

Booting, Initializing, and Virtualizing Linux

Objectives:

- ✓ 101.3 Change runlevels/boot targets and shutdown or reboot system
- ✓ 102.2 Install a boot manager
- ✓ 102.6 Linux as a virtualization guest
- ✓ 101.2 Boot the system

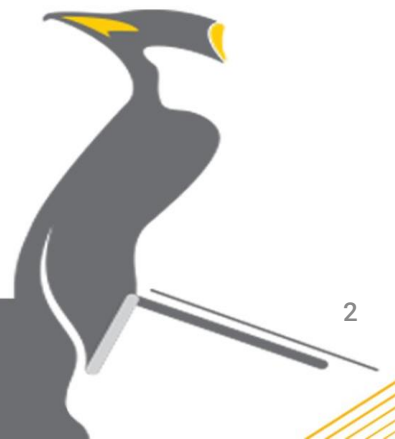


UNDERSTANDING THE BOOT PROCESS

When you turn on the power to your Linux system, it triggers a series of events that eventually leads to the login prompt. Normally, you don't worry about what happens behind the scenes of those events; you just log in and start using your applications.

However, there may be times when your Linux system doesn't boot quite correctly.

In this case, it helps to have a basic understanding of how Linux boots the operating system.

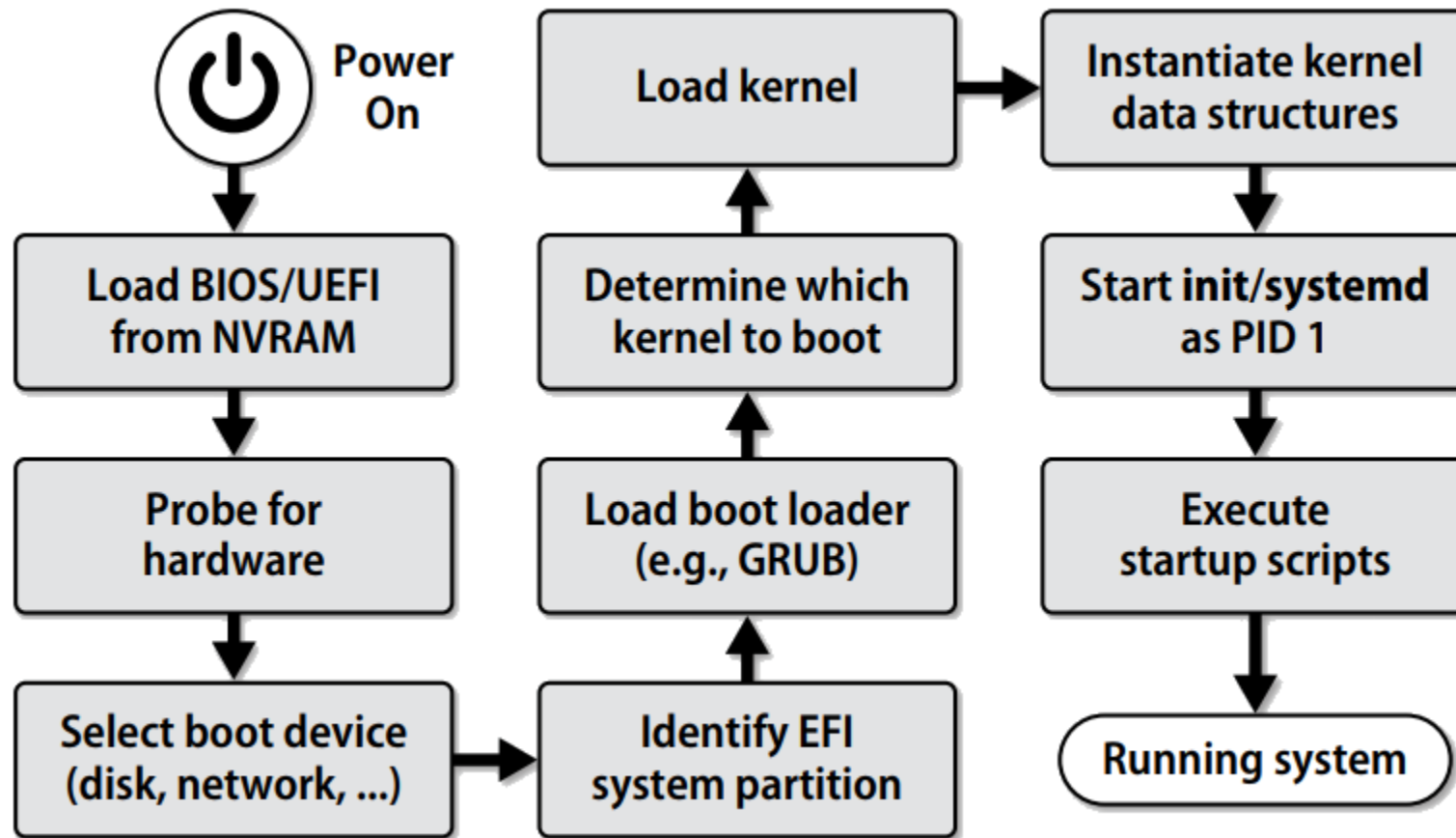


The Boot Process

❖ The Linux boot process can be split into these main steps:

1. The server firmware starts, performing a quick check of the hardware, called a **PowerOn Self-Test (POST)**, and then it looks for a boot loader program to run from a bootable device.
2. The boot loader runs and determines what Linux kernel program to load.
3. The kernel program loads into memory; prepares the system, such as mounting the root partition; and then runs the initialization program.
4. The initialization process starts the necessary background programs required for the system to operate (such as a graphical desktop manager for desktops, or web and database applications for servers).

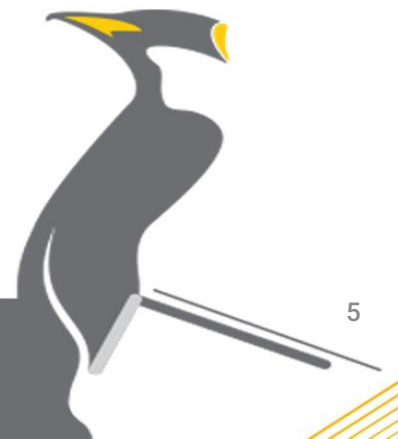
Linux boot process



LOOKING AT FIRMWARE

All IBM-compatible workstations and servers utilize some type of built-in firmware to control just how the installed operating system starts. On older workstations and servers, this firmware was called the Basic Input/Output System (BIOS).

On newer workstations and servers, a new method called the Unified Extensible Firmware Interface (UEFI) maintains the system hardware status and launches an installed operating system.



System Firmware

- ❖ When a machine is powered on, the CPU is hardwired to execute boot code stored in ROM.
- ❖ The system firmware typically knows about all the devices that live on the motherboard, such as SATA controllers, network interfaces, USB controllers, and sensors for power and temperature.
- ❖ In addition to allowing hardware-level configuration of these devices, the firmware lets you either expose them to the operating system or disable and hide them.
