KIWI Project

# OpenSuSE - KIWI Image System System Design

Project, Design and Implementation
by Marcus Schaefer (ms@suse.de)
*

Author: Marcus Schaefer
Datum: March 2, 2007
Revision: 2.7

*SUSE LINUX Products GmbH, Maxfeldstrasse 5, D-90409 Nuremberg

# Contents

# 1 Introduction

The OpenSuSE KIWI Image System provides a complete operating system image solution for Linux supported hardware platforms as well as for virtualisation systems like Xen. The KIWI architecture was designed as a two level system. The first stage, based on a valid **software package source**, creates a so called **physical extend** according to the provided image description. The second stage creates from a required physical extend an operating system image. The result of the second stage is called a **logical extend** or short an image.

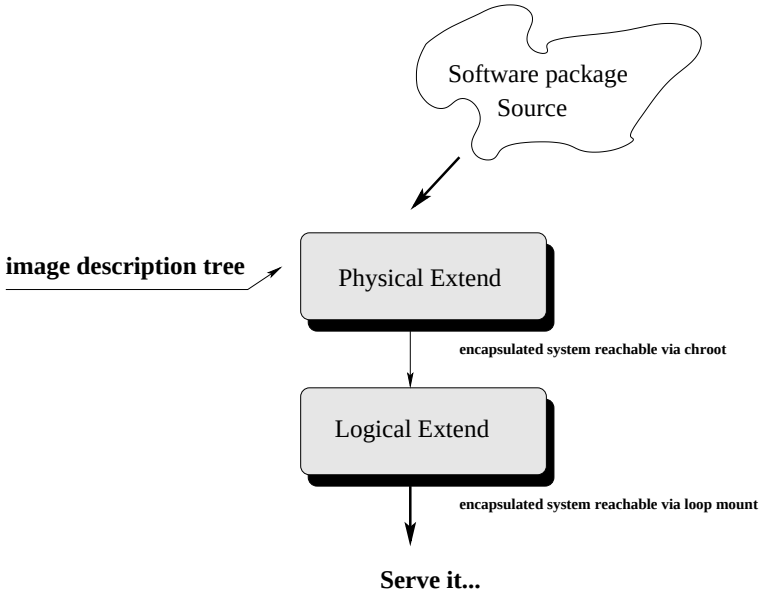


Figure 1.1: Image Serving Architecture

Because this document contains conceptual information about an image system, it is important to understand what an operating system image is all about. A normal installation process is starting from a given installation source and installs single pieces of software until the system is complete. During this process there may be manual user intervention required. However an operating system image represents an already completed *installation* encapsulated as a file and optionally includes the configuration for a specific task. Such an operating system starts working as soon as the image has been brought to a system storage device no matter if this is a volatile or non volatile storage. The process of creating an image takes place without user interaction. This means all requirements of the encapsulated system has to be fulfilled before the image is created. According to this the so called **image description tree** stores all the information needed to create an image.

## 1.1 What is KIWI good for

The solution introduced in this document covers an implementation for the following two major topics:

- How to create physical and logical extends
- How to serve/activate a logical extend, an image

As already said a physical extend requires a valid software package source. This is the location where KIWI can access the software to build up a system. Software exists as packages and packages exists in different formats. The amount of all packages are organized in a package tree including some meta data, this is called a repository. With the KIWI project I want to be free of choice what kind of package repository should be used. To achieve this goal this project will make use of the **smart** package manager which is very fast and can handle the most important repository structures. Information on smart can be found here:

- http://labix.org/smart

### 1.1.1 Supported image types

To create a logical extend it is necessary to create a filesystem the operating system data can be stored in. The image type corresponds to the selected filesystem. Supported image types are:

- **CPIO**
- **EXT2 / EXT3**
- **ReiserFS**
- **CramFS**
  for read-only part of **split** images only
- **Special cpio/ext2 Boot image**
  for image installation via network
- **Special cpio/ext2 XEN-Boot image**
  based on Xen kernel and mboot.c32 for multiboot of xen plus initrd (kiwi boot image).
- **Special cpio/ext2 ISO-Boot image**
  to create a kiwi boot image for booting a live CD system image. Used in combination with the **iso:** image type

- **Special cpio/ext2 USB-Boot image**
  to create a kiwi boot image for booting a USB stick system image. Used in combination with the **usb:** image type

- **Special cpio/ext2 VMX-Boot image**
  to create a kiwi boot image for booting a virtual machine system image. This is **not** used for paravirtualized machines like Xen but for full virtual machines like QEMU and VMware. Used in combination with the **vmx:** image type

