



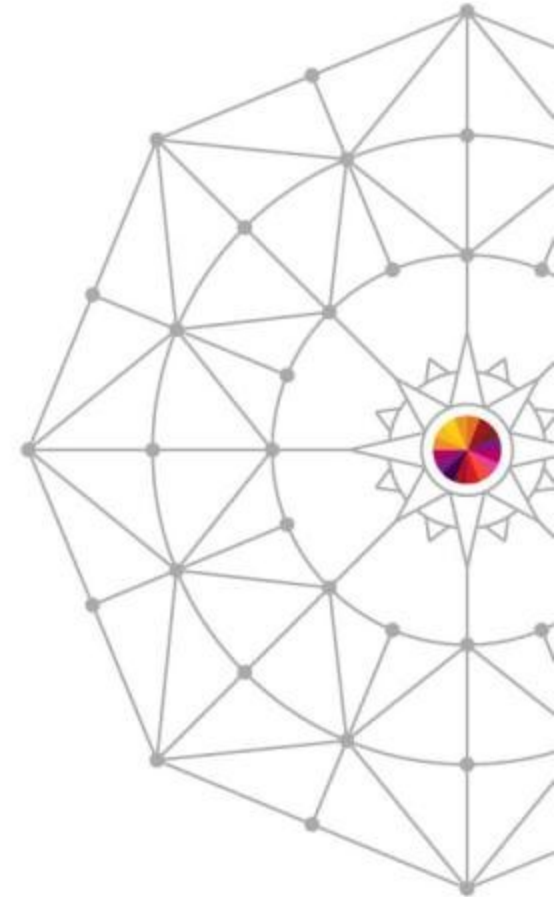
Know your competition

A review of qemu and KVM for System z

Mark Post

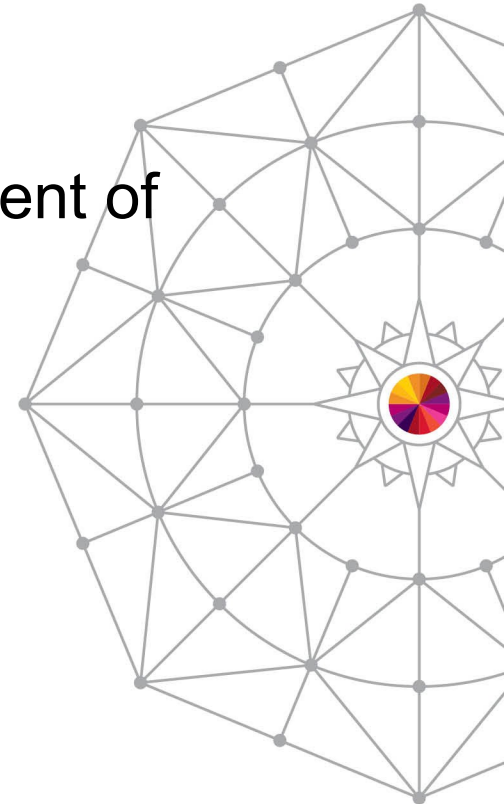
Software Engineer Consultant
mpost@suse.com

Tuesday, March 11, 2014
Session 14479



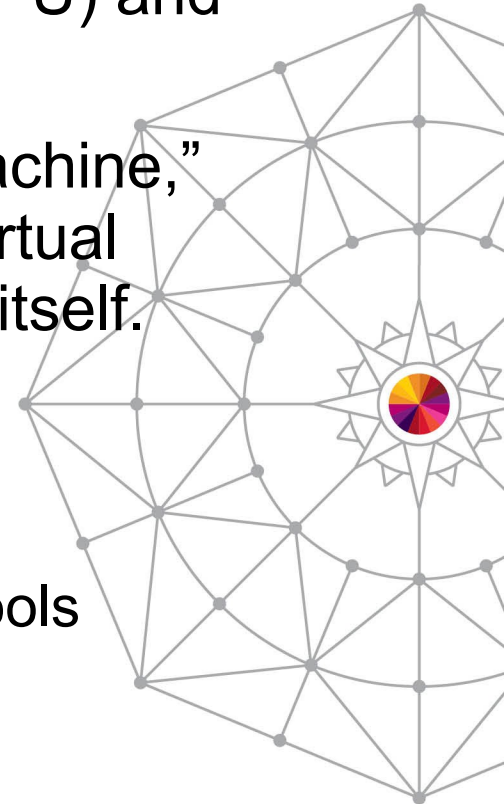
Agenda

- What is qemu and KVM
- High-level architecture
- Creation, interaction with, and management of virtual machines
 - Live demo?
- Terminology
- Questions



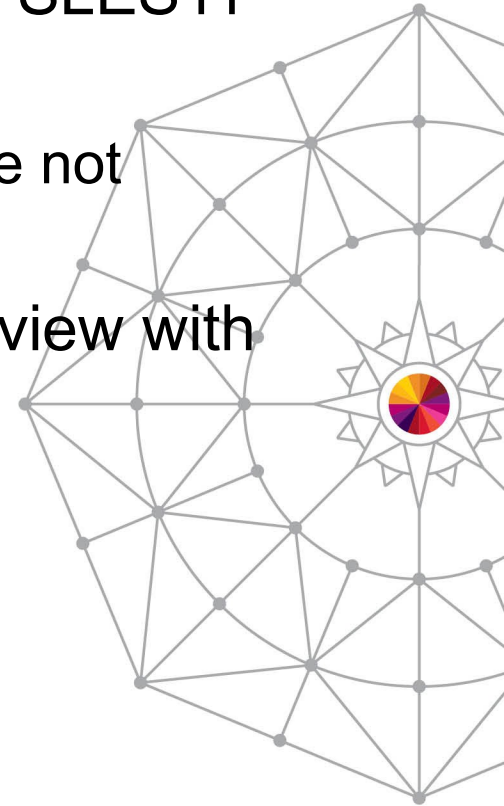
What is qemu and KVM

- qemu (quick emulator) is a processor (CPU) and other hardware emulator.
- KVM stands for “Kernel-based Virtual Machine,” meaning the infrastructure for creating virtual machines is included in the Linux kernel itself.
- KVM is a
 - Virtualization technology
 - Kernel module + userspace program + tools
 - Linux package



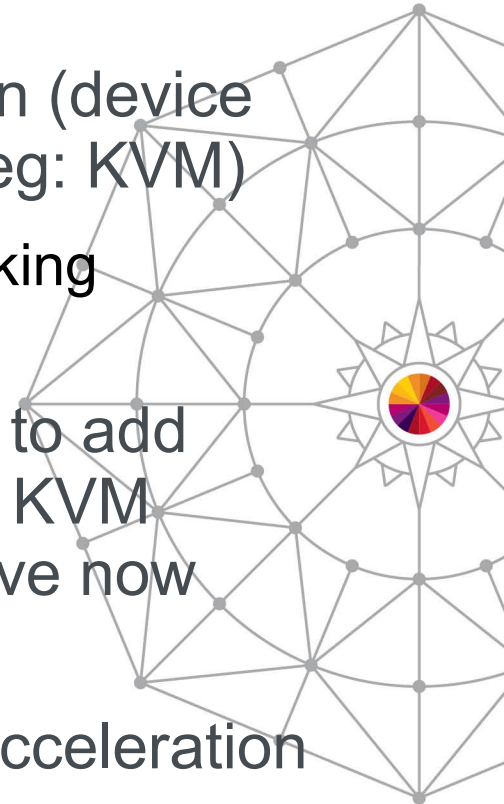
KVM/qemu status in SLES

- KVM/qemu is a Technology Preview with SLES11 SP3
 - Tech Previews are not supported, and are not intended for production use
- KVM/qemu will remain a Technology Preview with SLES12 when it ships.



