-level configurations, including any that may be VO-specific, are included here.

- *VO environment:* The VO-specific middleware, application software, libraries, utilities, data and configuration which may be necessary to provide the appropriate environment for use by members of a VO. No kernel modifications or root-level configurations are included here.

- *VM complete image:* The VM image resulting from the combination of the VM base image and the VO environment (if any).

- *Globally Unique Identifier:* A unique identifier for a VM complete image.

- *Endorser:* An individual who confirms that a particular VM complete image has been produced according to the requirements of this policy and states that the image can be trusted.

# Policy Requirements

An Endorser should be one of a limited number of authorised and trusted individuals appointed either by a VO or a Site. The appointing VO or Site must assume responsibility for the actions of the Endorser and must ensure that he/she is aware of the requirements of this policy.

# Policy Requirements on the Endorser

By acting as an Endorser you agree to the conditions laid down in this document and other referenced documents, which may be revised from time to time.

# Policy Requirements on the Endorser

1. You are held responsible by the Grid and by the Sites for checking and confirming that a VM complete image has been produced according to the requirements of this policy and that there is no known reason, security-related or otherwise, why it should not be trusted.

2. You recognise that VM base images, VO environments and VM complete images, must be generated according to current best practice, the details of which may be documented elsewhere by the Grid. These include but are not limited to:

   a. Any image generation tool used must be fully patched and up to date;

   b. All operating system security patches must be applied to all images and be up to date;

   c. Images are assumed to be world-readable and as such must not contain any confidential information.

   d. there should be no installed accounts, host/service certificates, ssh keys or user credentials of any form in an image.

   e. Images must be configured such that they do not prevent Sites from meeting the fine- grained monitoring and control requirements defined in the Grid Security Traceability and Logging policy to allow for security incident response.

   f. the image must not prevent Sites from implementing local authorisation and/or policy decisions, e.g. blocking the running of Grid work for a particular user.

3. You must disclose to the Grid or to any Site on request the procedures and practices you use for checking and endorsing images.

4. You must provide and maintain an up to date digitally signed list of your currently endorsed images together with the metadata relating to each VM image, as defined in the VM Image Catalogue document.

5. You must keep an auditable history of every image endorsed including the Globally Unique Identifier, date/time of generation and full list of OS/packages/versions in both the VM Base Image and VO Environment. This must be made available to sites on demand.

6. You must remove images from the approved list whenever a problem is found, e.g. a new security update is required. This removal must also be recorded locally in your auditable history.

7. You are responsible for handling all problems related to the inclusion of any licensed software in a VM image. You shall ensure that any software included in a VM image which is used for its intended purposes, complies with applicable license conditions and you shall hold the Site running the image free and harmless from any liability with respect thereto.

8. You must assist the Grid in security incident response and must have a security vulnerability assessment process in place.

9. You recognise that the Grid, the Sites, and/or the VOs reserve the right to block any endorsed image or terminate any instance of a virtual machine and associated user workload for administrative, operational or security reasons.

10. You recognise that if a Site runs an image which no longer appears on your list of endorsed images, that you are not responsible for any consequences of this beyond the time of your removal of the image from the list.

# Part III. Image Creation

## Table of Contents



This part of the working group covers how to generate `Virtual Machine Image`’s so that they can be easily shared in a site and hypervisor neutral way.

# Chapter 3. Multiple hypervisor support

Andrea Chierici

In this short report I will describe the possibility to configure a VM to be run "almost" unchaged under xen (both para and hvm) and kvm. No other hv has been considered at this stage.

This report is rather technical and will describe the operations a sysadmin has to perform in order to achieve the same result.

Some assumptions have been taken for this guide:

- a dhcp is present on the net and configured with mac addresses of the VMs
- VMs get ip via dhcp
- VMs IPs are not NATted by hypervisors
- VMs use disk-on-a-file approach
- VMs disk format is raw
- VMs use virtio drivers

# Testbed

The testbed I used is composed by two identica dual socket machines, configured in this way:

- 2x e5420 processors
- 16GB RAM
- 2x sas disk
- gigabit ethernet

The operating system running on the hosts is SL5.4, kernel version 2.6.18-164.11.1.el5. Node 1 shares a partition via nfs (mounted by node 2), in order to simplify image distribution between the 2. The dom0/ kvm_guest are configured following the standard documentation provided by redhat:

```
http://www.redhat.it/docs/en-US/Red_Hat_Enterprise_Linux/5.5/html/Virtualization_Guide/index.html
```

Since in my configuration all the VMs have public IP I configured network like this:

```
http://www.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/5.5/html/Virtualization_Guide/sect-
Virtualization-Network_Configuration-Bridged_networking_with_libvirt.html
```

# Virtual Machine Setup

The initial installation of the VM is done on a KVM host. Anyway it’s rather easy to do the same starting from xen (not documented).

The command to install the machine with kvm is the following:

```
virt-install \
 --connect qemu:///system \
 --name wn-test-01 \
 --accelerate \
 --ram 2048 \
 --disk path=/virtual_shared/wn-test-01_disk,size=25 \
 --network bridge:br0 \
 --mac 00:16:3e:00:00:1f \
 --arch x86_64 \
 --location http://os-server.cnaf.infn.it/distro/SL/54/x86_64/ \
 --vnc \
 --os-type=linux \
 --os-variant=virtio26
```

This command creates a node called "wn-test-01" with 2GB of ram, 25GB of hard disk (raw) stored in /virtual_shared, uses network bridge br0 to connect to the network, emulates a 64bit machine with virtio drivers, both for net and disk; SL54 is installed via network.

Please consider that --disk path requires an existing path on your host server and that --location requires a mirror of the opeating system you want to install.

Optionally it’s possible to add a kickstart file with the line:

```
--extra-args ks=http://(url where to get a ks)
```

See appendix for a sample ks file.

If not specifically required, I suggest not to use LVM for disk partitions. After installation is complete, we can reboot the node and begin customizing it for multiple HV support.

firstboot is executed when a machine is rebooted and I suggest to do the following customizations using this tool:

- selinux disabled
- firewall disabled
- bluetooth,cups,firstboot,gpm,ip6tables,isdn,kudzu,pcscd services disabled

After completing this step, reboot (selinux requires this).

# Multiple HV customization

## kvm -> xen-hvm

On the running VM, after having disabled kudzu (chkconfig kudzu off) you need to modify the conf file /etc/sysconfig/network-scripts/ifcfg-eth0 in this way:

Original file:

```
DEVICE=eth0
BOOTPROTO=dhcp
HWADDR=00:16:3E:00:00:1F
ONBOOT=yes
```

Modified file:

```
DEVICE=eth0
BOOTPROTO=dhcp
ONBOOT=yes
```

That is, basically, to remove the HWADDR line. In this way when the VM is started on a different HV, the different network card is detected and flawlessly configured. All the other devices *should* be automatically detected and the VM should boot regularly.

Edit /etc/modprobe.conf and remove the entry on eth0.

To boost performances we will create an initrd.img file that supports para-virtualized drivers for both xen and kvm.

To do so, issue these commands on the VM:

```
cd /boot
mkinitrd -f -v --with=xen-balloon --with=xen-vbd --with=xen-vnif --with=xen-platform-pci
initrd-paravirt.img `uname -r`
```

Now edit grub.conf and change the default kernel initrd with the one just created.

In order to support paravirt disk on xen-hvm you need to add "ide0 noprobe ide1 noprobe" to the kernel.

A sample of the resulting grub entry is the following:

```
title Scientific Linux SL (2.6.18-194.3.1.el5)
        root (hd0,1)
        kernel /vmlinuz-2.6.18-194.3.1.el5 ro root=LABEL=/ ide0=noprobe ide1=noprobe
        initrd /initrd-paravirt.img
```

Now we can shut down our VM and prepare the xen template.

