Summer Presentation Script

*Intro Slide –*

Good afternoon everyone. My name is Ben Ryan, and I’m a senior at William & Mary studying computer science and Japanese, [Cristian introduces], and today we will be presenting on Implementing and interfacing with KVM. This is a project we have been working on as part of the High-Performance Computing Cluster Engineering Academy at LLNL under the mentorship of Dave Fox.

*What is a VM –*

A virtual machine, or VM, is an emulated system that runs in conjunction with a host operating system and physical hardware. The hypervisor is the hardware or software layer that separates physical hardware from the OS to allow for creation of virtual machines, of which there are two types. A hypervisor is considered "bare metal" (or type 1) when it runs directly on the hardware, as opposed to a hosted hypervisor (or type 2) which runs on the OS like any other program.

Bare-metal hypervisors provide more efficient usage of resources than hosted hypervisors. Bare-metal based VMs are therefore more efficient for smaller or specialized tasks, ease of testing configurations prior to deployment of new technology. Both types of hypervisor allow a VM to mitigate the need for new physical hardware in cases where a small task might have required additional hardware that would not have been fully utilized, such as a hard drive for a small server.

Recently, the use of container interfaces such as Docker have become popular. These programs typically work in a similar way to a hosted hypervisor VM, with the most major difference being that they, while they are more isolated than a standard process, they share an OS and potentially other resources, while a VM has its own OS.

*What is KVM –*

Kernel based virtual machines, or KVM, is a type of virtual machine that uses the Linux kernel as a bare-metal hypervisor. This allows the host machine to treat every VM, called a ‘guest’, as if it were a Linux process. Therefore, every guest has the same level of access to physical hardware as any other process and can easily and efficiently be used thanks to tools that Linux already has built in, such as the task scheduler.

The main benefits of KVM specifically is that it is built into Linux and is extremely efficient considering that the Linux kernel acts as the hypervisor, meaning it has inherently unrestricted access to the physical hardware and Linux tools. This in turn provides features for enhanced security, scaling, and uninterrupted host migration.

*Goals –* Some of the high level goals we set going into this project were to set up nodes in the student environment cluster to run KVM, develop a process to build and deploy various operating systems, preferably with automation, and finally to translate these tools and processes into real world applications.

*Node Setup 1 –* Although KVM comes build into linux, there is a bit of setup required on the host machine before creating VM guests. The first thing to do is to make sure that the CPU of the host machine where you want to create guests supports hardware virtualization, which can be done using some simple commands. The next step after this would be to enable virtualization in the BIOS of the host machine, which can be done by rebooting the node and accessing the BIOS when prompted during the boot. Inside the BIOS, some options will need to be set to enable “Virtualization technology”. Once the machine has booted up, virtualization will be enabled on the node and the next step is to install the required packages. After packages are in place, there are a few more steps to make sure the node is setup properly. We need to enable libvirt on startup with, and then check to see if the KVM module has loaded properly. Once that is confirmed, the last thing to do is allocate new IP addresses and names for the guests VMs that are going to be created. This is done by editing the ‘/etc/hosts’ file in the management node of the cluster, and then doing the same thing in the host file of the node that will be hosting the guest VMs. After the completion of these tasks, the node is set up to create guest machines.

*Implementation and Tools 1 –* Before creating a guest, an ISO file is needed for the operation system. Download your desired ISO from a mirror and make sure that it is placed the correct directory. This will be used in the virt-install command, which is the command line tool used to create the guest. Within this command, you are able to specify flags to define the specifications of the guest machine. Some of the options that will need to be set are the name of the guest, number of CPUs, amount of memory, OS variant, and location, which will be the path to the ISO image we downloaded. There is also an extra arguments flag where we can specify things such as which console to use and preconfiguration files to use during installation. Upon execution of the command, the guest VM will be allocated and installation will begin. The user will be prompted to enter some base configurations towards the end of the installation, most of which can be left as default settings. After this the system will boot up.

*Implementation and Tools 2 –* Implementation of guests with Debian based operating systems is mostly very similar to the process laid out for Red Hat systems outlined in the previous slide. You will still need to use the virt-install command, but in a bit of a different way. Way figured out that a Debian ISO image functions a little bit differently and cannot be booted using the –location flag as we did before. Instead, Debian systems use the –cdrom flag in the install command. The problem with using this flag is that it will not allow for extra arguments, which becomes a problem when trying to use preconfiguration files for the installation. The way around this is to still use the –location flag, but instead of giving it the path to an ISO image, we give it the url address of a network boot mirror. Another difference in implementing a Debian based system is that the serial console is not automatically configured in the installation as it is with something like CentOS. The way to work around this is to SSH into the guest system once it is up and running and edit the grub file. In some cases, we experienced trouble with connecting to the guest via SSH so the other option to fix this issue is to remove the nographics flag from the install command and then use virt-manager along with VNC to access a GUI of the VM where we can configure these changes.