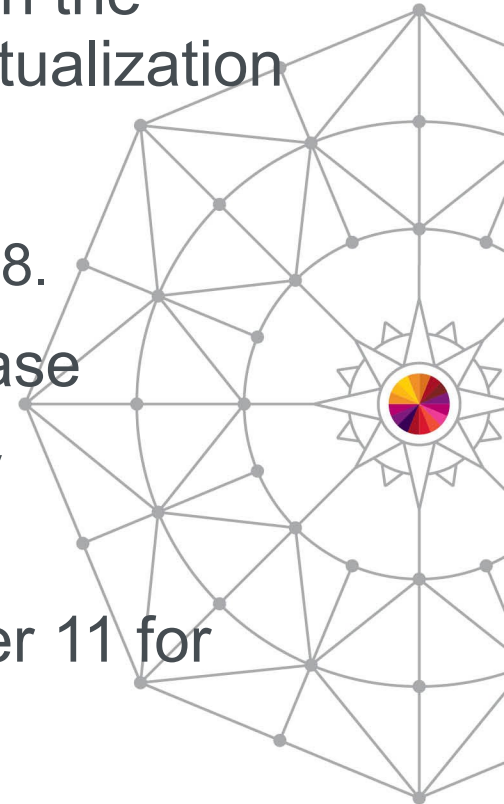
qemu

- Community project founded in 2003
- Provides (mostly PC) hardware emulation (device model) and ability to add accelerations (eg: KVM)
 - Host CPU, memory, storage, and networking resources
- Project forked (qemu-kvm) by Qumranet to add KVM support. Qemu project later added KVM support but not as complete. Projects have now mostly converged back together.
- qemu-kvm fork is mainly used for KVM acceleration
- Communicates with KVM via /dev/kvm



KVM origins

- First release in early 2007, coinciding with the introduction of x86 hardware-assisted virtualization
- Originally developed by Qumranet
 - Qumranet was bought by Red Hat in 2008.
- First included in 2.6.20 Linux kernel release
- Open source project (GPL v2) with many contributors
- Included in SUSE Linux Enterprise Server 11 for Intel/AMD since March, 2009

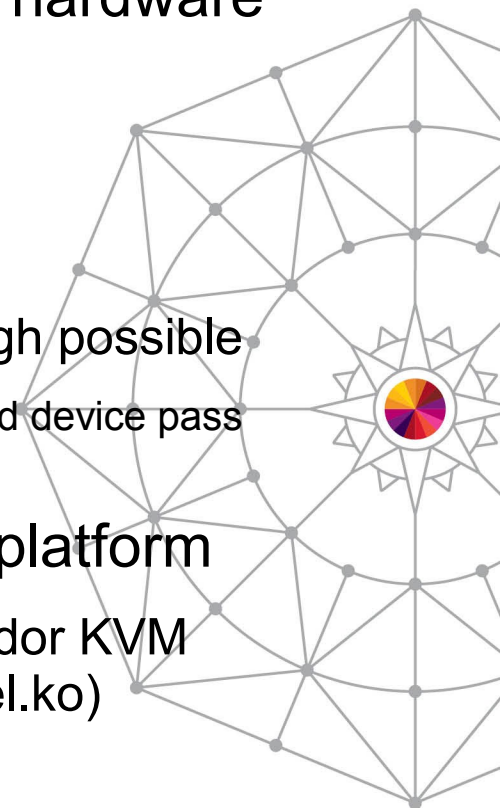


- KVM is based on hardware assisted virtualization, but also uses paravirtualization and trap and emulate as needed.
- Implemented as kernel modules
 - kvm.ko: provides virtualization infrastructure
 - kvm_amd.ko and kvm_intel.ko: hardware platform specific modules (no equivalent on s390x)
- Regular Linux kernel becomes virtual machine monitor (VMM, hypervisor), which can use any kernel infrastructure without modifications
- KVM virtual machines created by KVM accelerated qemu (qemu-kvm) run as regular user-space processes

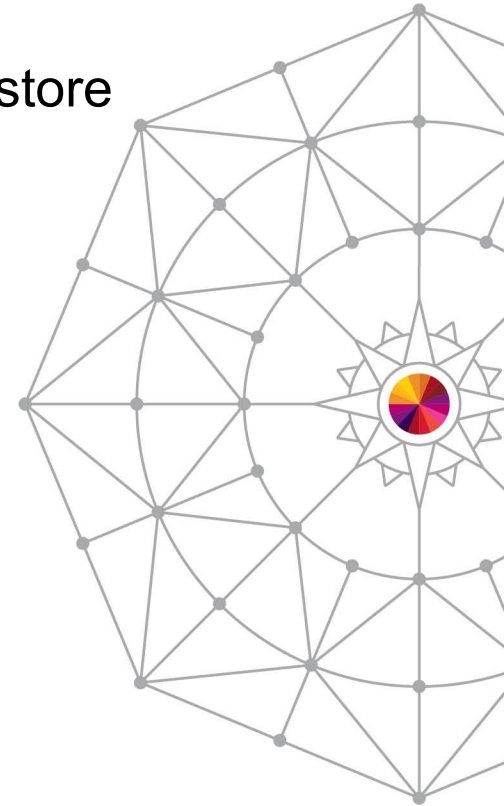


KVM virtualization

- Uses AMD-V, Intel VT-x, and System z SIE hardware virtualization
- Implements Full Virtualization
 - Guests run unmodified
 - Para-virtual drivers available, device pass through possible
 - For System z, the para-virtual drivers are mandatory and device pass through is not possible.
- Leverages Linux to provide a virtualization platform
 - Virtualization hardware control: generic and vendor KVM kernel modules (kvm.ko, kvm_amd.ko, kvm_intel.ko)
 - The Linux kernel acts as a hypervisor
 - A KVM accelerated QEMU userspace process runs the guest, which is just another userspace process to Linux