On the dom0 node, you need to add the config for the xen-hvm machine.

cd to /etc/xen and add a file with these settings:

```
kernel = "/usr/lib/xen/boot/hvmloader"
builder = 'hvm'
memory = "2048"
name = 'wn-test-01'
disk = ['tap:aio:/virtual_shared/wn-test-01_disk,xvda,w']
vif = [ 'type=netfront, mac=00:16:3e:00:00:1f' ]
device_model = '/usr/lib64/xen/bin/qemu-dm'
vcpus = '1'
vnc=1
serial = "pty"
vnclisten="0.0.0.0"
vncconsole=1
boot='c'
pae='1'
```

Change name and mac address according to your configuration. Now try starting the VM with "xm create wn-test-01".

Everything should work fine and you are ready now to add support for xen-para.

# kvm-> xen-para

Switching from kvm to xen-para is just a little bit more complicated. First of all it’s necessary to install a kernel-xen on the VM with:

```
yum -y install kernel-xen
```

then, since xen-para uses para-virtualized disk devices, you need to create a custom initrd in this way:

```
cd /boot
mkinitrd -f -v --preload=xenblk --with=xennet --builtin=virtio_pci --builtin=virtio --
builtin=virtio_blk --builtin=xen_vbd initrd-xen-para.img 2.6.18-194.3.1.el5xen
```

The version of the kernel may differ, depending on the one that was installed with yum command.

Now you should change the grub.conf and add the new initrd to the kernel-xen using a different syntax than the one used by default by xen:

```
title Scientific Linux SL (2.6.18-194.3.1.el5xen)
        root (hd0,0)
        kernel /vmlinuz-2.6.18-194.3.1.el5xen ro root=LABEL=/
        initrd /initrd-xen-para.img
```

I highly recommend configuring this kernel as the second one, leaving the vanilla kernel as the default (default=0): to do so move the kernel-xen stanza in second place.

To support the serial console, you should edit /etc/inittab in this way:

After:

```
# Run gettys in standard runlevels
```

add this line

```
co:2345:respawn:/sbin/agetty xvc0 9600 vt100-nav
```

and comment out

```
2:2345:respawn:/sbin/mingetty tty1
```

and the following...

A sample of the configuration file for the dom0 to run this node is the following:

```
memory = "2048"
name = 'wn-test-01'
disk = ['file:/virtual_shared/wn-test-01_disk,xvda,w']
vcpus = '1'
vif = [ 'mac=00:16:3e:00:00:1f' ]
bootloader="/usr/bin/pygrub"
bootargs="--entry=1"
on_reboot = 'restart'
on_crash = 'restart'
```

Notice that bootargs contains the entry of the kernel-xen in the grub.conf file, and so it must reflect exactly the entry you used in the conf file. If you followed my guide, 1 is ok.

# Kickstart file automatically generated by anaconda.

```
Install
url --url http://os-server.cnaf.infn.it/distro/SL/54/x86_64/
```

```
lang en_US.UTF-8
keyboard us
network --device eth0 --bootproto dhcp
rootpw --iscrypted $1$RBWTG48f$5Hcoi3Kh9hQeRb.S/r6vL/
firewall --enabled --port=22:tcp
authconfig --enableshadow --enablemd5
selinux \u2013enforcing
timezone --utc Europe/Rome
bootloader --location=mbr --driveorder=vda
# The following is the partition information you requested
# Note that any partitions you deleted are not expressed
# here so unless you clear all partitions first, this is
# not guaranteed to work
#clearpart \u2013linux
#part /boot --fstype ext3 --size=100
#part swap --size=4000
#part / --fstype ext3 --size=100 --grow --asprimary
%packages
@base
@core
@dialup
@editors
@java
@misc-sl
@text-internet
fipscheck
device-mapper-multipath
sgpio
alpine
```

# Part IV. Image transfer

## Table of Contents



This part is to cover image transfer issues.

# Chapter 4. Virtual Image distribution mechanism

Martin Bly

Owen Synge

This paper describes a possible distribution mechanism for virtual images to be used at wLCG sites, addressing various issues such as validation, verification, expiry and revocation. The actual distribution technology is not discussed in detail.

# Summary

Images are referenced by an Image List which contains a secure hash (sha512) signed using x509 technology. These Image Lists are published, and interested sites subscribe to the Lists in a catalogue at the site. When an instantiation request for an image is received, the Image validity is checked. If the Image list is valid, the Image is contextualised and then instantiated. Images that do not pass validation are not instantiated.

# Description

The idea is that a list of virtual images and the associated metadata is signed with a x509 certificates. This provides a mechanism by which a virtual image can be checked for validity: expiry, revocation, whether it has been tampered with etc, and traceability.

The image is then distributed to a site using (for example) the software distribution mechanism by the responsible VO, or automatically by the local catalogue. Information about the image (the certificate public keys, VO responsible, a unique ID, a VO tag family and a URI, inital global validity, i.e., the metadata) is registered with a local catalogue. A local validity flag may also be set by the site.

Requests for instantiation are made via the unique ID or the tag. When the local system receives an instantiation request for a particular image, the catalogue is checked for the location and global and local validity flags. If both are OK, the image is then checked for expiry and revocation using the x509 credentials. If the the chek is passed then the image is contextualised and instantiated. Expired or or revoked images are refused.

# Image list Signing

An Image will be created by a Creator and added to an Image List which is then signed by the Creator’s x509 certificate. The Image List will have an appropriate expiry for the Image in the Image List. The Image may then be distributed by an appropriate mechanism.

Images may be revoked by not including the Image in the latest version of the Image List. The URI to establish the latest version of the Image List will be contained in the Image List metadata.

# Distribution

The Image List is provided on a host that can be authenticated to prevent man-in-the-middle attacks blocking update notices. An image may be distributed to Sites by any suitable means including the

existing software distribution mechanism, via a URI referance in the metadata. The transfer can be validated using a secure hash which is expected to be sha512.

While any check on the authenticity of the host, would safely prevent man in the middle attacks it may be easier to use X509 authentication for this function.

# Local Catalogue

A Site will have a Local Catalogue which will be a registry (database) of Images which may be considered for running at the Site. The Catalogue will contain metadata for each subscribed Image and Image List. This metadata will at least contain the x509 certification data, the unique UUID, a Tag, resource locator, a global validity flag field and a local validity flag field.

The Local Catalogue will subscribe to the appropriate revocation service (CRLs) and flag Images that have been revoked by setting the global validity flag.

A local validity flag may be set against an Image, to allow local administrations to "blacklist" an Image. The local validity flag may also be set to "run anyway" to allow expired but NOT revoked images to be run

# Registration/Subscription

A distributed Image List will be subscribed into the local Catalogue by the distribution mechanism. The subscriber will write the Image List metadata to the Catalogue and set the global validity flag by checking the latest CRLs. Images may be subscribed locally.

# Instantiation

A request to the instantiation system to run an Image will cause the instantiation system to validate the Image against its global and local validity flags and metadata in the Local Catalogue, and to check for the expiry of the Image using its x509 certificate. If an image passes all the required checks, the Image may be contextualised and then run.

# Image Metadata

Images will have x509 signed metadata. Images will have a unique identifier (UUID). Images will have a Tag name which may be non-unique such that a series of images may belong to a tag family. Requests to run an image may be either via the UUID or the Tag. In the case of a Tag having more than one UUID registered the latest dated image will be considered for instantiation.