- ❖ During normal bootstrapping, the system firmware probes for hardware and disks, runs a simple set of health checks, and then looks for the next stage of bootstrapping code.
- ❖ The firmware UI lets you designate a boot device, usually by prioritizing a list of available options.

Understanding the Role of Firmware

- ❖ All IBM-compatible workstations and servers utilize some type of built-in firmware to control how the installed operating system starts.
- ❖ Basic Input/Output System (**BIOS**).
- ❖ Unified Extensible Firmware Interface (**UEFI**)

The BIOS Startup

- ❖ The BIOS firmware found in older workstations and servers was somewhat limited.
- ❖ It had a simple menu interface that allowed you to change some settings to control how the system found hardware and define what device the BIOS should use to start the operating system.
- ❖ One limitation of the original BIOS firmware was that it could read only one sector's worth of data from a hard drive into memory to run.
- ❖ That's not enough space to load an entire operating system.
- ❖ To get around that limitation, most operating systems split the boot process into two parts.



The BIOS Startup

- ❖ First, the BIOS runs a boot loader program, a small program that initializes the necessary hardware to find and run the full operating system program.
- ❖ It is usually located at another place on the same hard drive but sometimes on a separate internal or external storage device.
- ❖ The boot loader program usually has a configuration file so that you can tell it where to look to find the actual operating system file to run or even to produce a small menu allowing the user to boot between multiple operating systems.



The BIOS Startup

- ❖ To get things started, the BIOS must know where to find the boot loader program on an installed storage device.
- ❖ Most BIOS setups allow you to load the boot loader program from several locations:
 - ✓ An internal hard drive
 - ✓ An external hard drive
 - ✓ A CD or DVD drive
 - ✓ A USB memory stick
 - ✓ An ISO file
 - ✓ A network server using either NFS, HTTP, or FTP



master boot record (MBR)

- ❖ The MBR is the first sector on the first hard drive partition on the system.
- ❖ There is only one MBR for the computer system.
- ❖ The BIOS looks for the MBR and reads the program stored there into memory.
- ❖ The boot loader program mainly points to the location of the actual operating system kernel file, stored in a boot sector of a separate partition installed on the system.
- ❖ There are no size limitations on the kernel boot file.