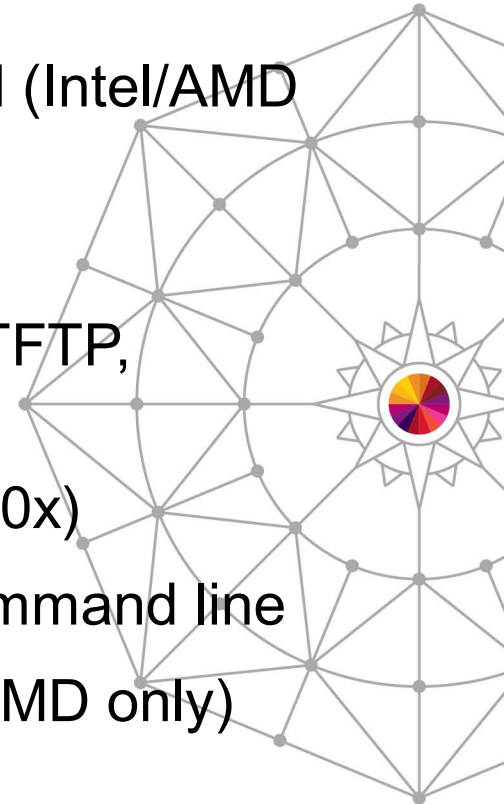


- Guest life-cycle controls
 - Start,stop,reboot, pause/resume, suspend/restore
 - Live migration (Intel/AMD only at this time)
 - Snapshots, delta storage images
- CPU, memory and disk over-commit
- Direct kernel boot option



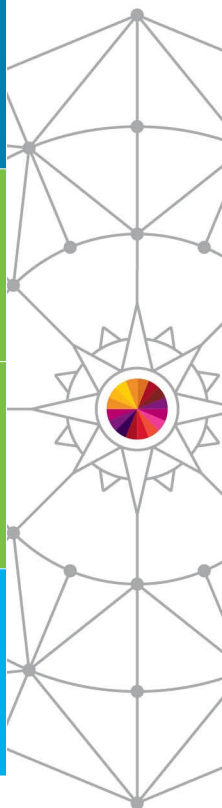
KVM features

- Transparent Huge Page (THP) optimized
- Kernel Samepage Merging (KSM) supported (Intel/AMD only)
- Non-root user support
- User-mode networking stack (DNS, DHCP, TFTP, BOOTP, SMB)
- Macvtap device networking (required on s390x)
- Guest details provided on the qemu-kvm command line
- Nested virtualization (“second level”) (Intel/AMD only)
- Built-in GDB server for guest debugging
- Various storage formats: raw, qcow2, qed, vmdk