- **iso:<boot-image-name> image support**
  LiveCD/DVD system image, which means the image gets splitted into two ext2 based sub-images whereas one represents the loop read-only part on CD and the other represents the read-write data in RAM

- **usb:<fstype>:<boot-image-name>**
  USB stick system image, which means the system image and the boot image will be created automatically. To deploy the image on the stick the –bootstick option can be used

- **vmx:<fstype>:<boot-image-name>**
  Virtual machine system image, which means a virtual disk will be created including boot manager boot image and system image to be used within qemu (.qemu file) or vmware player (.vmdk file)

- **split:<fstypeRW,fstypeRO> image support**
  Which means the image is splitted into a read-only and a read-write part whereas each part can have a different filesystem. To be able to boot such an image the config.<MAC> file must contain a *COMBINED_IMAGE=yes* statement.

# 2 Creating Operating System Images

## Contents



The creation of operating system images is based on image description trees. An image description contains all the files in a directory that are required to generate an image using the Perl-based image builder **kiwi.pl**. The directory to which the description files are written must contain a config.xml file with a three-part version number of the format:

**Major.Minor.Release**

- For smaller image modifications that do not add or remove any new packages, only the release number is incremented. The **config.xml** file remains unchanged.
- For image changes that involve the addition or removal of packages the minor number is incremented and the release number is reset.
- For image changes that change the size of the image file the major number is incremented.

## 2.1 Structure of the Image Description Tree

- **root**
  Subdirectory that contains special files, directories, and scripts for adapting the image environment **after** the installation of all the image packages. The entire directory is copied into the root of the image tree using *cp -a*.
- **config**
  Optional Subdirectory that contains Bash scripts that are called after the installation of all the image packages, primarily in order to remove the parts of a package that are not needed for the operating system. The

name of the Bash script must resemble the package name listed in the config.xml

- **config.sh**
  Optional configuration script while creating the physical extend. This script is called at the end of the installation but **before** the package scripts have run. It is designed to configure the image system, such as the activation or deactivation of certain services (insserv). The call is not made until after the switch to the image has been made with **chroot**.

- **images.sh**
  Optional configuration script while creating the logical extend. This script is called at the beginning of the image creation process. It is designed to clean-up the image system. Affected are all the programs and files only needed while the physical extend exists.

- **config-yast.xml**
  Optional configuration file which has been created by autoyast. To be able to create such an autoyast profile you should first call:

```
yast2 autoyast
```

  Once you have saved the information from the autoyast UI as config-yast.xml file in your image description directory kiwi will process on the file and setup your image as follows:

  1. While booting the image YaST is started in autoyast mode automatically
  2. The autoyast description is parsed and the instructions are handled by YaST. In other words the **system configuration** is performed
  3. If the process finished successfully the environment is cleaned and autoyast won't be called at next reboot.

- **config.xml**
  Configuration file that indicates the image type, base name, options, and which packages make up the image. The structure of the file corresponds to the format

```
<image name="Name" inherit="optional-path">
    <preferences>
        <type>Type</type>
        <version>1.2.3</version>                    <size
unit="Unit">Size</size>
        <packagemanager>name</packagemanager>
        <compressed>Yes/No</compressed>
    </preferences>
    <users group="groupname">
        <user name="user" pwd="word" home="dir"/>
    </users>
    <drivers type="Type">
        <file name="Filename"/>
    </drivers>
    <repository type="Type">
        <source path="URL"/>
    </repository>
    <packages type="Type">
        <package                    name="Packagename"
arch="Architecture"/>
        <opensusePattern name="Patternname"/>
        <ignore name="Packagename"/>
    </packages>
</image>
```

The config.xml file contains four major parts embedded into a XML image tag. The image tag contains the attribute **name** to indicate the base name of the image. It is automatically expanded using the version number and the date. The version number is extracted from the directory in which the description files for this image are located. Additionally one can add the attribute **inherit** followed by a path to another kiwi description. The system will then inherit the package information and prepend it to the current description.

1. The **preferences** tag contains information needed to create the logical extend. For this tag the following subtags are defined:
   - **packagemanager**
     Name of the packagemanager to be used for installing packages. Currently smart is the default packagemanager but zypper will be supported as well

– **size**
Image size as a number whereas the attribute **unit** defines the scale unit **M** or **G** standing for Megabyte or Gigabyte. Note: *kiwi* supports the feature extending the image size automatically to a calculated size, if the specified config size value is to small. If the config size value plus the additional size needed to build the image is more than 100MB, *kiwi* will abort with an error message. Furthermore *kiwi* will not reduce the image size automatically by design, because it must be possible to configure additional space, for example, if custom scripts are run.