# Chapter 5. Image transfer Use Cases

Ulrich Schwickerath

Owen Synge

# Use Cases

## 1: Endorser handling

**Each endorser runs his own VMIC.**

- Endorsers can be associated to a site (side endorser)
- Endorsers can be associated to a VO (VO endorser)

### 1.1: A new Endorser for VO XYZ enters

- The endorser sets up his VMIC
- The endorser VMIC gets listed in the VOs list of endorsers

#### 1.1.1:

- The change gets announced to the sites supporting this VO

#### 1.1.2:

- The sites decide to update their local list of endorsers or not
- The sites get the list of images of this new endorser from the endorsers VMIC

#### 1.1.3:

- The sites start the local approval procedure for the images of this endorser

### 1.2: An Endorser leaves VO XYZ and is not replaced

#### 1.2.1: The VO desides to revoke all his images

- The endorser is removed from the VOs list of endorsers
- The endorsers VMIC is destroyed
- The sites supporting this VOs are informed of the change by the VO
- The sites update their local list of endorsers
- The sites ensure that local copies of all affected images are revoked

#### 1.2.2: The VO wants the sites to keep running his images

- in this case the VO must provide a new endorser who re-endorses the images, case 1.3

### 1.2.3: As the endorsers VMIC has been destroyed, no new images can be released by this person

•

## 1.3: An Endorser E1 of VO XYZ is replaced by Endorser E2

• The VO follows step 1.1 to support the new endorser, and then 1.2 to revoke the old one

## 1.4: VO A elects a new Endorsers E1. Site S1 supports VO A. Site S2 does not support VO A

### 1.4.1/2:

• implicitly fulfilled as the VO only contacts those sites which support it

# 2: Endorser actions

## 2.1: A new image is released. Endorser A blesses it for publication

• The endorser adds the image to his VMIC

a. sites regularly poll the VMICs of all endorsers in their local endorser list for updates

b. VOs can ask sites to perform this check outside the normal intervals

• sites start the local approval procedure to make the image available

### 2.1.1/2: Fulfilled by construction

## 2.2: Endorser A revokes a previously blessed images, eg because a bug has been found, without replacing it

• The endorser removes these images from his VMIC

• sites regularly poll the VMICs in their local endorser list

• VOs can ask sites to poll outside the regular interval

• sites remove all images which are no longer in the VMICs from their local list of approved images

## 2.3: Endorser A wants to update an existing image

• The endorser adds the new image to his VMIC

• The endorser removes the old image from his VMIC

• sites regularly poll the VMICs in their local endorser list, and revoke support for all images which may have disappeared from the VMICs

- VOs can ask sites to poll outside the regular interval
- sites decide on the approval of the updated image, and add the image to their list of approved images
- sites remove all images which are no longer in the VMICs from their local list of approved images

# 3: Site actions

## 3.1: A site B wants to use a site specific image for local users

- the site's local endorser endorsed the image
- the site approves it's own image
- the image is added to the sites list of approved images

### 3.2: Site B has created an image which is useful for other sites and wants to share it

- the site's endorser adds the image to his VMIC
- remote sites supporting this endorser will pick up the image during regular update checks
- the remote sites launch their local approval procedure
- the remote sites add the new image to their local list of approved images (or not)

### 3.3: A site is asked to run an image from a trusted endorsers, and the site policies allow this

- implicitely handled

### 3.4: A site is asked to run an image from a trusted endorser, but site policies forbid them to run it

- implicietely handled. The site does not approve the image and does not add it to it's local list of approved images

### 3.5: A site needs to stop running images for type XYZ because of some reason (security, bug, wrong software,changed site policies)

- the site gets notified by an external instance of the problem
- the site checks the meta data information of the images in their local list of approved images
- the site flags image for removal based on their meta data
- the site removes the affected images from their local list of approved images
- in case of a VO, the affected VO is informed
- the VO proceeds as described above for the revokation of the image

### 3.6: A site needs to stop running all images of Endorser XYZ, which were previously trusted and used

- The endorser wil have disappeared from the list of trusted endorsers
- images of this endorser get removed from the local list of approved images

### 3.7: A site needs to make a new image available to their users

- The image has been added upstream and appears in one of the VMICs
- At the next poll, or on request of the VO, the approval procedure is started
- the image is added or not to the local list of approved images

### 3.8: Endorser A is replaced by Endorser B. Images of A are no longer trusted, Images of Endorser B are trusted.

- see 1.3

# Chapter 6. Virtual Image transfer software

Owen Synge

This chapter describes the software components that have been and need to be developed to complete cross site image transfer.

The software and implementation is only one of many posible solutions that could be produced to solve the objectives of the HEPIX Virtulisation Working Group.

# Imagelist Creation

This application creates manages and signs image lists.

## Status

This software is available at:
`https://github.com/hepix-virtualisation/hepixvmitrust`

The software is production grade, and has been used by Ian Gable to generate signed image lists.

### Features

- Command line application with library to support integration.
- Adds X509 Signature to lists of images.
- Ability to add and remove images to image list.
- Automatic setting of some meta data fields.

### To Do (15-04-2011)

- Auto add CA and DN fields to image list.
- Import signed image lists for modification.
- Support optional data fields.

# Imagelist Consumption

This application subscribes to image lists, downloads the latest and authenticates it.

## Status

This software is available at:
`https://github.com/hepix-virtualisation/hepixvmilsubscriber`

The software is not yet production grade, when it is it will provide the list of all trusted images and thier image lists the sites has subscibed to. This application will also retrive the imagelist updates.

This program will not provide an interface for admins to manage which images will be approved by the site as this functionality is provided by the VMIC.

## Features

- Add subscription.
- Delete subscription.
- Update subscriptions checking authenticity of the message.

## To Do (15-04-2011)

- Only message authenticity is checked, does not yet check authenticity of transport.
- Does not yet provide the tools to query by image.
- Does not yet provide the tools to query by image, and retrive signerd image list.
- Untested on production grade Data bases.

While it does check the authenticity of the message using X509, at the moment the authenticity of the host is unchecked. Programatically it would be far simpler to use x509 certificates to check the host server. In terms of deployment it would be far easyer just to check any host key mechanism, as this is sufficent.

The database does currently store the signed image list and can be queried by image, but no inteface is provided as yet. This feature is blocking the production of the image cache, or integration with the VMIC.

# VMIC

This application manages which images can be deployed at a site.

# Status

This software is available at:
`svn co svn+ssh://svn.cern.ch/reps/batchinter/trunk/virtualization/vmic/`

This software has been deployed at CERN for some months and is currently used to manage worker node deployment.

## Features

- Single web interface to manage image deployment as a site.
- Allows multiple admins to manage site deployment.
- Provides the sites signed list of approved images to worker nodes.
- In production since October 2010 at CERN.

**Note**
Both the producer of an image and the site produce signed list of images, these provide different functions, the image is signed by the producer to show to the site the image came from the producer, while the site may sign a list of images to allow them to be run at a site.

## To Do (15-04-2011)

- Integration with Imagelist Consumption.
- Integration with Imagelist Production.

# Image Cache

This application manages the downloads of images from the imagelist.

It is unclear if this should be stand alone product depending on the ImageList Consumer, for cache expiry, or part of the integration process of the image list consumer with the VMIC.

# Status

This software is not started:

# Chapter 7. Image Lists

Image lists are good because.

