Virtualization and Containerization

RIT Linux Users Group - Week 3

What is Virtualization?

Virtualization is the creation of a virtual (rather than actual) version of something, such as an operating system, a server, a storage device or network resources.

What is Containerization?

Containerization is a light(er)weight alternative to a full virtual machine.

The container wraps the application in its own operating environment. This provides similar benefits to using a virtual machine.

Containerization has gained recent prominence with the open-source Docker. Docker containers are designed to run on everything from physical computers to virtual machines, bare-metal servers, OpenStack cloud clusters, public instances and more.

What does this do for us?

- Run multiple operating systems on one physical machine
- Divide system resources depending on current needs of each virtual machine
- Provide fault and security isolation at the hardware level
- Move and copy virtual machines as easily as moving and copying files
- Test out new software, or dangerous malware, in a safe isolated environment

Virtual Machines: Terminology

- Host System
  - Your physical computer, desktop, or server, that has virtual systems running on it
- Guest System
  - Any of the virtual machines running on a host system
- Passthrough
  - Any hardware on the host system that is given directly to the guest system.
  - This is usually things like CD-ROM drives, USB devices
- Hypervisor
  - Virtual Machine Manager
  - The specific program that allows the virtualization to take place

Types of Hypervisors

- Type-1, native or bare-metal hypervisors
  - Run directly on the host's hardware to control the hardware and to manage guest operating systems
  - First developed by IBM in the 1960s
  - Oracle VM Server, Citrix XenServer, VMware ESX/ESXi

- Type-2 or hosted hypervisors
  - These hypervisors run on a conventional operating system just as other computer programs do. Windows, OS X, Desktop Linux.
  - VMware Player, VirtualBox, QEMU

We'll mostly focus on Type-2 as they are the most user-friendly!

VirtualBox and VMWare

Two of the most popular Virtual Machine software packages. Available on Windows, OS X, and Linux. Both are free, or at least have a free option.

- Both have a simple user-friendly ui that makes creating virtual machines easy
- ‘Hardware’ that is attached to the virtual machine is configurable through menu options
- Can install addons or additions inside of the ‘guest’ operating system to improve usability, share mouse, keyboard, clipboard, shared folders, etc.
- Can ‘passthrough’ hardware such as CD-ROM, USB from the host.

Creating a VirtualBox VM

- Install VirtualBox (pretty obvious....)
  - https://www.virtualbox.org/wiki/Downloads
- Follow the VM creation wizard
- Start your VM
- ???
- Profit!

Really. Its really easy!

In a little more detail maybe...

Let’s take a quick look at VirtualBox...

Looking at Docker

Here is what Docker says about themselves:

> Docker is an open-source engine which automates the deployment of applications as highly portable, self-sufficient containers which are independent of hardware, language, framework, packaging system and hosting provider.

The ‘container’ is a sort of stripped-down virtual machine that includes just the parts that are required for your collection of software/services/whatever to run.

Installing Docker

I'll be showing how to install Docker on Ubuntu, but the instructions for Windows, OSX, and any other Linux are very similar.

Docker is supported on these Ubuntu operating systems:

- Ubuntu Wily 15.10
- Ubuntu Trusty 14.04 (LTS)
- Ubuntu Precise 12.04 (LTS)

Docker Prerequisites

Docker requires a 64-bit installation regardless of your Ubuntu version.

You must be running Linux Kernel 3.10 or later. You can check like this:

```
$ uname -r
3.11.0-15-generic
```

Kernels older than 3.10 lack some of the features required to run Docker containers. These older versions are known to have bugs which cause data loss and frequently panic under certain conditions.

