KVM Forum 2012

KVM in the enterprise: an early adopter's take

Fernando Luis Vázquez Cao <fernando@oss.ntt.co.jp>
NTT Open Source Software Center

KVM at NTT : Early days

- KVM's upstream merge
  - Linux 2.6.20 (released 5 February, 2007)
- A new toy
  - 2007: informal tests
    - “Playing with a new toy” phase
  - 2008: in-depth evaluation with an eye on enterprise adoption
    - “Hey, it looks promising” phase

KVM at NTT : Getting serious

- More than a toy
  - 2009/3: NTT Labs decide to port the Xen-based Kemari to KVM
    - “We'd rather hedge our bets” phase
  - 2009/9: Red Hat includes KVM in RHEL 5.4
    - “Sigh of relief as we confirm that we are not the only ones betting on KVM” phase
  - 2009/10: NTT Communications announces beta release of KVM-based public cloud solution BHB
    - “Let's see if it can withstand the test of the real world” phase

KVM at NTT : The new normal

- Enterprise-ready hypervisor
  - 2010/4: BHB drops beta tag to become a fully supported public cloud solution
  - 2010/8: KVM enters NTT Open Source Software Center (NTT OSSC)'s official support menu
    - All NTT Group member companies can use KVM with the backing of NTT OSSC's support team

KVM at NTT : Reality check

- Promising but unpolished newcomer
  - 2010~: steady stream of contributions to KVM and virtualization related components
    - QEMU/KVM bug fixes and improvements: live migration, dirty page tracking, x86 decoder/emulator, rtl8139, etc
    - Linux bridge: IGMP snooping fixes
  - 2010~: contributing to closing the gap with certain established virtualization product
    - Sheepdog: distributed storage system
    - Ryu: OpenFlow network operating system

Early adoption : the good

- Easier to get involved when the size of the code base is still manageable
- Head start: if the project happens to be a success your are well-positioned to make the best of it
- Muti-faceted benefits of being an active member of the community
  - Easier to enter open source's virtuous cycle of give-and-take
  - You get to know not only the code, but the design decisions and compromises behind it
    - Fundamental when you want to add a new feature or want to fix an issue

Early adoption : the bad

- The user space side of things, qemu,wasn't that small
  - To make things more interesting KVM support started as a fork of qemu
  - To cap it all, KVM-specific code and TCG-specific code was interestingly intermixed
- Libvirt was playing catch-up with KVM
  - Management stack becomes a potpourri of libvirt/virsh code/scripts and custom code
- First row seat to the project's maturation process: too close to the action
  - One gets to experience first hand a considerable percentage of the bugs whose fix-up led to the stable code base we have today
- Some features that users expect to be there didn't exist or were didn't just work
  - You have to implement things your self
  - … and try to get your work merged upstream and in your favorite enterprise distribution

Early adoption : the ugly

- A not so virtuous give-and-take cycle
  - It can take a while for a bug fix/new feature you contribute to get merged upstream and, in turn, making it into a enterprise distro
    - Forced to maintain your own forks of qemu and the kernel
- It is not clear which features are production ready and actively maintained
  - If a feature is there people will assume it is usable and stable... until things start to break
- A tendency not to look beyond qemu and the kvm module
  - The kernel version and the hardware can restrict KVM's ability to provide proper emulation for certain guests

History repeating itself?

- KVM is now a very stable and efficient hypervisor
- Most of the action now takes place higher in the stack
  - Virtualization management software and, yeah, the so called cloud
- … which is very fragmented and basically a moving target
  - Open Stack, oVirt, Eucalyptus,...
- The fact that many of these solutions are under heavy development, and not exactly stable, does not keep some people from trying to sell them (too greedy?) and others from trying to use it (too eager/greedy?)
- Is the anything we can do to alleviate the pain of early adopters?

Alleviating the pain of early adopters

- Early adopters
  - Report bugs
  - Contribute fixes/features when possible
  - Fix and feature requests are fine but do not forget that sometimes programmers will not have that itch, so won't scratch it
  - Must be willing to share the pain: sometimes you will have to scratch you own itch

Alleviating the pain of early adopters

- Developers
  - Set the right expections
    - User will shoot themselves in the foot: if a feature is not ready either disable it or make it clear that it is not ready for prime time
    - Idem if it not being actively maintained anymore
  - Be responsive
    - Try to reply to all non-trollish email
  - Users cannot read your mind
    - If you are busy or the project does not scale due to lack of reviewers/maintainers let others know, they will understand!