– **type**
The image type of the logical extend:

* ext2,ext3 or reiserfs
can be used as system image filesystem
* cpio
for boot images only
* iso:filename
to create an .iso image whereas filename must be the name of an appropriate isoboot boot image from */usr/share/kiwi/image/isoboot*
* usb:type:filename
to create a system and boot image suitable to run on an USB stick deployed with the kiwi –bootstick option. type specifies one of ext2,ext3 or reiserfs and filename must be the name of an appropriate usbboot boot image
* vmx:type:filename
to create virtual disk images for qemu and VMware. type specifies one of ext2,ext3 or reiserfs and filename must be the name of an appropriate vmxboot boot image
* split:type-rw,type-ro
to create a splitted image into a read-only and a read-write part. type-rw specifies one of ext2,ext3 or reiserfs type-ro specifies one of ext2,ext3,reiserfs or cramfs

– **timezone**
The time zone. The possible time zones are located in the directory */usr/share/zoneinfo*. For the image itself, only one time zone each is required. For this reason, the relative path to the time zone to use in the image is indicated after the **timezone** key, for example, **Europe/Berlin**. kiwi uses this information to extract the corresponding time zone from the timezone package and to store it as */etc/localtime* in the image.

– **keytable**
Contains the name of the console keymap to use. The name corresponds to a map file stored below the path */usr/share/kbd/keymaps*. Furthermore, the variable *KEYTABLE* within the file */etc/sysconfig/keyboard*

will be set according to the keyboard mapping.

2. The optional **users** tag contains user information used to add users to the image. The attribute **group** specifies the group the user(s) belongs to. If this group doesn't exist it will be created. Within *users* each user is described in one **user** tag including the attributes **name, pwd and home** which describes the name of the user it password and its home directory.

3. The optional **drivers** tag contains driver file names. The names are interpreted as general driver name and captured if they are contained in the kernel tree. The attribute **type** specifies one of the following driver types:

   – **netdrivers**
   Every file is indicated relative to the directory */lib/modules/<Version>/kernel/drivers/net*

   – **drivers**
   Every file is indicated relative to the directory */lib/modules/<Version>/kernel*

4. The **repository** tag defines the source path and type used by the package manager. The attribute **type** specifies the type of the repository, for example, **type="yast2"** and the subtag **source** contains the attribute **path** to setup the the location of the repository, for example, **source="/image/CDs/full-i386"**. The path specifaction can be done as:

   – local path starting with /
   – **html://** or **ftp://** Network-Location
   – opensuse://Project-Name
   – The path can include the %arch macro if needed

   Multiple repository tags are allowed. For information on how to setup a smart source refer to *http://labix.org/smart*

5. The **packages** tag contains a list of package and/or pattern names whereas the attribute **type** specifies under which cirumstances this package set needs to be used. There are three different types of package sets:

   – **image**
   used to finish the image installation. All packages which makes up the image are listed there.

   – **boot**
   used to start buildig the image. Basic components like libc or the smart package manager are listed here.

   – **xen**
   used when the image needs support for Xen based virtualisation. Option **–virtual xen**

Using a pattern name will enhance the package list to a number of additional packages belonging to this pattern. Support for patterns is SuSE specific and available with openSuSE v10.2 or higher. If a pattern adds something you don't want to have in your image it is possible to specify an **ignore** tag together with the **name** of the package or package alias as tag attribute. If there is a package a pattern you want to use for a specific architecture only it is possible to specify an optional attribute named **arch** followed by a comma seperated list of allowed architectures.

- **cdboot**
  A directory containing all information required for the isolinux boot loader. This includes the **isolinux.cfg** configuration file, the **isolinux** directory which contains pictures, messages and translation files for the first boot screen while booting from a CD. In addition the **isolinux.sh** build script needs to be provided which creates an ISO image from a prebuild CD tree requiring the mentioned isolinux specification. According to this, the cdboot directory only makes sense for a CD boot image. For more infomation about creating a bootable CD, refer to chapter **??**.

In addition all values entered as **image** tag of the config.xml file are stored in a file called **.profile**. The file is created before the execution of an image script like *config.sh, images.sh* or a *package script* and can then be sourced. The parameters of the config file are then available as variables in the script and can be processed appropriately. The script itself is called within the image environment, which means it is not possible to damage the host system with your script even if you are using absolute paths. Such a script should look like the following template:

```
#!/bin/sh
echo -n "Image [<name>]..."
test -f /.profile && . /.profile

... script code

echo done
```

The parameter **name** should be the name of the image to which this script belongs.