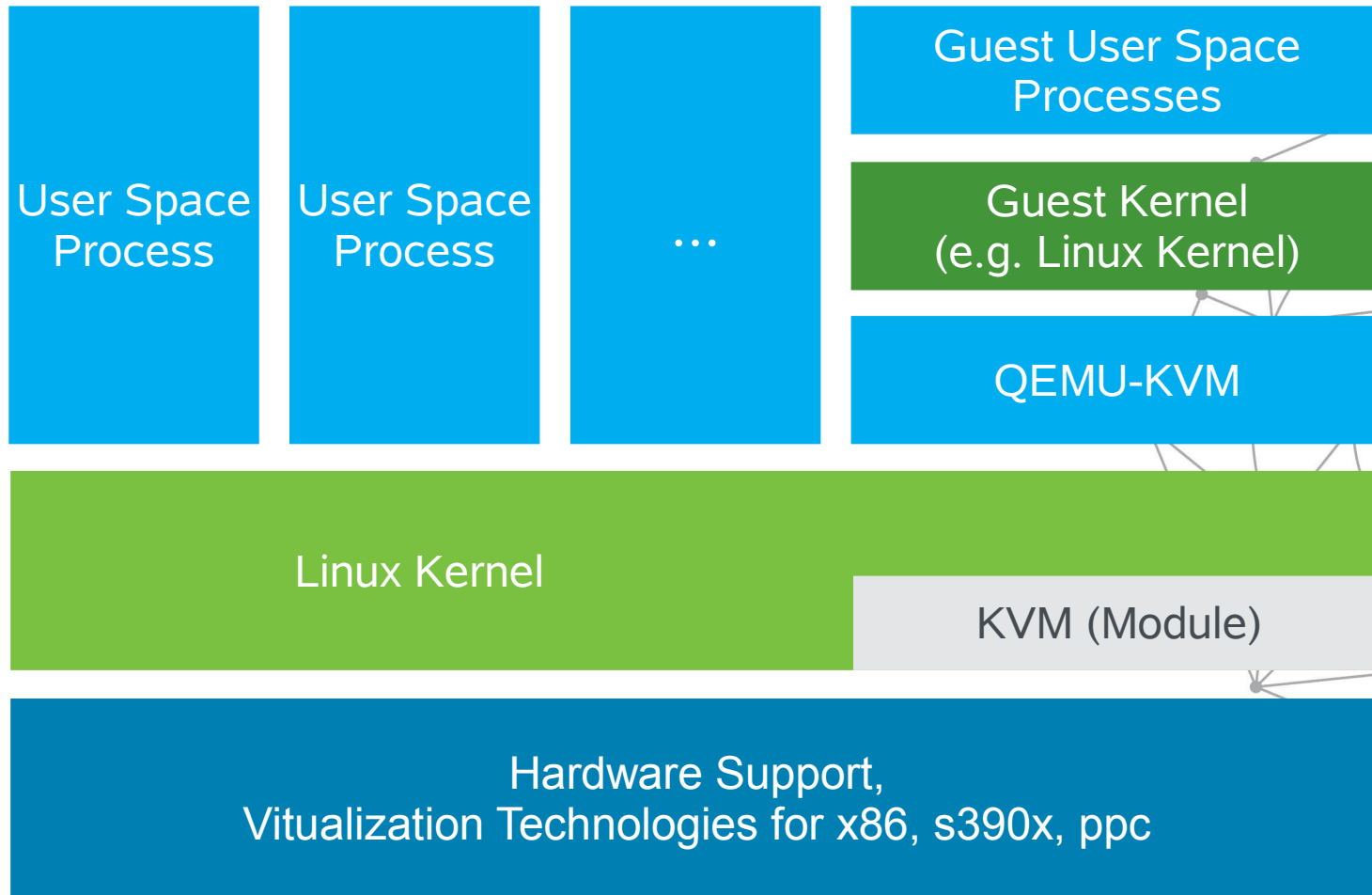


KVM – selected storage image formats and features

Name	Compression	Snapshot	Encryption	Deltas
raw				
qcow2	+	+	+	+
vmdk				+



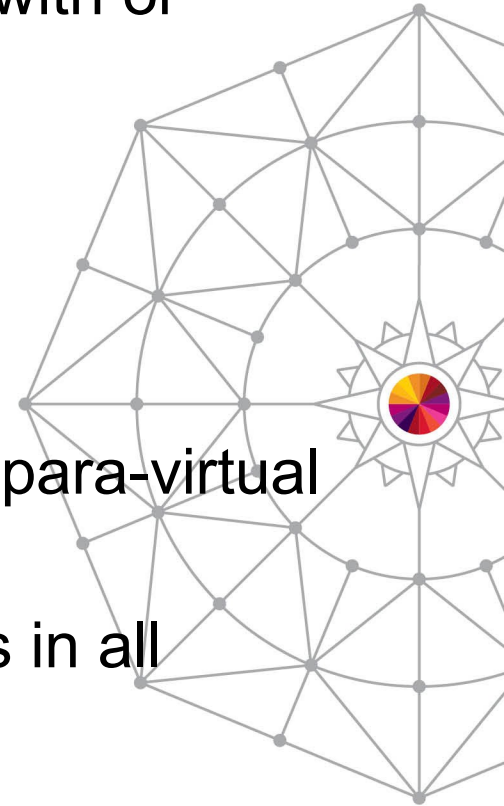
High-level architecture



Source: "Virtualization with KVM" training, B1 Systems GmbH

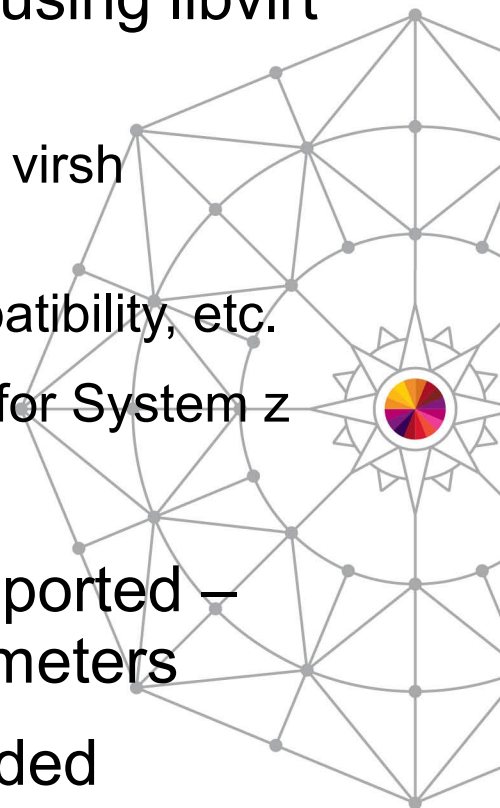
KVM limits supported by SUSE®

- Host RAM and CPU limits are the same with or without KVM modules loaded
- Guest RAM size: 512 GB
- Virtual CPUs per guest: 64
- NICs per guest: 8
- Block devices per guest: 4 emulated, 20 para-virtual (virtio-blk)
- Maximum number of guests: total vCPUs in all guests \leq 8 times total CPU cores in host



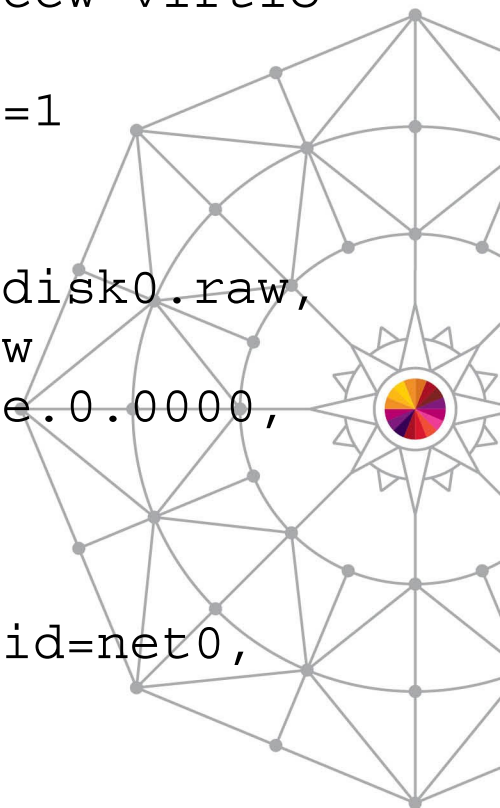
Using KVM

- For distributed systems, we recommend using libvirt and libvirt tools to access KVM
 - Includes: vm-install, virt-manager, virt-viewer, virsh commands
 - Adds additional security, configurability, compatibility, etc.
 - I'm not sure what we're going to recommend for System z
 - Plenty of command line tools to work with.
- Using qemu-kvm command-line also supported – documentation identifies supported parameters
- qemu-img image management tool provided



How's this for instantiating a guest?

```
/usr/bin/qemu-kvm -name sles11 -M s390-ccw-virtio
-enable-kvm
-m 512 -smp 1,sockets=1,cores=1,threads=1
-nographic
-rtc base=utc
-drive file=/var/lib/kvm/images/sles11/disk0.raw,
if=none,id=drive-virtio-disk0,format=raw
-device virtio-blk-ccw,scsi=off,devno=fe.0.0000,
drive=drive-virtio-disk0,id=virtio-
disk0,bootindex=1
-netdev tap,fd=19,id=hostnet0
-device virtio-net-ccw,netdev=hostnet0,id=net0,
mac=ea:02:91:19:cf:ff,devno=fe.0.0001
19<>/dev/tap?
```