# Structure of message

```
hv:imagelist
  dc:date:created
  dc:date:expires
  hv:endorser
    hv:ca
    hv:dn_x509
    hv:email
    dc:creator
  dc:identifier
  dc:description
  dc:title
  hv:images
    hv:image
      dc:title
      dc:description
      hv:size
      sl:checksum:sha512
      sl:arch
      hv:uri
      dc:identifier
      sl:os
      sl:osversion
      sl:comments
      hv:hypervisor
      hv:version
  dc:source
  hv:version
  hv:uri
```

# Imagelist Fields

## hv:imagelist

container for metadata

```
<hv:imagelist>...
</hv:imagelist>
```

## hv:imagelist/dc:date:created

date in format year-month-dayTHour:Minute:SecondTimezone

```
<dc:date type="created">2011-03-03</dc:date>
```

## hv:imagelist/dc:date:expires

date in format year-month-day when imagelist will no longer be trusted

```
<dc:date type="created">2011-03-03</dc:date>
```

```
hv:imagelist/hv:endorser
  comment)
    container for endorser metadata
  xml example)
    <hv:endorser>...
    </hv:endorser>
```

```
hv:imagelist/hv:endorser/hv:ca
  comment)
    standard hash id for the CA, corresponds with the CA directory files.
  xml example)
    <hv:ca>/C=GB/ST=Greater Manchester/L=Salford/O=Comodo CA Limited/CN=AAA Certificate Services</
hv:ca>
```

```
hv:imagelist/hv:endorser/hv:dn_x509
  comment)
    x509 DN of the endorser
  xml example)
    <hv:dn_x509>/C=DE/O=GermanGrid/OU=DESY/CN=Owen Synge</hv:dn_x509>
```

```
hv:imagelist/hv:endorser/hv:email
  comment)
    email address of the endorser
  xml example)
    <hv:email>owen.synge@desy.de</hv:email>
```

```
hv:imagelist/hv:endorser/dc:creator
  comment)
    Name of the endorser
  xml example)
    <hv:email>Owen Synge</hv:email>
```

```
hv:imagelist/dc:identifier
  comment)
    RFC 4122 UUID for imagelist this allows updating the list, and uniqueness.
  xml example)
    <dc:identifier>9ac60a74-c67f-457f-bd80-04c8ff6ecf3d</dc:identifier>
```

```
hv:imagelist/dc:description
  comment)
    description of the image list
  xml example)
    <dc:description>Owens list of images from DESY quattor.</dc:description>
```

```
hv:imagelist/dc:title
  comment)
    title of image list
  xml example)
```

```
    <dc:description>DISH service</dc:description>
```

```
hv:imagelist/hv:images
  comment)
    Container of the list of images
  xml example)
    <hv:images>....
    </hv:images>
```

```
hv:imagelist/hv:images/hv:image
  comment)
    Container of the image
  xml example)
    <hv:image>....
    </hv:image>
```

```
hv:imagelist/hv:images/hv:image/dc:title
  comment)
    Title of image
  xml example)
    <dc:title>generic:sl5</dc:title>
```

```
hv:imagelist/hv:images/hv:image/dc:title
  comment)
    Name of the image
  xml example)
    <dc:title>generic:sl5</dc:title>
```

```
hv:imagelist/hv:images/hv:image/dc:description
  comment)
    Free text describing the image
  xml example)
    <dc:description>Desy generated generic sl5 image</dc:description>
```

```
hv:imagelist/hv:images/hv:image/hv:size
  comment)
    size of the image in bytes
  xml example)
    <hv:size>4619442176</hv:size>
```

```
hv:imagelist/hv:images/hv:image/sl:checksum:sha512
  comment)
    size of the image in bytes
  xml example)
    <sl:checksum
 type="sha512">8b4c269a60da1061b434b696c4a89293bea847b66bd8ba486a914d4209df651193ee8d454f8231840b7500fab6740620c71
sl:checksum>
```

```
hv:imagelist/hv:images/hv:image/sl:arch
  comment)
    architecture of the image typically 'i386' or 'amd64'
  xml example)
    <sl:arch>amd64</sl:checksum>
```

```
hv:imagelist/hv:images/hv:image/hv:uri
  comment)
    URI pointing to the latest version of this file.
  xml example)
    <hv:uri>https://example.org/imagelist.xml</hv:uri>
```

```
hv:imagelist/hv:images/hv:image/dc:identifier
  comment)
    RFC 4122 UUID for image this allows updating the image, and uniqueness.
  xml example)
    <hv:uri>185c9f57-f838-4ac6-8246-85ccd6d6d3f4</hv:uri>
```

```
hv:imagelist/hv:images/hv:image/sl:os
  comment)
    The operating system the image runs, examples include "Linux" "BSD" "FreeBSD"
  xml example)
    <sl:os>Linux</sl:os>
```

```
hv:imagelist/hv:images/hv:image/sl:osversion
  comment)
    The operating system version the image runs, examples include "SL_6.0", maybe this
    tag should be further broken downs so easier to process for computers.
  xml example)
    <sl:osversion>SL_5.5</sl:osversion>
```

```
hv:imagelist/hv:images/hv:image/sl:comments
  comment)
    Stratus lab suggest comments on how the image is to be deployed.
  xml example)
    <sl:comments></sl:comments>
```

```
hv:imagelist/hv:images/hv:image/hv:hypervisor
  comment)
    Typically set to reflect the Virtualization technology values such as "xen", "kvm".
  xml example)
    <hv:hypervisor></hv:hypervisor>
```

```
hv:imagelist/hv:images/hv:image/hv:version
  comment)
    The version of the image, so that updates can share the same metadata
    (with exception of dc:identifier) in case of update. Typically a numeric value.
```

```
  xml example)
    <hv:version></hv:version>
```

```
hv:imagelist/dc:source
  comment)
    The URI where the latest version of this file can be found.
  xml example)
    <dc:source type="uri">http://example.org/example.img.gz</dc:source>
```

```
hv:imagelist/hv:version
  comment)
    The version of the imagelist, so that updates can share the same metadata
    (with exception of dc:identifier) in case of update. Typically a numeric value.
  xml example)
    <hv:version></hv:version>
```

```
hv:imagelist/hv:uri
  comment)
    The uri to retrive new versions of the imagelist, that updates can be handled.
  xml example)
    <hv:uri></hv:uri>
```

# Example metadata files

A typical image list looks like such

```
$cat imagelist.json
{
    "dc:date:created": "2011-03-10T17:09:12Z",
    "dc:date:expires": "2011-04-07T17:09:12Z",
    "dc:description": "a README example of an image list",
    "dc:identifier": "4e186b44-2c64-40ea-97d5-e9e5c0bce059",
    "dc:source": "example.org",
    "dc:title": "README example",
    "hv:endorser": {
        "hv:x509": {
            "dc:creator": "Owen Synge",
            "hv:ca": "/C=DE/O=GermanGrid/CN=GridKa-CA",
            "hv:dn": "/C=DE/O=GermanGrid/OU=DESY/CN=Owen Synge",
            "hv:email": "owen.synge@desy.de"
        }
    },
    "hv:images": [
        {
            "hv:image": {
                "dc:description": "This is an README example VM",
                "dc:identifier": "488dcdc4-9ab1-4fc8-a7ba-b7a5428ecb3d",
                "dc:title": "README example VM",
                "hv:hypervisor": "kvm",
                "hv:size": 2147483648,
                "hv:uri": "http://example.org/example-image.img",
                "hv:version": "1",
                "sl:arch": "x86_64",
                "sl:checksum:sha512":
 "8b4c269a60da1061b434b696c4a89293bea847b66bd8ba486a914d4209df651193ee8d454f8231840b7500fab6740620c7111d9a17d08b74
                "sl:comments": "Vanila install with contextulization scripts",
                "sl:os": "Linux",
                "sl:osversion": "SL 5.5"
```

```
            }
        }
    ],
    "hv:uri": "http://example.org/example-image-list.image_list",
    "hv:version": "1"
}
```