# 3 Activating an image

## Contents



After a logical extend (an image) has been created from a physical extend there are in principal four possibilities to activate the image:

1. In case of a **local accessable system harddisk**, the image can be installed by dumping (dd) the image file on a previously created partition on this disk. To activate the system a boot manager like grub or lilo can be used

2. In case of a **network enabled system** (netboot client) the image can be installed via a special boot image. The boot image which serves as initial ramdisk (initrd) and the appropriate kernel are downloaded from a network service. The linux kernel automatically calls a program named **linuxrc** which takes over all tasks needed to download and install the system image. The installation can be done persistently on disk or temporary into the RAM of the machine.

3. In case of a **para virtualized target system** like Xen, the image can be be installed by copying the image file on the target system. To activate the virtual system a configuration must be provided which points to the image in some way. One possibility is to use a loop mounted location.

4. In case of a **full virtualized target system** like VMware, or QEMU the image represents a virtual disk as file which can be *played* by the virtualisation system.

## 3.1 The KIWI netboot image

The KIWI netboot image can be used to install an operating system image to a network client. To establish communication with the client a boot server infrastructure including the following services is required:

- DHCP server to give the client an IP address
- TFTP server to allow file transfer from/to the client

### 3.1.1 The Boot Process of a netboot System

To understand how to use the operating system image with the netboot image, a short description of the netboot client is useful. The following diagram shows the simplified boot process of a netboot client.

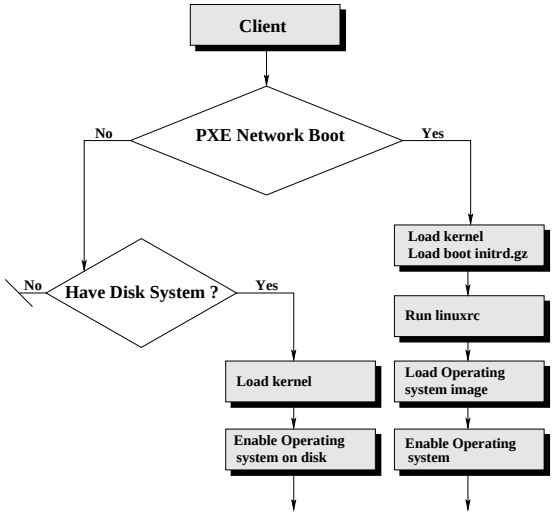


Figure 3.1: The Boot Process of a netboot client

The most important point is to understand the cooperation between the initial boot image **initrd.gz** and the intrinsic operating system image. If the system is able to boot via a network, it will load the kernel and the compressed boot image from the network. The *brain* of the boot image is the **linuxrc** script, which does all the stuff controlled by an image configuration file also obtained from the network. The major task is to download and activate the operating system image. The boot image is exchanged for the operating system image to be activated. The following overview describes the steps that take place when the netboot client is booted:

- Via PXE network boot or boot manager (GRUB), the client boots the initrd (initrd.gz) that it receives from the TFTP server. If no PXE boot is possible, the client tries to boot from the hard disk, if accessible.

- Running **linuxrc** starts the process described below.

1. The required file systems to receive system data are mounted. Example: **proc** file system.

2. Network support is activated. A default list of modules is used and are tried one after the other. The module is loaded using *modprobe*. Any dependencies to other modules are cleared at that time.

3. The network interface is set up via DHCP. After the interface has been established, the DHCP variables are exported into the file */var/lib/dhcpcd/dhcpcd-eth0.info* and the contents of DOMAIN and DNS are used to generate a */etc/resolv.conf*.

4. The TFTP server address is acquired. During this step, a check is first made to determine whether the host name **tftp.$DOMAIN** can be resolved. If not, the DHCP server is used as the TFTP server. For more infomation about the TFTP servers structure, refer to Section 3.1.2

5. The configuration file is loaded from the server directory */var/lib/tftpboot/KIWI* via TFTP. At this point, the client expects the file:

   config.<MAC Address>

   If this file is not available and cannot be loaded, it means this is a new client that can be immediately registered. A new client is registered by uploading a control file to the TFTP servers upload directory: */var/lib/tftpboot/upload*. After the upload, the client branches off into a loop in which the following steps are taken:

   - the DHCP lease file is renewed (*dhcpcd -n*).
   - a new attempt is made to load the file config.<MAC address> from the TFTP server.
   - if the file does not exist, there is a 60-second wait period before a new run begins.

   If the configuration file does load, it contains data on image, configuration, synchronization, or partition parameters. For more infomation about the file format of the configuration file, refer to Section 3.1.3.