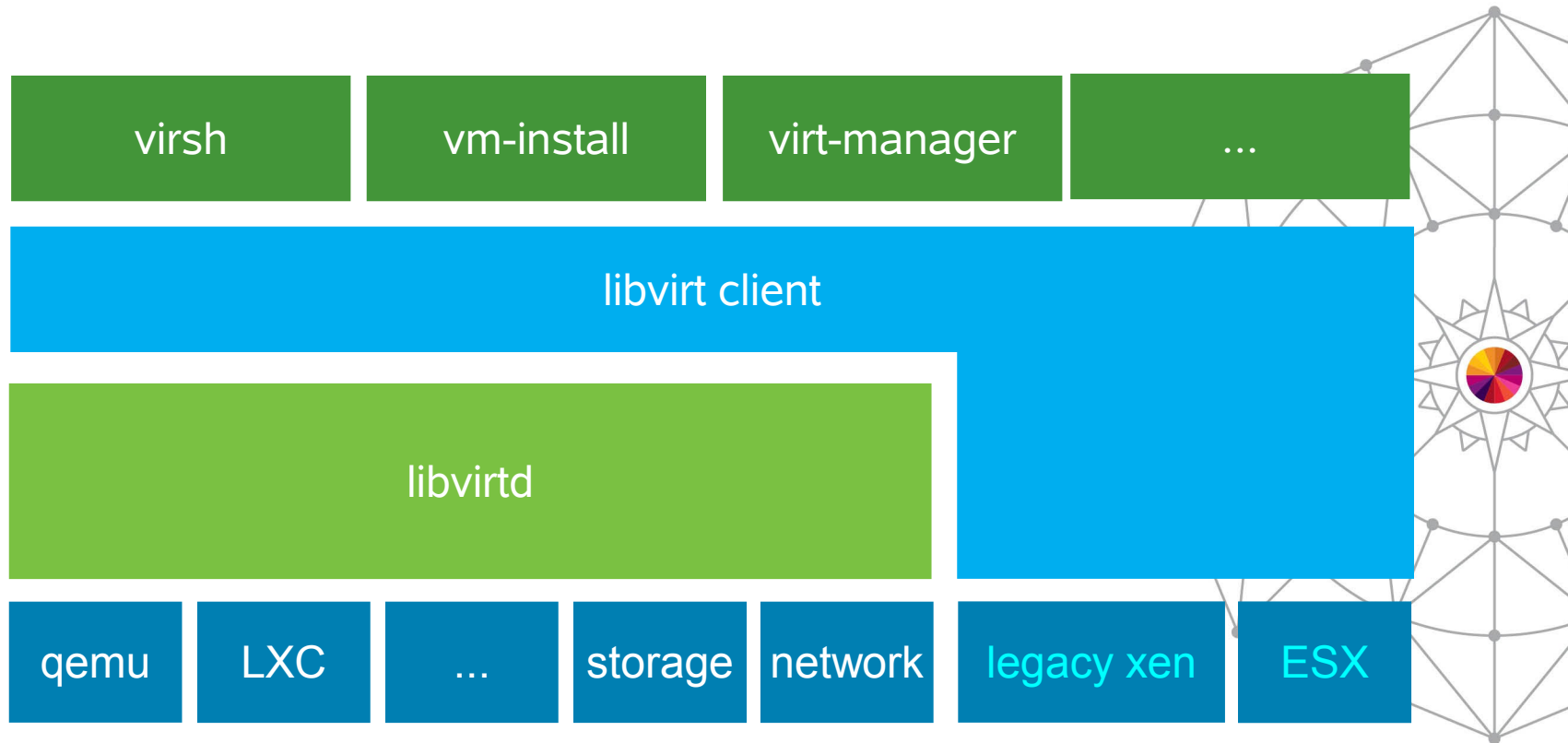


libvirt

- Virtualization library for managing one host
 - Domains, networks, storage, host devices, ...
- Share application stack between hypervisors
 - Xen, qemu/kvm, LXC, VMware, VirtualBox, ...
- Long-term API/ABI stability and compatibility
- Integration with other SUSE® Linux Enterprise components
 - AppArmor, SELinux, CGroups, Linux Audit Framework, PolicyKit, ...
- libvirt.org

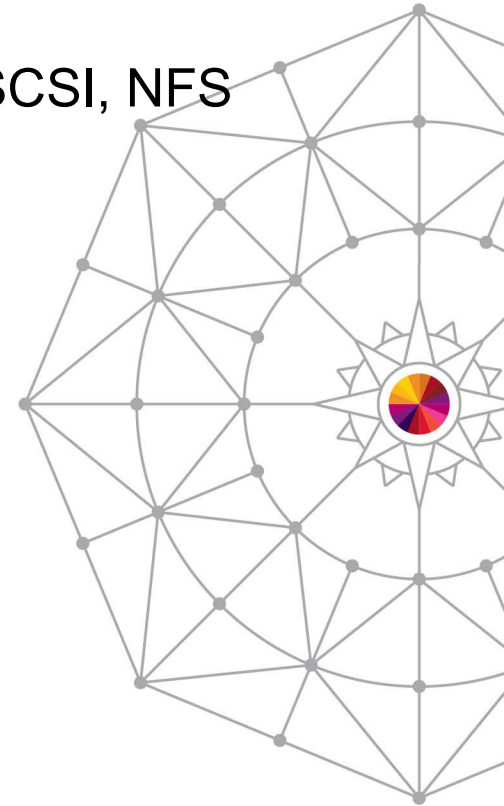


libvirt architecture



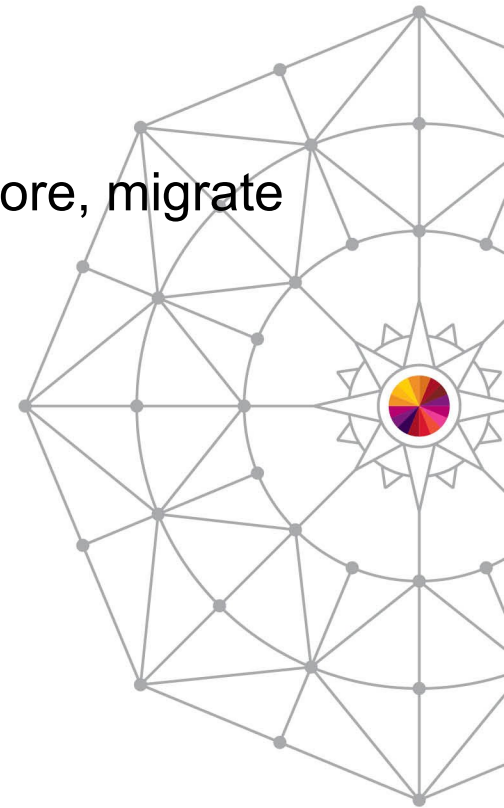
libvirt – host management

- Storage Pools
 - Dedicated device, partition, directory, LVM, iSCSI, NFS
- Storage Volumes
 - raw, qcow2, vmdk
- Network Interfaces
 - Bonds, bridges, ethernet devices, VLANs
- Virtual Networks
 - NAT with DHCP
 - Routed
 - Isolated



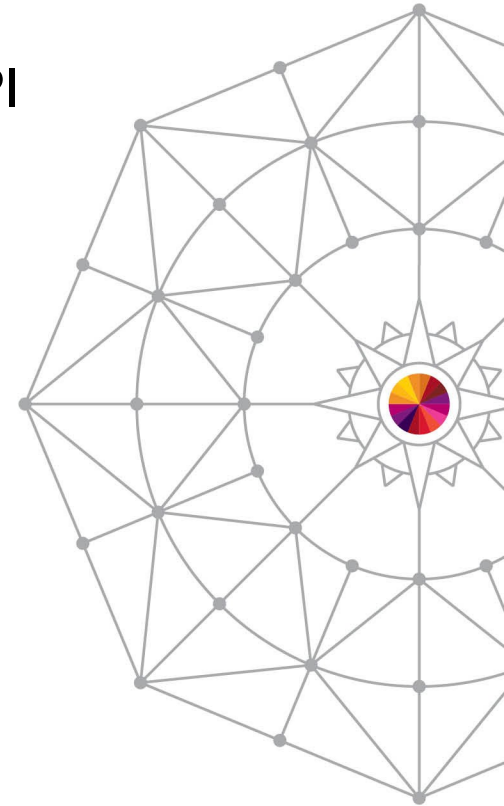
libvirt – domain management

- Domains defined in XML
- Lifecycle management
 - Define, start, stop, pause, resume, save, restore, migrate
- Configuration management
 - Change virtual hardware, e.g. memory, cpu
 - Add, remove, modify devices
- Tuning
 - CPU, memory, blkio, NUMA



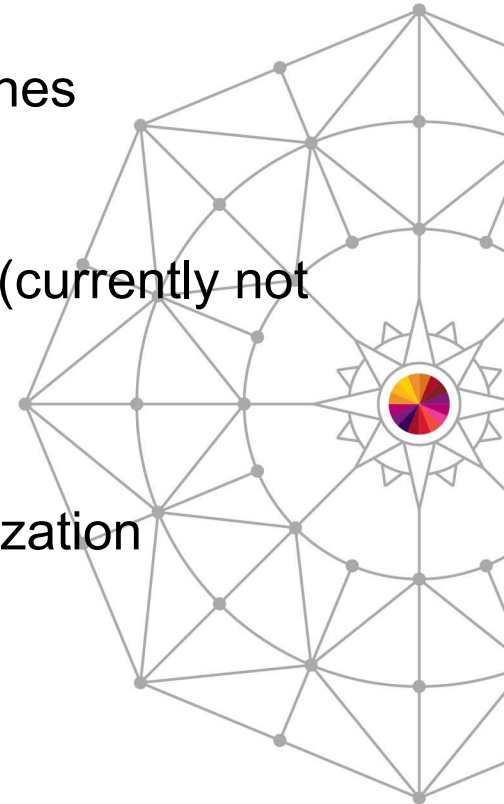
libvirt tools

- virsh
 - Command line application exposing libvirt API
- vm-install
 - Create virtual hardware configuration
 - Install an OS in a virtual machine