and when signed like this.

```
MIME-Version: 1.0
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg="sha1";
 boundary="----EAE3006C97F670EE450F46AC8DF4C070"

This is an S/MIME signed message

------EAE3006C97F670EE450F46AC8DF4C070
{
    "dc:date:created": "2011-03-10T17:09:12Z",
    "dc:date:expires": "2011-04-07T17:09:12Z",
    "dc:description": "a README example of an image list",
    "dc:identifier": "4e186b44-2c64-40ea-97d5-e9e5c0bce059",
    "dc:source": "example.org",
    "dc:title": "README example",
    "hv:endorser": {
        "hv:x509": {
            "dc:creator": "Owen Synge",
            "hv:ca": "/C=DE/O=GermanGrid/CN=GridKa-CA",
            "hv:dn": "/C=DE/O=GermanGrid/OU=DESY/CN=Owen Synge",
            "hv:email": "owen.synge@desy.de"
        }
    },
    "hv:images": [
        {
            "hv:image": {
                "dc:description": "This is an README example VM",
                "dc:identifier": "488dcdc4-9ab1-4fc8-a7ba-b7a5428ecb3d",
                "dc:title": "README example VM",
                "hv:hypervisor": "kvm",
                "hv:size": 2147483648,
                "hv:uri": "http://example.org/example-image.img",
                "hv:version": "1",
                "sl:arch": "x86_64",
                "sl:checksum:sha512":
 "8b4c269a60da1061b434b696c4a89293bea847b66bd8ba486a914d4209df651193ee8d454f8231840b7500fab6740620c7111d9a17d08b74
                "sl:comments": "Vanila install with contextulization scripts",
                "sl:os": "Linux",
                "sl:osversion": "SL 5.5"
            }
        }
    ],
    "hv:uri": "http://example.org/example-image-list.image_list",
    "hv:version": "1"
}
------EAE3006C97F670EE450F46AC8DF4C070
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"

MIIHdAYJKoZIhvcNAQcCoIIHZTCCB2ECAQExCzAJBgUrDgMCGgUAMAsGCSqGSIb3
DQEHAaCCBSUwggUhMIIECaADAgECAgIz7DANBgkqhkiG9w0BAQUFADA2MQswCQYD
VQQGEwJERTETMBEGA1UEChMKR2VybWFuR3JpZDESMBAGA1UEAxMJR3JpZEthLUNB
MB4XDTExMDExMDE1MDMxN1oXDTEyMDIwOTE1MDMxN1owRjELMAkGA1UEBhMCREUx
EzARBgNVBAoTCkdlcm1hbkdyaWQxDTALBgNVBAsTBERFU1kxEzARBgNVBAMTCk93
ZW4gU3luZ2UwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCkgbPFZrVL
pmwf7GKBBFkwTK5V7RmlupsU3Z3FqdfMnJGn2NrrnHIhthUTCTq4WbLIZTbOEh0n
JqZgZBvYcwJV4V9pais4YlsEug+JLMbB9hZ6e2XgdjXWgLqz6vBSIf6KXi4KhCxe
a4FylvIk7OtY+bg0mg5IFHib6uP7fXhFKdBEapoi+B05wpluBMA+2DBdSt+rjzA8
SwiHUuan60VIyJAxammyOe3IKSpwyBXkQ10XjIhIpoSavqYXJboFOVzUcqxawdbX
Con2W8QfiwFKYupohG/VTusDXFT2MP4k+KxG3/rTTPWUDJme7VUPv3+CTcEO+z4v
X8/XhI44oAXlAgMBAAGjggInMIICIzAMBgNVHRMBAf8EAjAAMA4GA1UdDwEB/wQE
```

```
AwIE8DAdBgNVHQ4EFgQUGakUy66kgvulNBIf18WBXjGolqYwXgYDVR0jBFcwVYAU
xnXJKKzRC/w8/7m1HtNfO4BiEjShOqQ4MDYxCzAJBgNVBAYTAkRFMRMwEQYDVQQK
EwpHZXJtYW5HcmlkMRIwEAYDVQQDEwlHcmlkS2EtQ0GCAQAwHQYDVR0RBBYwFIES
b3dlbi5TeW5nZUBkZXN5LmRlMB8GA1UdEgQYMBaBFGdyaWRrYS1jYUBpd3IuZnpr
LmRlMDUGA1UdHwQuMCwwKqAooCaGJGh0dHA6Ly9ncmlkLmZ6ay5kZS9jYS9ncmlk
a2EtY3JsLmRlcjAaBgNVHSAEEzARMA8GDSsGAQQBlDarLAEBAQUwEQYJYIZIAYb4
QgEBBAQDAgWgME4GCWCGSAGG+EIBDQRBFj9DZXJ0aWZpY2F0ZSBpc3N1ZWQgdW5k
ZXIgQ1AvQ1BTIHYuIDEuNSBhdCBodHRwOi8vZ3JpZC5memsuZGUvY2EwJAYJYIZI
AYb4QgECBBcWFWh0dHA6Ly9ncmlkLmZ6ay5kZS9jYTAzBglghkgBhvhCAQgEJhYk
aHR0cDovL2dyaWQuZnprLmRlL2NhL2dyaWRrYS1jcHMucGRmMDMGCWCGSAGG+EIB
AwQmFiRodHRwOi8vZ3JpZC5memsuZGUvY2EvZ3JpZGthLWNybC5kZXIwDQYJKoZI
hvcNAQEFBQADggEBAMbn91TOQ6r4D/aKwgIFXiXe40B7iccz/P5pCFSi1R6IC3KH
Ui4s/f9iAGl9rA21h8QAaRaJ/h1OQNlgMZbc9jDCWcqxr8wQTYAQDiBkspLT68ZO
5xVFRiq3HjkkhwnFfFzsNSiLFYZTRjChPluclYG3TEvSg8dz9Lv/IEJxE5C5lZ2d
e3CSu0vcD0DESiu/sVqPOOHi8NL/59U2ine3z23Y+piCabQCxjT0inT2MmR8UNDF
ij2JJYxlt56U/SQCEe0304w3x1jIg8vcpm4dfh+L2IjJ9hVfEeLaCyhv9Wjbmu5O
vk0yLjcEZ7b4RKeo7djVYh+5kCWJYCr/W6uGW44xggIXMIICEwIBATA8MDYxCzAJ
BgNVBAYTAkRFMRMwEQYDVQQKEwpHZXJtYW5HcmlkMRIwEAYDVQQDEwlHcmlkS2Et
Q0ECAjPsMAkGBSsOAwIaBQCggbEwGAYJKoZIhvcNAQkDMQsGCSqGSIb3DQEHATAc
BgkqhkiG9w0BCQUxDxcNMTEwMzEwMTczMzU1WjAjBgkqhkiG9w0BCQQxFgQUd43y
VT05Zk+7acFF+EeqExNI57cwUgYJKoZIhvcNAQkPMUUwQzAKBggqhkiG9w0DBzAO
BggqhkiG9w0DAgICAIAwDQYIKoZIhvcNAwICAUAwBwYFKw4DAgcwDQYIKoZIhvcN
AwICASgwDQYJKoZIhvcNAQEBBQAEggEAkA0RgB5AkGIYvFsFETzx7QHKWu9qas5k
vlHn2a+EpRE9K1p+qrFNzS53E2BGqubyRcePfgG/WyGqYOK2h20d6GZH+ENUFkvM
EAthbvQaHye6WEvF/0GUrr0QUBT1gQswkkryPHcqTVmJANQORakkNvCwynEBmfSC
vb2TEppRuOCmxx3zqrzMr7zPNPY4w2+YaXQ1fHfmEmOrlf0ImP20TyTKIoQWqzbq
WXwlRhZBUoD9zfiEM/jFvOvkuxLkQeiEcSzlLAGHXsHJ3anPMX9sobJFbJI0wYdN
sUOInHRhksokh2ow68KZK4vXLI173v5yZE7FZZ1Gl9T+YpkmOIW4iQ==

------EAE3006C97F670EE450F46AC8DF4C070--
```

# Part V. Contextulisation

## Table of Contents



Contextulisation is teh process of making a `Virtual Machine Image` run at a site.

# Chapter 8. Contextulisation-recipe

Sebastien Goasguen

Certainly it does not represent all possible solutions to the problem but shows an end to end mechanism for a VM produced at INFN to start at CERN.

We can add the VMIC checks to bridge the two properly.