6. The PART: line in the configuration is analyzed. If there is a PART line in the configuration file, the following analysis takes place:

   - A check is made using the image version to see whether any local system needs to be updated. If not, local boot process continues immediately. No image download occurs.

- No system or update required: client hard disk is partitioned.

7. Indicated images are downloaded with multicast TFTP.
8. Checksums checked. Repeat download if necessary.
9. The CONF: line is evaluated. All the indicated files are loaded from the TFTP server and stored in a */config/* path.
10. Terminate all the user-land processes based on the boot image (dhcpcd -k).
11. operating system image is mounted.
12. The configuration files stored in */config/...* are copied into the mounted operating system image.
13. The system switches to the mounted operating system image. The root file system is converted to the operating system image via **pivot_root**. All the required configuration files are now present, because they had been stored in the operating system image or have been downloaded via TFTP.
14. The boot image is unmounted using an **exec umount** call.
15. At termination of linuxrc or the exec call, the kernel initiates the **init** process that starts processing the boot scripts as specified **/etc/inittab**, for example, to configure the network interface.

### 3.1.2 TFTP Server Structure

The TFTP server directory structure is divided into the following main areas

- **Image configurations**
  The */var/lib/tftpboot/KIWI/* directory contains the various *config.<MAC Address>* image configuration files.
- **Configuration files**
  The */var/lib/tftpboot/KIWI/<MAC Address>/* directory contains the various system configuration files, such as xorg.conf.
- **Boot files**
  The */var/lib/tftpboot/boot/* directory is where the initrd.gz, and the kernel to boot are kept.
- **PXE second stage boot loader(s)**
  The */var/lib/tftpboot/* directory is where the boot loaders for PXE are kept (pxelinux.0,mboot.c32)

- **PXE configuration file**
  The */var/lib/tftpboot/pxelinux.cfg* directory is where the PXE configuration file is kept.

- **Image files and checksums**
  The */var/lib/tftpboot/image/* directory is where all the image files and their checksums are kept.

- **Upload area**
  The directory */var/lib/tftpboot/upload/* is the directory into which the *hwtype.<MAC Address>* files for registering new netboot clients are uploaded.

### 3.1.3 The netboot client Configuration File

This section describes the netboot client configuration file:

config.<MAC Address>

The configuration file contains data about image, configuration, synchronization, or partition parameters. The configuration file is loaded from the TFTP server directory */var/lib/tftpboot/KIWI* via TFTP for previously installed netboot clients. New netboot clients are immediately registered and a new configuration file with the corresponding MAC address is created. Below, find an example of a cash register configuration file:

```
IMAGE=/dev/hda2;image/browser;1.1.1;192.168.1.1;4096
CONF=/KIWI/00:30:05:1D:75:D2/ntp.conf;/etc/ntp.conf;192.168.1.1;1024,      \
     /KIWI/00:30:05:1D:75:D2/xorg.xonf;/etc/X11/xorg.xonf;192.168.1.1;1024
PART=200;S;x,300;L;/,500;L;/opt,x;L;/home
DISK=/dev/hda
```

The following format is used:

**IMAGE**=device;name;version;srvip;bsize;compressed,...,
**SYNC**=syncfilename;srvip;bsize
**CONF**=src;dest;srvip;bsize,..., src;dest;srvip;bsize
**PART**=size;id;Mount,...,size;id;Mount
**JOURNAL**=ext3
**DISK**=device

- **IMAGE**
  Specifies which image (name) should be loaded with which version (version) and to which storage device (device) it should be linked, e.g., **/dev/ram1** or **/dev/hda2**. The netboot client partition (device) **hda2** defines the root file system "/" and `hda1` is used for the swap partition. The numbering of the hard disk device should not be confused with the RAM disk device, where `/dev/ram0` is used for the initial RAM disk and can not be used as storage device for the second stage system image. SUSE recommends to use the device `/dev/ram1` for the RAM disk. If the hard drive is used, a corresponding partitioning must be performed.

- **srvip**
  Specifies the server IP address for the TFTP download. Must always be indicated, except in PART.

- **bsize**
  Specifies the block size for the TFTP download. Must always be indicated, except in PART. If the block size is too small according to the maximum number of data packages (32768), **linuxrc** will automatically calculate a new blocksize for the download.

- **compressed**
  Specifies if the image file on the TFTP server is compressed and handles it accordingly. To specify a compressed image download only the keyword "**'compressed"'** needs to be added. If compressed is not specified the standard download workflow is used. **Note:** The download will fail if you specify "'compressed"' and the image isn't compressed. It will also fail if you don't specify "'compressed"' but the image is compressed. The name of the compressed image has to contain the suffix **.gz** and needs to be compressed with the **gzip** tool. Using a compressed image will automatically **deactivate** the multicast download option of atftp.