libvirt tools

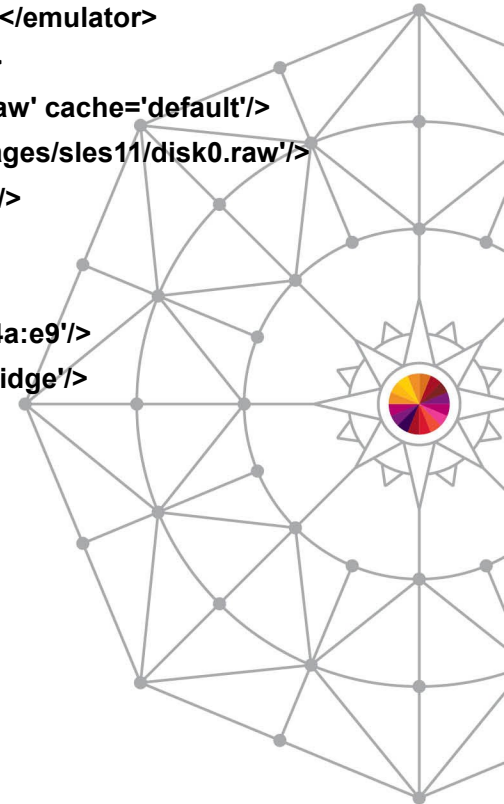
- virt-manager
 - Graphical tool for administering virtual machines
- virt-viewer
 - Graphical console client for virtual machines (currently not working on s390x)
- libvirt-cim
 - libvirt-based implementation of DMTF Virtualization Management standards



Domain definition file

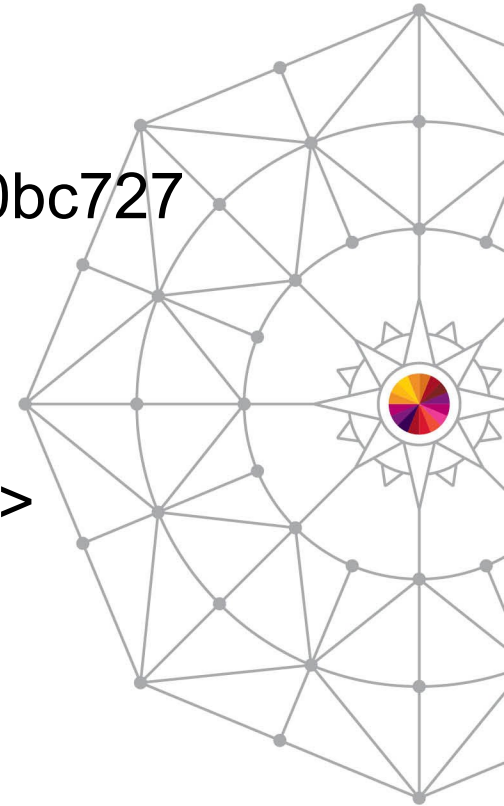
```
<domain type='kvm'>
  <name>sles11</name>
  <uuid>0b596405-e956-1412-f098-d4c0d00bc727
</uuid>
  <memory>524288</memory>
  <currentMemory>524288</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='s390x' machine='s390-ccw-virtio'>hvm
    </type>
    <boot dev='hd' />
  </os>
  <features />
  <clock offset='utc' />
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
```

```
<devices>
  <emulator>/usr/bin/qemu-kvm</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='raw' cache='default' />
    <source file='/var/lib/kvm/images/sles11/disk0.raw' />
    <target dev='vda' bus='virtio' />
  </disk>
  <interface type='direct'>
    <mac address='52:54:00:06:4a:e9' />
    <source dev='eth0' mode='bridge' />
    <model type='virtio' />
  </interface>
  <console type='pty'>
    <source path='/dev/pts/1' />
    <target type='sclp' port='0' />
  </console>
</devices>
</domain>
```



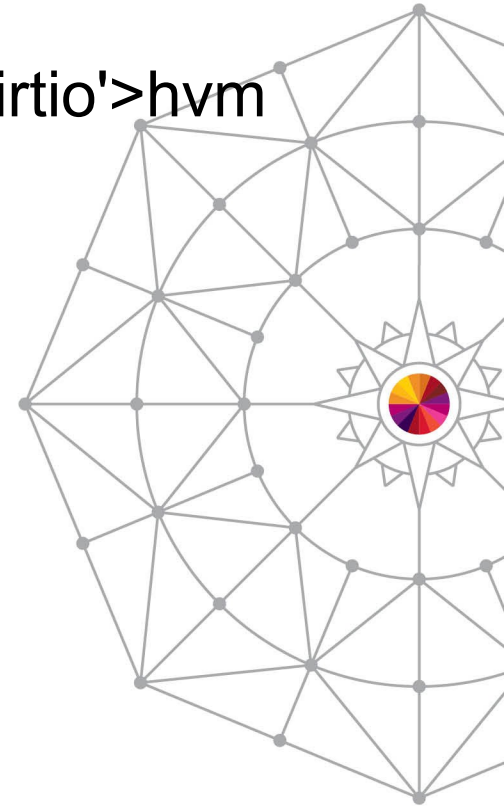
Domain definition file – part 1/6

```
<domain type='kvm'>  
  <name>sles11</name>  
  <uuid>0b596405-e956-1412-f098-d4c0d00bc727</uuid>  
  <memory>524288</memory>  
  <currentMemory>524288</currentMemory>  
  <vcpu>2</vcpu>
```



Domain definition file – part 2/6

```
<os>  
  <type arch='s390x' machine='s390-ccw-virtio'>hvm  
</type>  
  <boot dev='hd'/>  
</os>
```



Domain definition file – part 3/6

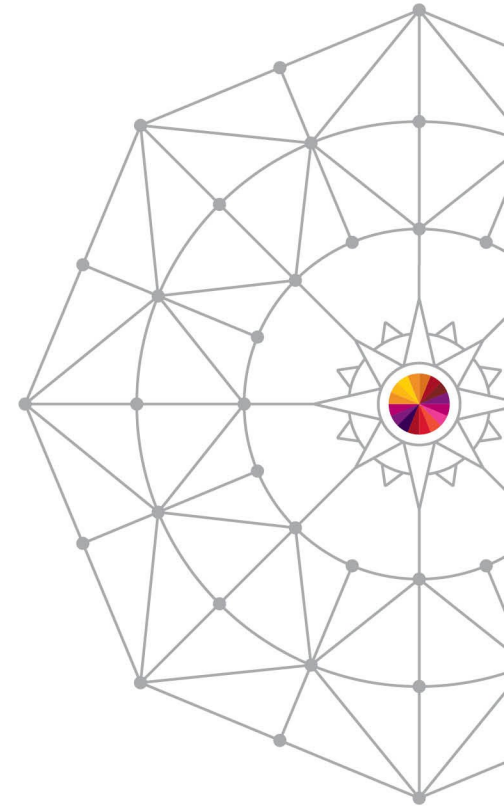
```
<features/>
```

```
<clock offset='utc' />
```

```
<on_poweroff>destroy</on_poweroff>
```

```
<on_reboot>restart</on_reboot>
```

```
<on_crash>destroy</on_crash>
```