Docker APT Sources

Update package information, ensure that APT works with the https method and ensure that CA certificates are installed and up to date.

```
$ apt-get update && apt-get install apt-transport-https ca-certificates
```

Add the Docker GPG Key

```
$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --
recv-keys 58118E89F3A912897C070ADBF76221572C52609D
```

Docker APT Sources

Add the correct repository depending on your version of Ubuntu.
Edit the file /etc/apt/sources.list.d/docker.list and add the repository

This should look something like:

```
deb https://apt.dockerproject.org/repo ubuntu-precise main
deb https://apt.dockerproject.org/repo ubuntu-wily main
```

After saving the file, update apt again to pull in the new package cache:

```
$ apt-get update
```

Install and Run the Docker Service

To install the Docker Service:

```
$ sudo apt-get install docker-engine
```

Start the Docker Service:

```
$ sudo service docker start
```

Run the Docker Hello World to verify that things are working correctly:

```
$ sudo docker run hello-world
```

Other Install Notes

By default the Docker service is only accessible to the root user. You will need sudo access to do anything.

If you want to control Docker as a normal user, you can add your user to the docker group and then log out and back in to apply the changes to your account.

You should then be able to run the Docker Hello World as your normal user:

```
$ docker run hello-world
```

Installing Containers in Docker

Download a pre-built image. This is an easy way to get started. Docker has some built in images that you can simply pull and get started. This container is downloaded from the “Docker Hub” which is like a container repository.

```
$ docker pull ubuntu
```

You can then run an interactive shell inside of this container!

```
$ docker run -i -t ubuntu /bin/bash
```

The -i flag starts an interactive container.
The -t flag creates a pseudo-TTY that attaches stdin and stdout.

What to do next?

That completely depends on what you want to run inside of Docker! Take a look at the very nice quickstart guide: https://docs.docker.com/engine/quickstart/

systemd-nspawn

- The systemd answer to containers
  - If your Linux is modern, it probably uses systemd so you likely already have it!

- Super light, super easy
  - Capable, but not a full container solution like Docker

Requirements

- systemd
- root or “sudo” access
- The ability to type “machinectl”

Getting Started

- Download a Linux image (Docker images *can* work, but are frequently too stripped down to work *well*)

- Or, create one if your distro has tools for it
  - `mkdir ~/MyContainer`
  - `# Arch: pacstrap -i -c -d ~/MyContainer base`
  - `# Debian: debootstrap -arch=amd64 jessie ~/MyContainer`

- If you create one, you can usually strip it down a lot (it doesn't need its own Linux kernel, for example)

The Quick Way

- You’re done!

- Start your container with `systemd-nspawn -b -D ~/MyContainer -n`

- Log in to your container with `machinectl login MyContainer`

But...

- What if we want a container that can talk to the outside network?
  - Or start on boot?
  - Or share files and folders with the host?

Starting at host boot

- The quick way (your container is at /var/lib/containers/MyContainer)
  - `systemctl enable machines.target`
  - `systemctl enable systemd-nspawn@MyContainer.service`