- **CONF**
  Specifies a comma-separated list of source:target configuration files. The source (src) corresponds to the path on the TFTP server and is loaded via TFTP. The download is made to the file on the netboot client indicated by the target (dest).

- **PART**
  Specifies the partitioning data. The comma-separated list must contain the size (size), the type number (id), and the mount point (Mount). The size is measured in MB by default. Additionally all size specifications supported by the sfdisk program are allowed as well. The type number specifies the ID of the partition. Valid ID's are listed via the *sfdisk – list-types* command. The mount specifies the directory the partition is mounted to.

  – The first element of the list must define the swap partition.

  - The second element of the list must define the **root** partition.
  - The swap partition must not contain a mount point. A lowercase letter **x** must be set instead.
  - If a partition should take all the space left on a disk one can set a lower **x** letter as size specification.

- **DISK**
  Specifies the hard disk. Used only with PART and defines the device via which the hard disk can be addressed, e.g., **/dev/hda**.

- **RELOAD_IMAGE**
  If set to a non-empty string, forces the configured image to be loaded from the server even if the image on the disk is up-to-date. Used mainly for debugging purposes, this option only makes sense on diskful systems.

- **RELOAD_CONFIG**
  If set to an non-empty string, forces all config files to be loaded from the server. Used mainly for debugging purposes, this option only makes sense on diskful systems.

- **COMBINED_IMAGE**
  If set to an non-empty string, indicates that the both image specified needs to be combined into one bootable image, whereas the first image defines the read-write part and the second image defines the read-only part.

### 3.1.4 The netboot client Control File

This section describes the netboot client control file:

```
hwtype.<MAC Address>
```

The control file is primarily used to set up new netboot clients. In this case, there is no configuration file corresponding to the client MAC address available. Using the MAC address information, the control file is created, which is uploaded to the TFTP servers upload directory */var/lib/tftpboot/upload*.

## 3.2 The Xen virtual machine

To use an image within Xen the most important point is to use the para virtualized xen kernel within the operating system image. While creating a xen based image **kiwi** will take care for the correct kernel to be used.

### 3.2.1 Activate a KIWI xen image

To activate an image including the xen kernel the following steps needs to be performed:

1. loop mount the image

   ```
   mount -o loop <image> <loop path>
   ```

2. create a xen configuration file and use the loop mounted path as disk parameter:

   ```
   disk = [ 'phy:<loop path>' ]
   ```

   for more information on how to configure xen refer to http://en.opensuse.org/Installing_Xen3

3. boot the system using the command:

   ```
   xm create -c <xen config file>
   ```

## 3.3 The VMware virtual machine

VMware represents a full virtualized machine which means all components are virtualized. This includes the storage devices as well as the processor and all other parts of the system. An image for such a system is in principal a virtual disk which contains the boot manager the initrd and all other system data as one file. Kiwi is able to create such a disk appropriate for use in QEMU and VMware (vmdk format).

### 3.3.1 Activate a KIWI virtual disk system

To activate such a system the user only needs to call the correct *player* application which in case of QEMU is **qemu** and in case of VMware **vmplayer** is used.

# 4 Deployment of Operating System Images

## Contents



Creating an operating system image always implies the question how to activate the image on the target system. As there are many possible targets the so called image deployment higly depends on the system environment. If a customer buys a product for example SLEPOS the system environment is given and it is required to take care for the rules to successfully deploy the image. On the one hand this makes it easy for customers to control a complex system but on the other hand one will loose the flexibility to use the same system in another environment.

kiwi doesn't stick to a specific system environment but therefore it leaves out important tasks concerning the deployment architecture. So to say kiwi is an image creator but it doesn't setup the deployment infrastructure. This chapter provides information about the different possibilities to deploy an image and how kiwi fits into this picture.

## 4.1 Via network using the PXE protocol

PXE is a boot protocol mostly implemented in todays BIOS's or boot ROMs of some network cards. If activated it broadcasts the network for a DHCP to obtain an IP address and the information where to find a TFTP server to manage file transfers. If such a server exists the second stage bootloader controls the subsequently boot process. Using PXE with kiwi requires the infrastructure explained in section 3.1.2 and a TFTP server as well as a DHCP server run-

ning. kiwi provides the package **kiwi-pxeboot** which setup the boot structure and installs some prebuild boot images.