Domain definition file – part 4/6

```
<devices>
```

```
<emulator>/usr/bin/qemu-kvm</emulator>
```

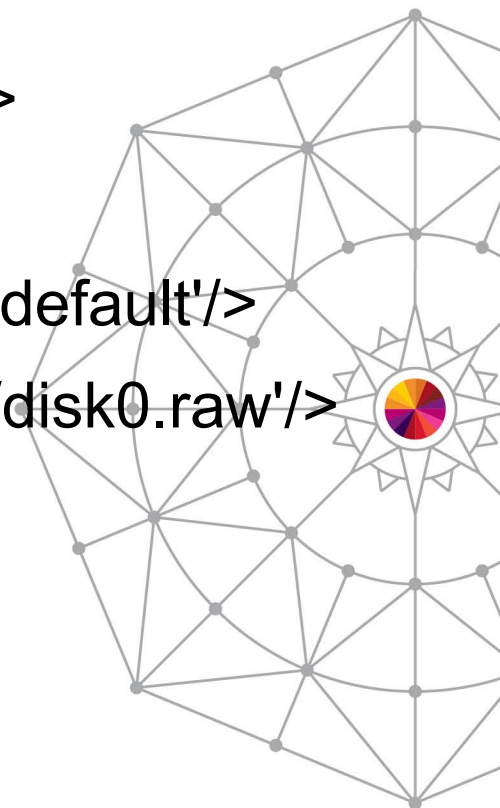
```
<disk type='file' device='disk'>
```

```
<driver name='qemu' type='raw' cache='default'/>
```

```
<source file='/var/lib/kvm/images/sles11/disk0.raw'/>
```

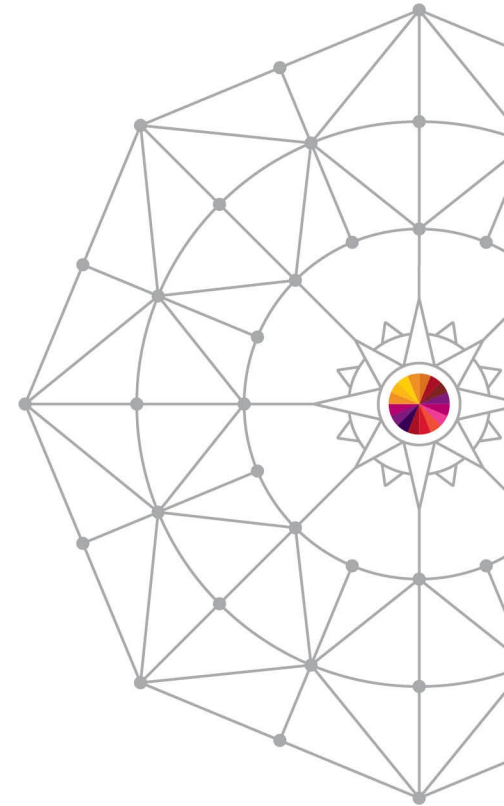
```
<target dev='vda' bus='virtio'/>
```

```
</disk>
```



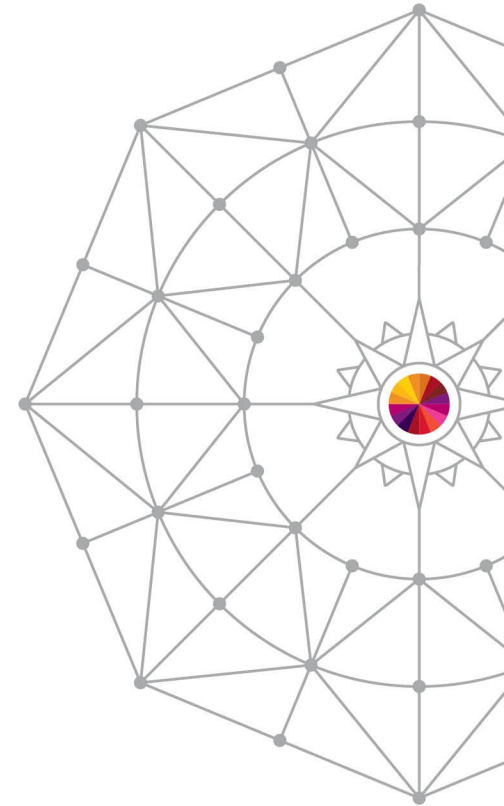
Domain definition file – part 5/6

```
<interface type='direct'>  
  <mac address='52:54:00:06:4a:e9'/>  
  <source dev='eth0' mode='bridge'/>  
  <model type='virtio'/>  
</interface>
```



Domain definition file – part 6/6

```
<console type='pty'>  
  <source path='/dev/pts/1'>  
  <target type='sclp' port='0'>  
</console>  
</devices>  
</domain>
```

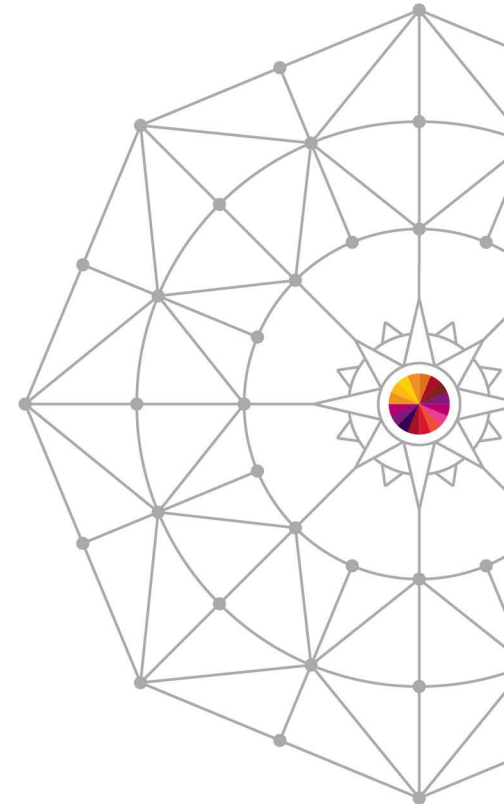


- All that XML allows you to do this:
virsh start sles11 --console
or
virsh start sles11
and then later
virsh console sles11
- “virsh help” or “man virsh” will reveal a huge amount of subcommands and options to manipulate the host or a guest.



