innotek VirtualBox

Open-source virtualization for all purposes: desktop, server, high-security
On the menu:

- Very brief feature overview
- Differences compared to other virtualization solutions
- VirtualBox processes
- COM/XPCOM introduction
- The VirtualBox COM/XPCOM API
Running Vista on Linux...
... or Linux on Vista
Full software virtualization: Use *software* to virtualize (no hardware support required); run *unmodified* operating systems in the virtual machines.
Full virtualization
Comparison

- Software complexity: high
- Installation effort: low
- I/O speed: fast
- Maturity: stable
- Hardware virtualization: supported, but not required
- Summary: comparable to closed-source competitors, but open-source
VirtualBox characteristics

- Flexibility
- Modularity
- Portability
1. Flexibility

- Easy-to-use graphical user interface: VirtualBox, a Qt UI
1. Flexibility

- Easy-to-use graphical user interface: VirtualBox, a Qt UI
- Full control via command-line interface: VBoxManage
1. Flexibility

- Easy-to-use graphical user interface: VirtualBox, a Qt UI
- Full control via command-line interface: VBoxManage
- Remote execution with headless VRDP server: VBoxVRDP (shown later)
1. Flexibility

- Easy-to-use graphical user interface: VirtualBox, a Qt UI
- Full control via command-line interface: VboxManage
- Remote execution with headless VRDP server: VBoxVRDP (shown later)

Soon: web service
2. Modularity

- VirtualBox core: properly encapsulated in XPCOM API

[XPCOM API]

- VirtualBox core processes
- Graphical UI
- VBoxManage
- VBoxVRDP
- Web service
2. Modularity

- VirtualBox core: properly encapsulated in XPCOM API

Diagram:
- XPCOM API
- VirtualBox core processes
  - Graphical UI
  - VBoxManage
  - VBoxVRDP
  - Web service
  - Gnome?
3. Portability

• Host support:
  - Windows 2000, XP, Vista
  - Linux
    (all major distributions: Ubuntu, Debian, Red Hat, Suse, Gentoo, ...)
  - New with 1.4:
    Mac OS X + Linux 64-bit hosts
3. Portability

Platform encapsulation:

• GUI is written with Qt (hence Windows, Linux, Mac support)
• Glue code for COM: native COM on Windows, XPCOM elsewhere
• Ring-0 driver is split in platform-dependent and platform-independent code
• Ring-3 code uses custom “innotek Portable Runtime Library” for compiler independence
Usage scenarios

1. Operating system support
2. Software testing
3. Server consolidation
4. High-security environments
5. Desktop consolidation
Features

- Snapshots: save and restore the state of a virtual machine (for example after software installation or virus infection)
- Shared folders
- Remote Desktop Protocol (VRDP) server
- USB over VRDP
Features

Storage: disk image files (VDI, VMDK), iSCSI targets, raw disk support (1.4)
Network: NAT or host networking (TUN/TAP), PXE supported
Remote Desktops

- Thin-client infrastructure
- Promoted since 1996, but largely unsuccessful
- Employees want rich clients
innotek RDP server is built into the virtual machine

can show all VM data, including text mode
Remote Desktops

Data center

VM

Thin client with rich operating system

RDP
Remote Desktops

Data center

VM

Thin client with rich operating system

USB over RDP!
Remote Desktops

- Data center
- VM
- Thin client with rich operating system
- RDP
- Automation via snapshots
- USB over RDP!
Remote Desktops

- Thin client with rich operating system
- USB over RDP!
- Automation via snapshots
- VM cloning
- VM
- Data center
Licensing

• Two editions:
  – “Binary release”, complete commercial product
  – VirtualBox Open-Source Edition (OSE)
    GPL source code; complete product except USB, shared folders & RDP
Licensing

• Two editions:
  - “Binary release”, complete commercial product
  - VirtualBox Open-Source Edition (OSE)
    GPL source code; complete product except USB, shared folders & RDP

• Download commercial version free of charge for
  - personal use
  - educational use
  - evaluation
And finally:

technical stuff.
VirtualBox processes

A bird's eye view.
VirtualBox processes

[1] VirtualBox

[2] VBoxSVC

shared object files

VBoxXPCOMIPCD (helper daemon)
VirtualBox processes

[1] VirtualBox

“start VM”

shared object files

[2] VBoxSVC

spawns process

VBoxXPCOMIPCD (helper daemon)

[3] VirtualBox -startvm b5bbec5c-4e73-4c8c-abc2-966247ef267d
VirtualBox processes

[1] VirtualBox

"start VM"

[2] VBoxSERV

[3] VirtualBox -startvm b5bbec5c-4e73-4c8c-abc2-966247ef267d

VBoxXPCOMIPCD (helper daemon)
A virtual machine process

Inspection:
A close-up view
Intel 386 ring architecture

Applications
("user space")

(normally unused)

Kernel

ring 3

ring 1

ring 0
A virtual machine process

[3] VirtualBox -startvm b5bbe5c-4e73-4c8c-abc2-966247ef267d

Host kernel

vboxdrv (ring-0 module)

ring 3

ring 1

ring 0
A virtual machine process

[3] VirtualBox -startvm b5bbec5c-4e73-4c8c-abc2-966247ef267d

Host kernel

vboxdrv (ring-0 module)

Guest OS
A virtual machine process

[3] VirtualBox -startvm b5bbec5c-4e73-4c8c-abc2-966247ef267d

executed natively like other host processes!