If the plan is to use a system image for which no prebuild boot image exist, it is need to create a custom boot image before a system image could be deployed. Boot images consists of the image itself and the appropriate kernel to that image. Both file must be stored in the directory */var/lib/tftpboot/boot*. Assuming there is no prebuild boot image for the SuSE Linux 10.2 distribution the steps to create it are as follows:

```
kiwi –prepare netboot-suse-10.2 –root /tmp/myroot
kiwi –create /tmp/myroot -d /tmp
```

The result of this example is created in the /tmp directory:

```
ls -1 /tmp/initrd-netboot*
   /tmp/initrd-netboot-suse-10.2.i686-2.1.1.gz
   /tmp/initrd-netboot-suse-10.2.i686-2.1.1.kernel.2.6.18.2-31-default
```

The next step is to make the boot image known to the TFTP server. To do this the files must be copied to the */var/lib/tftpboot/boot* directory. Optionally they can be renamed. For the following example the boot image is renamed to **initrd** and the boot kernel is renamed to **linux**

```
cp initrd-netboot.i686-2.1.1.gz \
        /var/lib/tftpboot/boot/initrd
cp initrd-netboot.i686-2.1.1.kernel.2.6.18.2-31-default \
        /var/lib/tftpboot/boot/linux
```

Switching on the target machine, which needs to run PXE by default, will load the linux kernel and the kiwi created initrd. The initrd will register the machine if there is no configuration found in */var/lib/tftpboot/KIWI*. Registration means a file including the MAC address of the machine is uploaded into the directory */var/lib/tftpboot/upload*. For information about creating the configuration refer to section 3.1.3. The following example configuration fits for a machine including a disk on /dev/sda:

```
IMAGE=/dev/sda2;full-suse-10.2.i686;1.1.2;192.168.100.2;4096
PART=1024;S;x,x;L;/
DISK=/dev/sda
```

According to this configuration the kiwi boot image will try to download a system image named **full-suse-10.2.i686-1.1.2** from the TFTP server with IP

address 192.168.100.2. On the TFTP server the system images are stored in */var/lib/tftpboot/image*. The boot image will prepare the disk and create a 1GB swap partition and another full size linux partition. The process of creating this full-suse-10.2 image can be done by using kiwi. Once the boot image has successfully downloaded the system image it is getting activated and operates as configured.

### 4.1.1 Via TFTP with initial boot from CD or USB stick

Sometimes it is not possible to use PXE as boot protocol. This could happen if the target machine doesn't support PXE or the installed network card doesn't provide a PXE boot rom. Preconditioned your network provides a DHCP and a TFTP server the problem can be solved by storing the boot image on another bootable medium like a CD or an USB stick. Referring to the information from the section above it is easy to create a bootable CD / USB stick with kiwi:

```
kiwi –bootcd /tmp/initrd-netboot-suse-10.2.i686-2.1.1.gz
```

The command will create a bootable ISO image which only needs to be burned on a CD. The following file will be created after the call: */tmp/initrd-netboot-suse-10.2.i686-2.1.1.cdboot.iso*

```
-plug in an USB stick then call-
kiwi –bootstick /tmp/initrd-netboot-suse-10.2.i686-2.1.1.gz
```

The command will create a bootable USB stick. kiwi is searching for the USB stick plugged in before and lists all devices found. The user needs to select one of the devices by typing one of the suggested device names. It is important to be careful at that stage because all data on the given device will be lost. Because of this reason the user has to type the device name which hopefully increases the chance not to do something thoughtlessly.

## 4.2 Split image system via PXE

kiwi supports system images to be splitted into two parts, a read-only part and a read-write part. This allows to put data on different filesystems which is mostly used to have read-only data available on a compressed filesystem like cramfs or squasfs. Please note that almost all compressed filesystems available have some kind of restrictions which needs attention before starting to use it in an image.

To turn a system image into a split image only the type of the image must be adapted. This information is part of the **config.xml** file and could be changed

like the following example shows:

```
<preferences>
    <type>split:ext3,cramfs</type>
    ...
</preferences>
```

Creating an image from this description results in two image files whereas one of them will contain the **-read-only** extension in its name. Ay one may imagine booting such an image always requires a boot process which must be able to bring both images together again. Because of this, split images can only deployed in combination with one of the kiwi boot images.