# Contextulisation recipe in 2 parts

## Download image from Andrea

### Part 1

Download image from Andrea

```
[root] #wget http://www.cnaf.infn.it/~chierici/repo/test-seba.img.bz2
--2010-07-06 15:54:46--  http://www.cnaf.infn.it/~chierici/repo/test-seba.img.bz2
Resolving www.cnaf.infn.it... 131.154.3.61
Connecting to www.cnaf.infn.it|131.154.3.61|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 531888187 (507M) [application/x-bzip2]
Saving to: `test-seba.img.bz2'

100%[======================================================================================>]
 531,888,187 5.19M/s   in 86s

2010-07-06 15:56:12 (5.91 MB/s) - `test-seba.img.bz2' saved [531888187/531888187]
```

### Handling the Image

```
[root] #ls -l
total 519956
drwx------ 2 root root     16384 Apr  1 17:22 lost+found
-rw-r--r-- 1 root root 531888187 Jul  6 15:19 test-seba.img.bz2
```

Bunzip it

```
[root] #bunzip2 test-seba.img.bz2
[root] #ls -l
total 10250036
drwx------ 2 root root       16384 Apr  1 17:22 lost+found
-rw-r--r-- 1 root root 10485760000 Jul  6 15:19 test-seba.img
```

### Inspection

#### Partitions

Start Inspection steps

```
[root] #kpartx -a test-seba.img
[root] #ls -l /dev/mapper/
total 0
crw------- 1 root root  10, 62 May  6 15:26 control
brw-r----- 1 root disk 253, 26 Jul  6 16:12 loop0p1
brw-r----- 1 root disk 253, 27 Jul  6 16:12 loop0p2
brw-r----- 1 root disk 253, 28 Jul  6 16:12 loop0p3
```

## Check partition 1

Check first partition....See the kernels

```
[root] #mount /dev/mapper/loop0p1 /mnt
[root] #cd /mnt
[root] #ls
config-2.6.18-164.2.1.el5      initrd-2.6.18-194.8.1.el5xen.img  symvers-2.6.18-164.2.1.el5.gz
 vmlinuz-2.6.18-164.2.1.el5
config-2.6.18-194.8.1.el5xen   initrd-paravirt.img               symvers-2.6.18-194.8.1.el5xen.gz
 vmlinuz-2.6.18-194.8.1.el5xen
grub                           initrd-xen-para.img               System.map-2.6.18-164.2.1.el5
 xen.gz-2.6.18-194.8.1.el5
initrd-2.6.18-164.2.1.el5.img  lost+found                        System.map-2.6.18-194.8.1.el5xen
 xen-syms-2.6.18-194.8.1.el5
```

## Check Grub Config

Checking the Grub config has One kernel for KVM, one kernel for paravirtualized Xen

```
[root] #cd grub
[root] #cat grub.conf
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,0)
#          kernel /vmlinuz-version ro root=/dev/hda3
#          initrd /initrd-version.img
#boot=/dev/hda
default=0
timeout=5
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Scientific Linux (2.6.18-164.2.1.el5)
        root (hd0,0)
        kernel /vmlinuz-2.6.18-164.2.1.el5 ro root=LABEL=/
        initrd /initrd-paravirt.img
title Scientific Linux SL (2.6.18-194.8.1.el5xen)
        root (hd0,0)
        kernel /vmlinuz-2.6.18-194.8.1.el5xen ro root=LABEL=/
        initrd /initrd-xen-para.img
```

## Check Other Partitions

```
[root] #umount /mnt
[root] #mount /dev/mapper/loop0p2 /mnt
/dev/mapper/loop0p2 looks like swapspace - not mounted
mount: you must specify the filesystem type
[root] #mount /dev/mapper/loop0p3 /mnt
[root] #cd mnt
[root] #ls
bin   dev  home  lib64       media  mnt  poweroff  root  selinux  sys  usr
boot  etc  lib   lost+found  misc   opt  proc      sbin  srv      tmp  var
```

```
[root] #
```

### Unmounting Partitions

umount and delete device map

```
[root] #umount /mnt
[root] #kpartx -d test-seba.img
```

## Prepare volume for disk image

Could use the file directly if you wanted to

```
[root] #lvcreate -L25G -n andrea xen_vg
  Logical volume "andrea" created

[root] #dd if=/stagein/test-seba.img of=/dev/xen_vg/andrea bs=4M
2500+0 records in
2500+0 records out
10485760000 bytes (10 GB) copied, 142.881 seconds, 73.4 MB/s
```

## Handling the Image

Create Xen config file....or libvirt xml description

```
[root] #more /etc/xen/andrea
name   = "andrea"
bootloader = "/usr/bin/pygrub"
bootargs="--entry=1"
memory = 1024
vcpus = 1
on_poweroff = "destroy"
on_reboot = "restart"
on_crash = "restart"
disk   = ["phy:/dev/mapper/xen_vg-andrea,xvda,w"]
vif    = ["mac=<PUT SOMETHING VALID AT YOUR SITE HERE>,bridge=xenbr0,script=vif-bridge"]
```

## Handling the Image

Start VM

```
[root] #xm create andrea
Using config file "/etc/xen/andrea".
Started domain andrea

[root] #xm list
Name                                        ID Mem(MiB) VCPUs State   Time(s)
Domain-0                                     0     2048     8 r----- 105172.5
andrea                                      29     1023     1 -b----     10.7

[root] #xm console andrea
<snip>...
Starting auditd: [  OK  ]
Starting system logger: [  OK  ]
Starting kernel logger: [  OK  ]
Starting irqbalance: [  OK  ]
Starting portmap: [  OK  ]
Starting NFS statd: [  OK  ]
Starting RPC idmapd: [  OK  ]
Starting system message bus: [  OK  ]
Mounting other filesystems:  [  OK  ]
```

```
Starting HAL daemon: [  OK  ]
Starting hidd: [  OK  ]
Starting autofs:  Loading autofs4: [  OK  ]
Starting automount: [  OK  ]
[  OK  ]
Starting sshd: [  OK  ]
Starting sendmail: [  OK  ]
Starting sm-client: [  OK  ]
Starting crond: [  OK  ]
Starting anacron: [  OK  ]
Starting atd: [  OK  ]
Starting Avahi daemon... [  OK  ]
Starting smartd: [  OK  ]

Scientific Linux SL release 5.4 (Boron)
Kernel 2.6.18-194.8.1.el5xen on an x86_64

localhost.localdomain login:
```

## Handling the Image

In CERN's case we would then make an opennebula template for this VM

Something like the following

> **Note**
> this particular template has not been tested yet

```
[one] #more andrea.template

NAME = ANDREA
CPU    = 0.1
MEMORY = 512
```

kernel and boot device

```
OS = [  bootloader = "/usr/bin/pygrub", bootargs = "--entry=1" ]
```

1 disk assumes volume is present on Hypervisors

```
DISK = [
  type = "block",
  source   = "/dev/xen_vg/andrea",
  target   = "xvda",
  readonly = "no" ]
```

Use Leasing of Public Network MAC/IPs

```
NIC = [ NETWORK="LXCLOUDIPs" ]
```

Context disk

```
```

```
CONTEXT = [
vmid       = "$VMID",
TTL        = "48",
files      = "/opt/vmimage/vmcontext.prologue, /opt/vmimage/vmcontext.prologue",
target     = "xvdb"
]
```

Scheduling

```
REQUIREMENTS = "MACS=\"*$NIC[MAC, NETWORK=\"LXCLOUDIPs\"]*\""
RANK = FREEMEM
```

# Part 2

I got Andrea's VM to start in both Xen paravirt and KVM. For the KVM one I even used opennebula.

I attach a terse transcript of the commands and configuration files used.