Guest OS

Host kernel

vboxdrv (ring-0 module)
A virtual machine process

[3] VirtualBox -startvm b5bbec5c-4e73-4c8c-abc2-966247ef267d

tricked to run at host ring 1!

Host kernel

vboxdrv (ring-0 module)
A virtual machine process

[3] VirtualBox -startvm b5bbec5c-4e73-4c8c-abc2-966247ef267d

tricked to run at host ring 1!

privileged instruction or external interrupt or guest trap

Host kernel  vboxdrv (ring-0 module)
A virtual machine process

[3] VirtualBox -startvm b5bbec5c-4e73-4c8c-abc2-966247ef267d

Guest OS

Host kernel

cleanup/patch/emulate/virtualize/resume

vboxdrv (ring-0 module)
A virtual machine process

Any privileged instruction in ring 1 causes traps. Thousands per second -> slow. Also, some nasty instructions such as CLI do not trap.

VirtualBox analyzes and patches code at runtime:
  a) before executing ring-0 code
  b) after a trap to avoid subsequent traps

CSAM: Code Scanning and Analysis Manager
PATM: Patch Manager
A virtual machine process

Traps allow for virtualizing hardware.
IDE device: Guest writes command to I/O port. Expects device to perform I/O and then raise IRQ.

VirtualBox: out instruction in guest IDE driver (ring 1) traps.
Hypervisor (VirtualBox ring 0) analyzes trap. Performs virtual I/O with hard disk image. Looks at guest IDT and jumps to guest interrupt handler.
Messy stuff.
What's easy?
The COM/XPCOM API

(a very brief introduction)
VirtualBox has a glue layer that allows it to use:

- native Component Object Model (COM) on Windows hosts
- Mozilla's XPCOM on other host platforms
What's COM?

- Components = cross-process objects
- Use objects as if they were local, irrespective of which process they live in
What's COM?

- Components:
- Use objects as if they were local, irrespective of which process they live in.

Separate interface from implementation:
- All interfaces are declared in IDL;
- Interfaces are mapped to abstract virtual methods in C++;
- All interfaces must support the QueryInterface(), AddRef() and Release() methods.
Absolutely critical: **reference counting**!
- Each new object initially has refcount of 1
- Each new user must call AddRef()
- When done, call Release()
- When refcount reaches zero, object destroys itself
• Absolutely critical: **reference counting**!
• Each new object initially has refcount of 1
• Each new user must call AddRef()
• When done, call Release()
• When refcount reaches zero, object destroys itself

In VirtualBox client code, simply use ComPtr<Interface>
Interfaces
- **Interface**: abstract definition of valid operations on an object
- Somewhat like a class, but without private implementation
- `QueryInterface()` method maps objects for different interfaces
• Interfaces are described in IDL
• Pseudo-IDL example:

```java
interface IVirtualBox {
    readonly attribute version;
    createMachine(string name);
}
```
Actually: VirtualBox uses an XML file to describe interfaces, which is converted to COM or XPCOM IDL via XSLT

```xml
<interface
    name="IVirtualBoxCallback" extends="$unknown"
    uuid="ee95fffc2-b6c6-4ce8-9e9e-ceadbb5019fe"
    wsmap="suppress">
  <method name="onMachineStateChange">
    <param name="machineId" type="uuid" dir="in">
      <desc>ID of the machine this event relates to.</desc>
    </param>
    <param name="state" type="MachineState" dir="in">
      <desc>New execution state.</desc>
    </param>
  </method>
</interface>
```
• Actually: VirtualBox uses an XML file to describe interfaces, which is converted to COM or XPCOM IDL via XSLT

```xml
<interface
    name="IVirtualBoxCallback" extends="$unknown"
    uuid="ee95ffc2-b6c6-4ce8-9e9e-ceadb5019fe"
    wsmap="suppress">
    <method name="onMachineStateChange">
        <param name="machineId" type="uuid" dir="in">
            <desc>ID of the machine this event relates to.</desc>
        </param>
        <param name="state" type="MachineState" dir="in">
            <desc>New execution state.</desc>
        </param>
    </method>
</interface>
```

```c
interface IVirtualBoxCallback : nsISupports
{
    void onMachineStateChange (in nsIDRef machineId,
                               in PRUint32 state
    );
}
```
Interesting:
VirtualBox

HOWTO 1: start a VM in C++
VBoxManage -startvm "Windows XP"