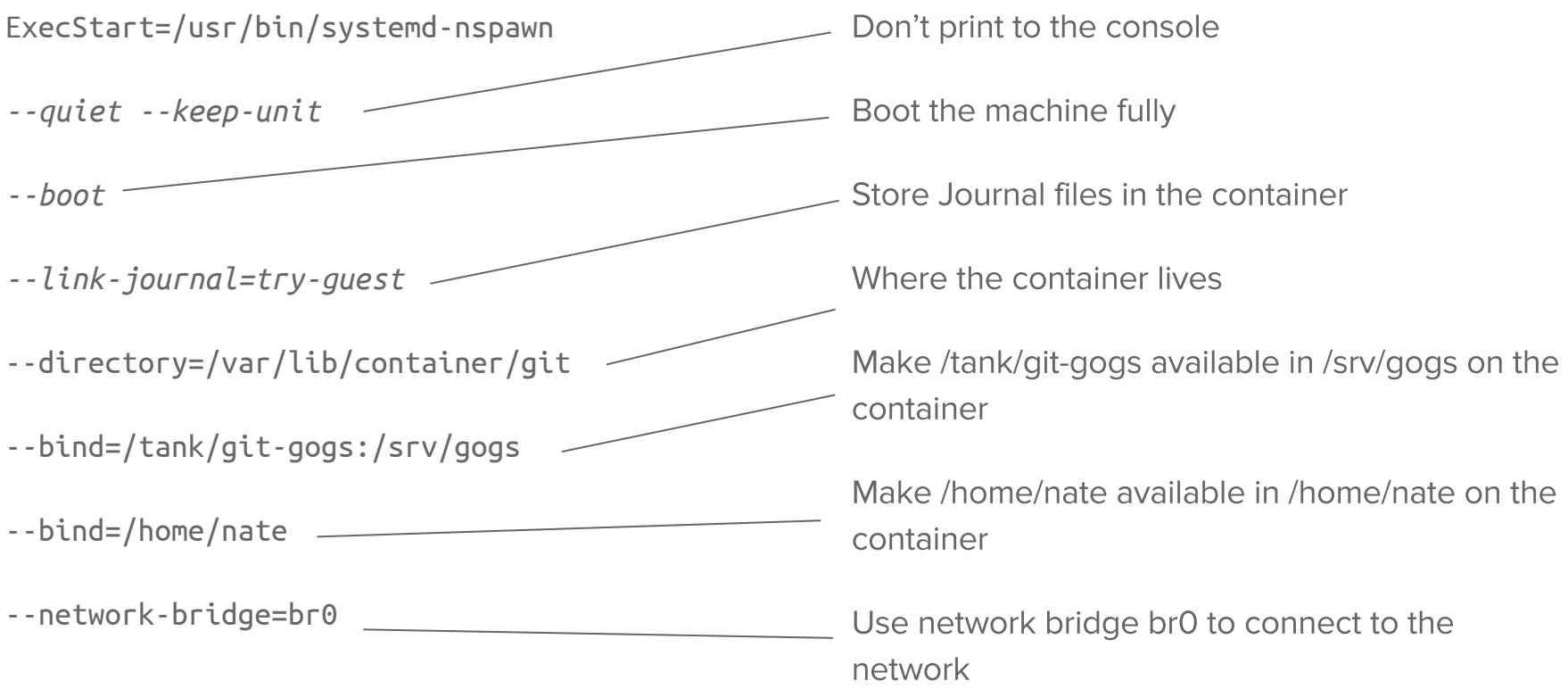
- The advanced way:
  - `cp /usr/lib/systemd/system/systemd-nspawn@.service /usr/lib/systemd/system/MyContainer.service`
  - `Edit to your heart's content (we'll get there in a sec)`
  - `systemctl enable machines.target`
  - `systemctl enable MyContainer.service`

Customizing your container

- Finally!

- Open up the file you copied; /usr/lib/systemd/system/MyContainer.service
  - You can do everything you need by editing the line starting with ExecStart

Real life example



You don’t want to hear it...

...but look at the systemd-nspawn man page. There are a LOT of options and useful examples.

File Permissions

- systemd-nspawn allows you to partition users and groups so they don't collide
  - Otherwise, users might get access to files you don't intend to give them access to
  - If you're not sharing files with your host, you don't need to worry about this

- What if you *want* file permissions to collide?
  - This gets super fun, especially with multiple containers
  - You *don't* need to do this if you're only binding user directories

Collisions

- Users and groups are backed by numbers
  - user and group “root” is 0
  - the first user and group start at 1000
  - you can set and change these numbers

- Create the users and groups you want on the container(s), then change their IDs to match
  - But be careful! Make sure only the things that you want to collide collide - this is super easy to mess up, and it isn’t obvious if you made a mistake
  - You will need to update file ownership manually to match the new IDs

machinectl

- systemd utility for managing containers
  - `machinectl poweroff MyContainer # power down a container`
  - `machinectl start MyContainer # power on a container`
  - `machinectl list # list running containers`
  - `machinectl show MyContainer # show details about MyContainer`
  - ...as well as many, many other things. Check out the help and man pages