*Networking –*

Most VMs would simply not be useful if they could not communicate with other local machines or the wider internet. VMs are emulations, and therefore, any network interfaces that a guest might posses are virtual themselves. To connect a virtual network to a physical network, a bridge device must be set up to allow communication between the two.

For example, If you were to set up a basic guest machine using the methods we have described, in order to interface on the physical network that the cluster itself utilizes, you would have to define a new bridge. The clusters as the Academy has configured them have a backend Ethernet network as well as an infiniband network. The Ethernet network is managed by a physical switch, and the nodes use the *em1* interface to access that switch. If we create a virtual bridge device that assigns and passes IP addresses via DHCP, then instruct *em1* to look for that bridge device, we have essentially created that bridge between the physical and virtual networks. To visualize this, it would be as though the port that a given node physically occupies on the Ethernet switch now acts as a virtual switch itself, assigning multiple virtual ports to one physical one that the VMs can occupy. The virtual bridge device *br1* 'bridges' the physical network via *em1* to the virtual network that the VMs share.

*Automation –*

Now, as you might imagine, installing different distributions of Linux or any other OS on a guest is a time-consuming and error prone process. Furthermore, certain scenarios might need even more robust control over the OS that is installed, such as the kernel version. Without automation of the installation process, use of VMs for testing over various hardware and software configurations would not be practical.

While one could install a guest via a GUI tool, using the libvirt package and ‘virt-install’ command opens the possibility of automation through BASH scripting and Ansible. Creating a script that can run the command that defines and creates a virtual machine, especially if it has capabilities to accept user input, means that a lengthy command need not be typed repeatedly.

However, one would still have to proceed manually through the installation prompts of the OS. Many Linux distributions provide a way to automate the actual installation through preconfiguration files. The file can be injected into an operating system’s ISO or network install file with a simple addition to the ‘virt-install’ command. These files provide instructions on OS configurations, packages to be installed, users to be defined, just to name a few, that will allow the installer to run without any user intervention. Therefore, the resulting guest will be fully configured from the original ‘virt-install’, completely automated. Some common forms of preconfiguration files are Kickstart for CentOS/RHEL-based distros, preseed for Ubuntu/Debian, and AutoYaST for OpenSUSE/SLES.

*Other Tools –* Some other tools:The virsh tool allows for some basic commands such as displaying a list of currently active guests on the machine, connecting to a guest, and destroying a guest instance. Also, as mentioned earlier, another tool that we used was virt-manager. Virt-manager is the GUI tool that is used to manage VM, as opposed to lib-virt for using a command line. In certain cases, we were not able to set up a serial console for a guest by connecting through SSH, so we were able to use virt-manager along with VNC to get a virtual desktop of the guest we created. In order to get RealVNC to work together with virt-manager, some additional setup is required. Another tool worth mentioning is oVirt, which is an open source virtualization management platform created by Red Hat. OVirt is useful for large scale, kernel-based virtualization management.

*Outcomes –* As far as what we accomplished, Ben and I were able to outline the procedures to create guests instances of various Linux distributions. It was our first tie working with virtual machines and KVM, so there was a lot of research involved at the beginning of the process followed by a lot of trial and error as we figured out how to integrate KVM. We made sure that we created detailed documentation of the procedures we used and things we learned in this process. It was also important that our documentation can be used for future reference by other admins since this could potentially implemented to enhance CI workflow. These operating systems include CentOS 7 and 8, Debian 9, Ubuntu 18.04 and 20.04, and openSUSE 15.2. We made sure that these systems were implemented with base configurations to install required packages, create a test user with privileges, and run an automated install of the OS. Later on, these guests can serve as an systems to build test environments on.

*Future Plans –* Ben and I have learned a lot about working with KVM during our time at LLNL, but our time as students was limited and . For our future plans, we had a couple of goals we would have liked to work towards if time allowed. The first would be to set up and configure a guest running TOSS as the operating system. While we did not get a chance to try implementing this, it would likely be fairly straightforward considering TOSS is a Red Hat based operating system similar to CentOS, which we worked with and know how to implement. Another goal of ours was to apply KVM in order to create test environments for MSR-safe kernel modules. A variety if Linux distributions for these environments is important because it would allow testing of modules on various kernels. These test environments would then be used to respond to Continuous integration requests in Gitlab by using a script or tool to run the requests on the allocated image, capturing results, and then deallocating and cleaning up the instance.

*Final Slide –*

Thank you so much for your attention. At this point, we’d be happy to take any questions or comments.