ComPtr<IVirtualBox> virtualBox;
virtualBox.createLocalObject(CLSID_VirtualBox);
session.createInprocObject(CLSID_Session)
ComPtr<IMachine> machine;
virtualBox->GetMachine(Guid(argv[0]), machine.asOutParam());
ComPtr<IProgress> progress;
virtualBox->OpenRemoteSession(session,
    uuid,
    sessionType,
    progress.asOutParam());
RTPrintf("Waiting for the remote session to open...\n");
progress->WaitForCompletion(-1);
• VBoxManage -startvm “Windows XP”

```cpp
ComPtr<IVirtualBox> virtualBox;
virtualBox.createLocalObject(CLSID_VirtualBox);
```

create a local object that represents the IVirtualBox singleton, which really lives in the VBoxSVC process.
VBoxManage -startvm “Windows XP”

```cpp
ComPtr<IVirtualBox> virtualBox;
virtualBox.createLocalObject(CLSID_VirtualBox);
session.createInprocObject(CLSID_Session);
ComPtr<IMachine> machine;
virtualBox->GetMachine(Guid(argv[0]), machine.asOutParam());
ComPtr<IProgress> progress;
virtualBox->OpenRemoteSession(session, uuid, sessionType, progress.asOutParam());
RTPrintf("Waiting for the remote session to open...

create a “session” (= transaction manager for write operations, blocks other writers; this way writes can be prevented while the VM is running)")
```
VBoxManage -startvm "Windows XP"

ComPtr<IVirtualBox> virtualBox;
virtualBox.createLocalObject(CLSID_VirtualBox);
session.createInprocObject(CLSID_Session);
ComPtr<IMachine> machine;
virtualBox->GetMachine(Guid(argv[0]), machine.asOutParam());
ComPtr<IProgress> progress;
virtualBox->OpenRemoteSession(session, uuid, sessionType, progress.asOutParam());
RPrintf("Waiting for the remote session to open...\n");
progress->WaitForCompletion(-1);

tell VBoxSVC to start a “remote session” (= another process) for the VM
Interesting:
VirtualBox

HOWTO 2: save a VM in C++
VBoxManage -controlvm "W..." savestate

```cpp
ComPtr<IVirtualBox> virtualBox;
virtualBox.createLocalObject(CLSID_VirtualBox);
session.createInprocObject(CLSID_Session);
ComPtr<IMachine> machine;
Guid uuid(argv[0]);
virtualBox->GetMachine(uuid, machine.asOutParam());
virtualBox->OpenExistingSession(session, uuid);
ComPtr<IConsole> console;
session->COMGETTER(Console)(console.asOutParam());
ComPtr<IProgress> progress;
console->SaveState(progress.asOutParam());
```
VBoxManage -controlvm "W..." savestate

much the same as previously:
create IVirtualBox object,
find machine from command line argument
• VBoxManage -controlvm "W..." savestate

    ComPtr<IVirtualBox> virtualBox;
    virtualBox.createLocalObject(CLSID_VirtualBox);
    session.createInprocObject(CLSID_Session);
    ComPtr<IMachine> machine;
    Guid uuid(argv[0]);
    virtualBox->GetMachine(uuid, machine.asOutParam());
    virtualBox->OpenExistingSession(session, uuid);

    ComPtr<IConsole> console;
    session->COMGETTER(Console)(console.asOutParam());
    console->SaveState(progress.asOutParam());

    now create a session object by “opening” the running VM window
VBoxManage -controlvm "W..." savestate

```cpp
ComPtr<IVirtualBox> virtualBox;
virtualBox.createLocalObject(CLSID_VirtualBox);
ComPtr<IMachine> machine;
Guid uuid(argv[0]);
virtualBox->GetMachine(uuid, machine.asOutParam());
virtualBox->OpenExistingSession(session, uuid);
ComPtr<IConsole> console;
session->COMGETTER(Console)(console.asOutParam());
ComPtr<IProgress> progress;
console->SaveState(progress.asOutParam());
```
• Important interfaces in VirtualBox:

IVirtualBox: out-of-process singleton
  (createMachine, createHardDisk, openRemoteSession, ...)

IHost: out-of-process singleton
  (DVDDrives, floppyDrives, USBDevices, USBDeviceFilters,
   operatingSystem, memoryAvailable, ...)

IMachine: one for each registered VM
  (name, id, memorySize, DVDDrive, FloppyDrive,
   USBController, sessionState, currentSnapshot, ...)

IConsole: mouse + keyboard + video for a running VM
  (reset, pause, powerButton, saveState, takeSnapshot, ...)

IDisplay, IMouse, IKeyboard, ...

ISnapshot: representation of a particular saved VM state
New in 1.4

- Mac OS X hosts
- Linux 64-bit hosts
- VMDK support
- Raw partition support
- Shared clipboard
- Performance, performance, performance
- Tons of fixes
Wishlist / Roadmap

- Drag & drop
- 3D graphics
- Ports to more platforms (BSD hosts?)
- Guest additions for more guest operating systems
- More translations
- ...

innotek
Where to get?

www.virtualbox.org

innotek GmbH
Werkstrasse 24
71384 Weinstadt
www.innotek.de

www.innotek.de
Where to get?

www.virtualbox.org

innotek GmbH
Werkstrasse 24
71384 Weinstadt

www.innotek.de

Questions?