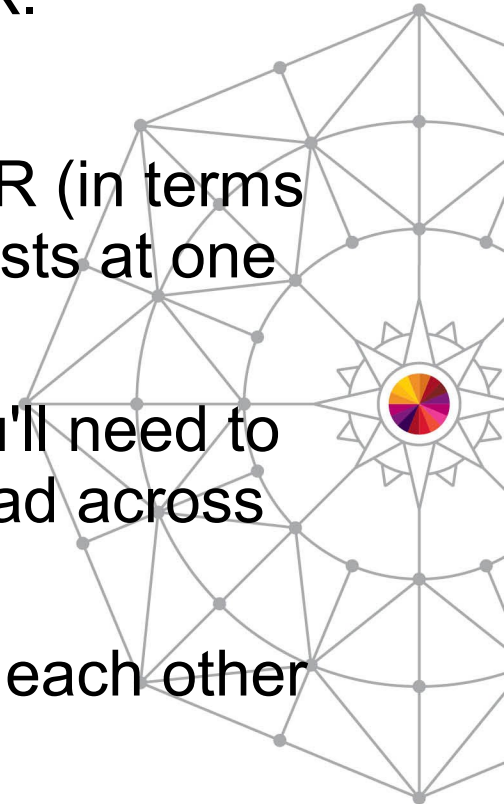
Places of Interest

- /etc/kvm/vm
- /etc/libvirt/
- /var/lib/kvm/images
- /var/lib/libvirt/images and qemu
- /var/log/libvirt/
- /var/run/libvirt/
- ~/.virt-manager/virt-manager.log



Things to consider

- Install your KVM Host system in an LPAR.
 - SIE won't be available to a z/VM guest
- If you're going to start with a “small” LPAR (in terms of real storage), don't start too many guests at one time.
- You can use multiple OSAs, but then you'll need to manually make sure the guests are spread across them.
- Guests will be able to communicate with each other via TCP/IP but not with the Host.



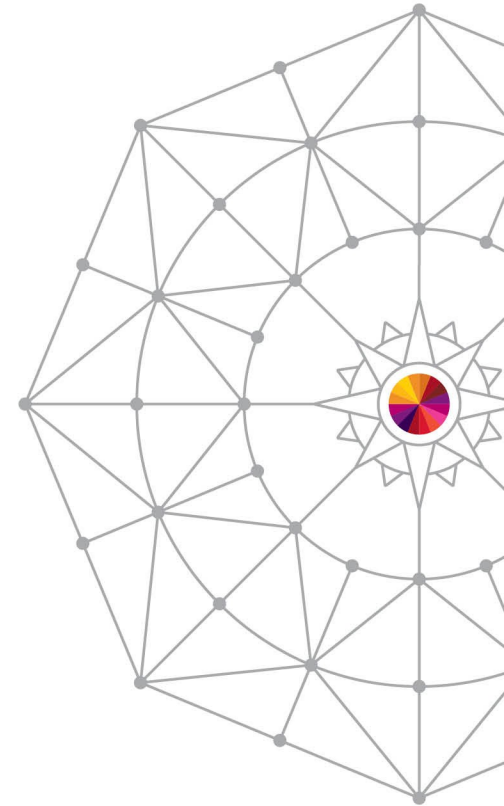
- If you don't use the “Install Hypervisor and Tools” module in YaST, you'll have to manually add “switch_amode” to your kernel parameters before you reboot.
- If you issue the qemu-kvm command directly, you'll need to issue this command for every virtual NIC used by a guest:

```
ip link add link eth0 name macvtap0 type macvtap mode bridge
```
- Hard to keep track of things if you don't use virsh or virt-manager.



Live demo

- Pray to the demo gods!

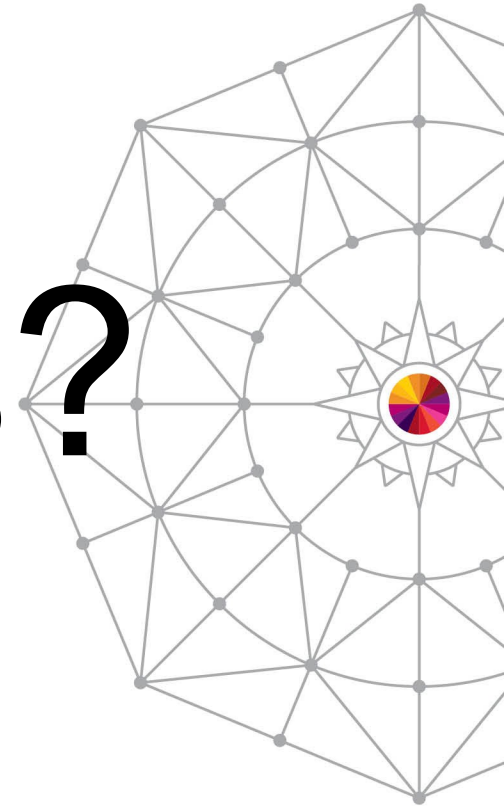


Terminology

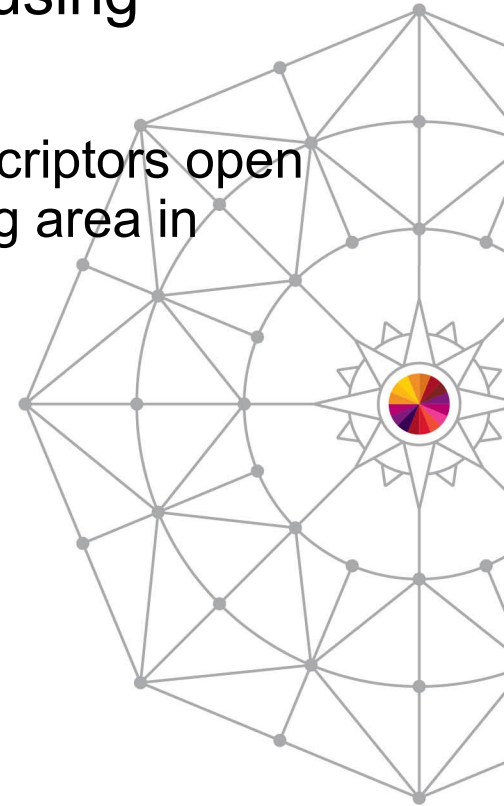
- Domain – guest virtual machine
- Image – file, partition, etc., used as the block device backing a virtual disk for a guest.
- VM – virtual machine, *not* z/VM.
- Host, Node – Linux instance hosting the Linux guests.
- Virtual Machine Manager - virt-manager



Questions?



- Is there a way to check whether KVM is using hardware virtualization (SIE instruction)?
 - Your qemu process should have kvm file descriptors open in `/proc/<pid>/fd/` and you should see a debug area in `/sys/kernel/debug/s390dbf/kvm-<pid>/`





Corporate Headquarters
Maxfeldstrasse 5
90409 Nuremberg
Germany

+49 911 740 53 0 (Worldwide)
www.suse.com

Join us on:
www.opensuse.org

Unpublished Work of SUSE. All Rights Reserved.

This work is an unpublished work and contains confidential, proprietary and trade secret information of SUSE. Access to this work is restricted to SUSE employees who have a need to know to perform tasks within the scope of their assignments. No part of this work may be practiced, performed, copied, distributed, revised, modified, translated, abridged, condensed, expanded, collected, or adapted without the prior written consent of SUSE. Any use or exploitation of this work without authorization could subject the perpetrator to criminal and civil liability.

General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. SUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for SUSE products remains at the sole discretion of SUSE. Further, SUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All SUSE marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.