Missing component is a recipe for the contextualization as well as recipe to get the VM via proper VMIC mechanism.

```
[root] #more andrea
name   = "andrea"
bootloader = "/usr/bin/pygrub"
bootargs="--entry=1"
memory = 1024
vcpus = 1
on_poweroff = "destroy"
on_reboot = "restart"
on_crash = "restart"
disk   = ["phy:/dev/mapper/xen_vg-andrea,xvda,w"]
vif    = ["mac=00:16:3E:00:03:49,bridge=xenbr0,script=vif-bridge"]
```

## Let's start it with Xen

```
[root] #xm create andrea
Using config file "./andrea".
Started domain andrea
```

Using virsh you can get a libvirt xml description file

## basic xen cfg file

It is Probably easier to start with a basic xen cfg file

```
[root] #virsh dumpxml andrea
<domain type='xen' id='41'>
  <name>andrea</name>
  <uuid>64034478-9051-3010-be8d-1469c44021a3</uuid>
  <memory>1048576</memory>
  <currentMemory>1048576</currentMemory>
  <vcpu>1</vcpu>
  <bootloader>/usr/bin/pygrub</bootloader>
  <bootloader_args>--entry=1</bootloader_args>
  <os>
```

```
    <type>linux</type>
    <kernel>/var/lib/xen/boot_kernel.URfEIv</kernel>
    <initrd>/var/lib/xen/boot_ramdisk.gqW2y2</initrd>
    <cmdline>ro root=LABEL=/</cmdline>
  </os>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <disk type='block' device='disk'>
      <driver name='phy'/>
      <source dev='/dev/mapper/xen_vg-andrea'/>
      <target dev='xvda' bus='xen'/>
    </disk>
    <interface type='bridge'>
      <mac address='00:16:3e:00:03:49'/>
      <source bridge='xenbr0'/>
      <script path='vif-bridge'/>
      <target dev='vif41.0'/>
    </interface>
    <console type='pty' tty='/dev/pts/5'>
      <source path='/dev/pts/5'/>
      <target port='0'/>
    </console>
  </devices>
</domain>
```

# Opennebula Template

In Opennebula the VM template for a KVM host is

```
[one] #more andrea.test

NAME = Andrea
CPU    = 0.1
MEMORY = 512
```

kernel and boot device

> **Note**
> we don't use the bootargs entry

```
OS = [  bootloader     = "/usr/bin/pygrub" ]
```

1 disk defined, disk image was dd’d in a local LVM volume

```
DISK = [
  type = "block",
  source   = "/dev/xen_vg/andrea",
  target   = "hda",
  readonly = "no" ]
```

Use Leasing of Public Network MAC/IPs

You might want to specify MAC directly

In this case we are doing a lease

```
NIC = [ NETWORK="CLOUDIPs" ]
```

No contextualization for now, start VM as is

Scheduling , update matchmaking for your own setup

```
REQUIREMENTS = "MACS=\"*$NIC[MAC, NETWORK=\"CLOUDIPs\"]*\""
RANK = FREEMEM
```

End of the Opennebula template

# Starting the VM in OpenNebula

```
[one] #onevm create andrea.test
```

```
[one] #onevm list | grep Andrea
 76225 oneadmin   Andrea runn   0       0       example.org 00 00:57:36
 76226 oneadmin   Andrea runn   0       0       example.org 00 00:52:14
```

On the KVM host

```
[root] #virsh list
 Id Name                 State
----------------------------------
  3 one-76225            running
  4 one-76226            running
```

you might want to check the xml description generated by opennebula.

you can use it to start the VM with virsh without opennebula.

```
<domain type='kvm'>
        <name>one-76226</name>
        <memory>524288</memory>
        <os>
                <type>hvm</type>
                <bootloader>/usr/bin/pygrub</bootloader>
                <boot dev='hd'/>
        </os>
        <devices>
                <emulator>/usr/bin/kvm</emulator>
                <disk type='block' device='disk'>
                        <source dev='/opt/opennebula/76226/images/disk.0'/>
                        <target dev='hda'/>
                </disk>
                <interface type='bridge'>
                        <source bridge='br0'/>
                        <mac address='00:16:3e:00:02:d3'/>
                </interface>
        </devices>
        <features>
                <acpi/>
        </features>
</domain>
```

# Checking the KVM processes

### Note

You could have started these VMs without virsh or Opennebula

```
[root] #ps -ef | grep kvm
```

```
root      2396     1  0 10:25 ?        00:00:27 /usr/bin/kvm -S -M rhel5.4.0 -m 512
-smp 1 -name one-76225 -uuid 1d2ce4aa-9168-b0a7-950b-91b04ab3bb51 -no-kvm-pit-reinjection
-nographic -monitor pty -pidfile /var/run/libvirt/qemu//one-76225.pid -boot c -drive
file=/opt/opennebula/76225/images/disk.0,if=ide,index=0,boot=on -net
 nic,macaddr=00:16:3e:00:02:d1,
vlan=0 -net tap,fd=18,script=,vlan=0,ifname=vnet1 -serial none -parallel none -usb
root      2803     1  0 10:31 ?        00:00:27 /usr/bin/kvm -S -M rhel5.4.0 -m 512
-smp 1 -name one-76226 -uuid 06a609e1-9d11-831c-1510-5efdf70622af -no-kvm-pit-reinjection
-nographic -monitor pty -pidfile /var/run/libvirt/qemu//one-76226.pid -boot c -drive
file=/opt/opennebula/76226/images/disk.0,if=ide,index=0,boot=on -net
 nic,macaddr=00:16:3e:00:02:d3,
vlan=0 -net tap,fd=17,script=,vlan=0,ifname=vnet0 -serial none -parallel none -usb
```

Bridge networking is used with each VM is attached to the bridge

```
[root] #brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.003048d2bdf0       no              vnet0
                                                        vnet1
                                                        eth0
virbr0          8000.000000000000       yes
```

# Chapter 9. Contextualization principles for HEPiX

Sebastien Goasguen

Ruben Montero

Ulrich Schwickerath

Certainly it does not represent all possible solutions to the problem but shows an end to end mechanism for a VM produced at INFN to start at CERN.

We can add the VMIC checks to bridge the two properly.

# Definition

Contextualization is defined as the process by which a virtual machine instance is configured based on virtual machine master image. In general, contextualization consists in passing arbitrary data to the virtual machine at boot time. The goal of this process is to configure generic installations of system services (e.g. sshd) and it is the basis to implement a "install once deploy many" strategy. It was first introduced by Kate Keahey of Argonne National lab (http://www.nimbusproject.org/ )

# Virtual Appliances

A Virtual Appliance is a virtual machine image with a set of pre-installed software packages designed to run on a given hypervisor. A successful contextualization strategy starts at the Virtual Appliance creation, by defining a set a conventions and pre-configurations that will allow the Virtual Appliance (also refer as master image) to be specialized at boot time.

wo aspects of the contextualization process were introduced and discussed within the context of running virtual machines from the OSG VO: STAR. (Ref: https://twiki.grid.iu.edu/bin/view/ CampusGrids/ DeployingTheSTARVM); and can be refered to as image level contextualization and instance level contextualization.

# Image Level Contextualization

Image level contextualization applies to the Virtual Appliance building process and includes:

- Image Format, the representation of the disk's data within the image file (e.g. raw, qcow2...). The image format may be hypervisor specific.
- Image Layout, refers to how the various partitions are placed on the disk and to what other disk structures are present (e.g. swap FS).
- Software appliance, the software packages installed in the image and its preparation for integration with other site services. For example, shared file system support or batch scheduler software.

# Instance Level Contextualization

Whereas image-level contextualization can be performed manually by a systems administrator, instance-level contextualization occurs once per VM instantiation and as such must be automated. Note that

certain resources must be leased from the physical site. These resources include network addresses, disk space, and scheduler slots.