To deploy the image only PXE or a boot CD / USB-stick can be used. The most important part while making use of split images is the configuration for the target machine. More information on this config.MAC file can be found in section 3.1.3. In case of a split image the following information must be provided:

- The IMAGE key must contain both images the read-write and the read-only image. The read-write image must appear as first entry in the list
- The PART key has to specifiy a partition table with at least three partitions. A swap partition and two system partitions which provides enough space for the first and the second image portion
- The Option COMBINED_IMAGE to tell the boot image to combine both images into one entire system

The following example shows the configuration of a split image named minimal-10.1 / minimal-10.1-read-only

```
IMAGE=/dev/sda2;minimal-10.1.i686;1.1.2;192.168.100.2;4096,\
      /dev/sda3;minimal-10.1-read-only.i686;1.1.2;192.168.100.2;4096
PART=200;S;x,500;L;/,x;L;
DISK=/dev/sda
COMBINED_IMAGE=yes
```

## 4.3 USB stick system

A very popular method is storing complete operating systems on an USB stick. This means not only the boot image and the kernel is stored on the stick but also the entire system image is part of the stick. To be able to do this kiwi provides special boot image(s) located in **/usr/share/kiwi/image/usbboot**. Currently it is tested and available for SuSE Linux 10.2 only but can be adapted

to other versions as well. To use your system image on a boot stick setup the system's config.xml as follows:

```
<preferences>
    <type>usb:ext3:usbboot/suse-10.2</type>
    ...
</preferences>
```

After this the boot and system image needs to be created. Concerning to the size of the stick the image is not allowed to grow beyond it. Assuming the system image is named full-suse-10.2 the command is as follows:

```
kiwi –prepare full-suse-10.2 –root /tmp/mysystem
kiwi –create /tmp/mysystem -d /tmp
```

The result of the command above is represented in the image file */tmp/full-suse-10.2.i686-1.1.2*. The usbboot image is created automatically and is stored as */tmp/initrd-usbboot-suse-10.2.i686-2.1.1.gz*. To create a bootable stick with a SuSE Linux 10.2 operating system on it, plug in the stick and call:

```
-plug in an USB stick then call-
kiwi –bootstick /tmp/initrd-usbboot-suse-10.2.i686-2.1.1.gz \
    –bootstick-system /tmp/full-suse-10.2.i686-1.1.2
```

## 4.4 Virtual disk system (QEMU or VMware)

To be able to use a virtualization system a virtual disk needs to be created. This can be done by specifying the following type in the system's image config.xml:

```
<preferences>
    <type>vmx:ext3:vmxboot-suse-10.2</type>
    ...
</preferences>
```

After this the system image can be created. The process will create the system image and the specified vmxboot-suse-10.2 boot image. The result is then used to create the virtual disk. The following command needs to be called:

```
kiwi –prepare full-suse-10.2 –root /tmp/mysystem
kiwi –create /tmp/mysystem -d /tmp
```

The result of the command above is a set of virtual disks, one with suffix .qemu and the other with the suffix .vmdk (VMware). To run the system on the virtual disk with for example qemu call:

```
qemu /tmp/full-suse-10.2.i686-1.1.2.qemu
```

## 4.5 Live CD system

Normally an image will be installed on a disk or into the main memory of a computer. This is done by a deployment architecture which transfers the image via a boot image into its final destination. such a boot image can also exist on a CD. The task of the CD-Boot image is to setup a system which is divided into two parts:

- Read-Only part which is obtained from the CD
- Read/Write part which is pushed into main memory

An image which works partially on disk and partially on RAM is called a live CD system. The CD-Boot structure of KIWI will put the directories */bin, /boot, /lib, /opt, /sbin and /usr* on the CD and the rest into the main memory of the system.

### 4.5.1 How do you setup an image as live CD

To create an .iso image which can be burned on CD you only need to specify the boot image which should handle your live system. This is done by setting the **type** of the image in the **config.xml** file as follows:

```
<preferences>
    <type>iso:name</type>
    ...
</preferences>
```

The parameter **name** refers to the CD boot image which must exist in */usr/share/kiwi/image/is*
Like for all boot images the most important point is that the boot image has to match the operating system image. This means the kernel of the boot- and

operating system image must be the same. If there is no boot image which matches you need to create your own boot image description. A good starting point for this is to use an existing boot image and adapt it to your needs.

### 4.5.2 How does the boot image work

The boot process from a CD works in five steps:

1. After the image has been prepared which means a physical extend exists the process of building the logical extend will split the physical part into two physical extends:
   - read/write extend which is loaded into RAM
   - read-only extend consisting of the data stored in the */bin, /boot, /lib, /opt, /sbin and /usr* directories
2. From the two physical extends kiwi will create two ext2 based logical images.
3. Next kiwi prepares and creates in one step the boot image given as the type of the image configuration file config.xml.
4. Next kiwi will setup the CD directory structure and copy all image and kernel files as well as the isolinux data into this directory tree.
5. At last kiwi calls the **isolinux.sh** script provided by the ISO boot image to create an .iso file from the CD directory tree.

# 5 Troubleshooting

# Index