Basic instance level contextualization is based on the availability of data not contained in the base image but thereforth available in the instance. In its most basic form it is achieved via the use of a CDrom device, or attached disk. The CD-ROM content is dynamically created and mounted within the instance. This basic process is compliant with OVF standards defined at ( Ref: http://www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf , pp 38 ), this process is used in OpenNebula (Ref: http://www.opennebula.org/documentation:rel2.0:cong )

While these basic mechanisms work for single instances, a more involved contextualization may be needed when multiple instances need to contextualize themselves with respect to each other. There are two approaches:

- This may be better achieved by a context broker such as the one defined in Nimbus (Ref: http://www.nimbusproject.org/docs/2.2/faq.html#ctxbroker). In practice, the context broker is a server that the instance contact to get their configuration parameters. This maybe a mechanisms that the VO could use for post instantiation configuration. As long as it complies with the HEPiX security policies.
- Use of VM orchestration utilities, this approach could be more flexible as it also allows complete management of the VM group and it can include synchronization points between VMs. These utilities implements typical DAG dependencies, so for example NFS clients are booted after the NFS server. An example of such VM orchestrator for OpenNebula can be found at (Ref: http://dev.opennebula.org/projects/oneservice).

# Contextualization for Cloud Computing

Cloud computing represents the on-demand instantiation of virtual machines per user request. In that mode of operation the contextualization is achieved when the user passes data to the cloud API, data that is then made available within the instance. As an example, the following strategies are available in commercial providers:

- Image layout conventions (e.g. do not include swap partitions or user data FS should be availble under /dev/sde).
- Valid Image Formats.
- Recommend software packages (e.g. ssh)
- Virtual Appliance Metadata (e.g. name of maintainer, version and date of creation, contents...)
- Support for the HEPiX instance level contextualization process.

Therefore the scope of the HEPiX contextualization \u201cpolicy\u201d is defined as instance level contextualization of a third party trusted VM image when the image needs to share services with the local site infrastructure. In the following, we assume that the context data is made available to the VM in the form of a context CD-ROM. We see different contextualization strategies in terms of the flexibility offered to the user of the Virtual Appliance.

- Variables. A set of predefined variables are recognized. The Virtual Applianceis prepared in such a way that variables are read and the corresponding services configured. Examples may include, SSH_PUB_KEY NFS_SERVER_IP, TCP_IN_PORTS...
- Files. The contents of predefined files are made available at boot time. The Virtual Appliance is built to read the context of these files and place them in their target locations at a very early stage of the boot process.

- Scripts. The VM is supplied with custom scripts to be executed at a given point of the booting process or/and the shutdown process. The scripts can be predefined to hook in specific points of the booting process (e.g. prologue.script, epilogue.script, or rc2.script ...).

Note that the previous strategies can be combined as needed. Also note that the context service would need to be present in the image, this service would be run at a very early stage of the boot process and check whether the context disk is present or not according to the Image layout conventions (see above).

The previous contextualization processes requires that the users made available the context data to the remote site. There are three basic procedures:

- The user is able to upload files and contents to the site. This implies a file storage service to be available in the sites and is not recommended.
- The user made available the contents of variables, files or scripts in the form of an URL.
- The user embed the contents of variables, files or scripts in the VM creation request (e.g. as part of an XML request using base64 encoding). Given the nature of the content data (i.e. very limited size) this would be the preferred method.

# CERN current site contextualization

The VM contains two init scripts in /etc/init.d, vmcontext.prologue and vmcontext.epilogue

vmcontext.prologue is:

```
#!/bin/bash
# vmcontext       Bring up/down contextualization
#
# chkconfig: 2345 2 91
# description: HEPiX compliant contextualisation epilog
#
### BEGIN INIT INFO
# Provides: $vmcontext_prolog
### END INIT INFO
#
# Changelog:
#             $Log: vmcontext,v $
#             Revision 1.2 2010/08/06 19:10:36 uschwick
#             add support for KVM
#
#             Revision 1.1 2009/12/03 08:15:59 uschwick
#             import initial version of vmcontext from SG
#
#
# Source function library
. /etc/init.d/functions
start() {
    if [ -e /dev/xvdb ] ; then
         disk="/dev/xvdb"
    elif [ -e /dev/vdb ] ; then
         disk="/dev/vdb"
    elif [ -e /dev/hdb ]; then
         disk="/dev/hdb"
    elif [ -e /dev/sdb ]; then
         disk="/dev/sdb"
    else
         echo $"No context disk found."
         exit 0;
    fi
    echo -n $"Trying to mount context disk $disk: "
    mount -t iso9660 $disk /mnt
    RETVAL=$?
    if [ 0 -eq $RETVAL ] ; then
         echo -n "Performing contextualization initial steps"
```

```
        if [ -e /mnt/prolog.sh ]; then
             . /mnt/prolog.sh
        else
             . /mnt/init.sh
        fi
        RETVAL=$?
        if [ 0 -eq $RETVAL ]; then
             echo " DONE"
        else
             echo " FAILED"
        fi
        umount /mnt
        return $RETVAL
    else
         umount /mnt
         return $RETVAL
    fi
}
stop() {
    echo -n $"Unmouting context disk: "
    umount /mnt
    RETVAL=$?
    echo
    return $RETVAL
}
case "$1" in
    start)
#                [ $running -eq 0 ] && exit 0
                 start
                 RETVAL=$?
                 ;;
    stop)
#               [ $running -eq 0 ] || exit 0
                stop
                RETVAL=$?
                ;;
        *)
        echo $"Usage: $0 {start|stop}"
        RETVAL=2
esac
exit $RETVAL
```

And the vmcontext.epilogue is (very similar):

```
#!/bin/bash
# vmcontext epilog     contextualization - final steps after boot
#
# chkconfig: 2345 99 2
# description: HEPiX compliant contextualisation epilog
#
### BEGIN INIT INFO
# Provides: $vmcontext_epilog
### END INIT INFO
#
# Changelog:
#              $Log: vmcontext,v $
#              Revision 1.2 2010/08/06 19:10:36 uschwick
#              add support for KVM
#
#              Revision 1.1 2009/12/03 08:15:59 uschwick
#              import initial version of vmcontext from SG
#
#
# Source function library
. /etc/init.d/functions
start() {
    if [ -e /dev/xvdb ] ; then
         disk="/dev/xvdb"
    elif [ -e /dev/vdb ] ; then
         disk="/dev/vdb"
    elif [ -e /dev/hdb ]; then
         disk="/dev/hdb"
    elif [ -e /dev/sdb ]; then
```

```
          disk="/dev/sdb"
     else
          echo $"No context disk found."
          exit 0;
     fi
     echo -n $"Trying to mount context disk $disk: "
      mount -t iso9660 $disk /mnt
      RETVAL=$?
      if [ 0 -eq $RETVAL ] ; then
           if [ -e /mnt/epilog.sh ]; then
               echo -n "Completing contextualization"
               . /mnt/epilog.sh
               RETVAL=$?
               if [ 0 -eq $RETVAL ]; then
                    echo " DONE"
               else
                    echo " FAILED"
               fi
               umount /mnt
               return $RETVAL
           fi
      else
           umount /mnt
           return $RETVAL
      fi
}
stop() {
      echo -n $"Unmouting context disk: "
      umount /mnt
      RETVAL=$?
      echo
      return $RETVAL
}
case "$1" in
      start)
#                   [ $running -eq 0 ] && exit 0
                    start
                    RETVAL=$?
                    ;;
      stop)
#                   [ $running -eq 0 ] || exit 0
                    stop
                    RETVAL=$?
                    ;;
           *)
           echo $"Usage: $0 {start|stop}"
           RETVAL=2
esac
exit $RETVAL
```

These scripts try to mount the contextualization CDrom and then run scripts that are in the disk, namely prolog.sh and epilog.sh .

The VM template in OpenNebula defines what is in the context disk using the following:.

```
CONTEXT = [
vmid        = "$VMID",
#TTL         = "12",
TTL         = "2",
AFS         = "off",
files       = "/opt/vmimage/init.sh /opt/vmimage/etchosts /opt/vmimage/
etcsysconfigifcfg /opt/vmimage/id_rsa.pub /opt/vmimage/lsfcontext.c
onf /opt/vmimage/etcsysconfignetwork",
target      = "vdb"
]
```