The UEFI Startup

- ❖ As operating systems became more complicated, it eventually became clear that a new boot method needed to be developed.
- ❖ Intel created the **Extensible Firmware Interface (EFI)** in 1998 to address some of the limitations of BIOS.
- ❖ It was somewhat of a slow process, but by 2005, the idea caught on with other vendors, and the **Unified EFI (UEFI)** specification was adopted as a standard.

The UEFI Startup

- ❖ Instead of relying on a single boot sector on a hard drive to hold the boot loader program, UEFI specifies a special disk partition, called the EFI System Partition (ESP), to store boot loader programs.
- ❖ This allows for any size of boot loader program, plus the ability to store multiple boot loader programs for multiple operating systems.
- ❖ The ESP setup utilizes the old Microsoft File Allocation Table (FAT) filesystem to store the boot loader programs.
- ❖ On Linux systems, the ESP is typically mounted in the `/boot/efi` directory, and the boot loader files are typically stored using the `.efi` filename extension, such as `linux.efi`.



Extracting Information about the Boot Process

- ❖ **dmesg** - print or control the kernel ring buffer

`dmesg [options]`

- ❖ **journalctl** - Query the systemd journal

`journalctl [OPTIONS...] [MATCHES...]`

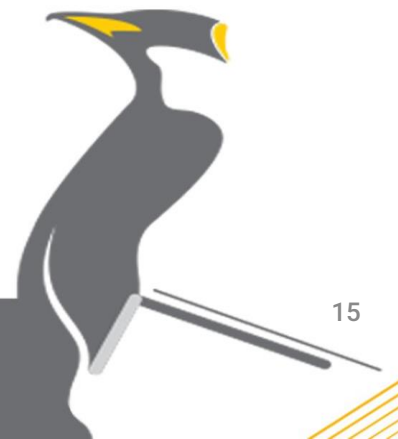
- ❖ **/var/log/boot** ==> on Debian distros

- ❖ **/var/log/boot.log** ==> on RedHat distros

LOOKING AT BOOT LOADERS

The boot loader program helps bridge the gap between the system firmware and the full Linux operating system kernel.

In Linux, there are several choices of boot loaders to use, which are covered in this section.



Boot Loader Principles

- ❖ The first version of the **GR**and **U**nified **B**ootloader (GRUB) boot loader (now called **GRUB Legacy**) was created in 1999 to provide a robust and configurable boot loader.
- ❖ GRUB quickly became the default boot loader for all Linux distributions, whether they were run on BIOS or UEFI systems.
- ❖ **GRUB2** was created in 2005 as a total rewrite of the GRUB Legacy system.
- ❖ It supports advanced features, such as
 - ✓ the ability to load hardware driver modules and
 - ✓ using logic statements to alter the boot menu options dynamically, depending on conditions detected on the system (such as if an external hard drive is connected).

Using GRUB Legacy as the Boot Loader

- ❖ GRUB Legacy allows you to select multiple kernels and/or operating systems using:
 - ✓ a menu interface
 - ✓ and interactive shell

Configuring GRUB Legacy

- ❖ The GRUB Legacy system stores the menu commands in a standard text configuration file called **menu.lst**:
 - ✓ It is stored in the **/boot/grub/** directory.
 - ✓ Red Hat derived Linux distributions use **grub.conf**
 - ✓ Also, you may find that the **menu.lst** file is symbolically linked to the **grub.conf** file.
- ❖ The GRUB Legacy configuration file consists of two sections:
 - ✓ Global definitions
 - must appear first in the configuration file
 - ✓ Operating system boot definitions

GRUB Legacy Configuration

❖ GRUB Legacy global commands:

Setting	Description
color	Specifies the foreground and background colors to use in the boot menu
default	Defines the default menu option to select
fallback	A secondary menu selection to use if the default menu option fails
hiddenmenu	Don't display the menu selection options
splashimage	Points to an image file to use as the background for the boot menu
timeout	Specifies the amount of time to wait for a menu selection before using the default

❖ GRUB Legacy operating system commands:

Setting	Description
title	The first line for each boot definition section, this is what appears in the boot menu.
root	Defines the disk and partition where the GRUB /boot folder partition is located on the system.
kernel	Defines the kernel image file stored in the /boot folder to load. Uses 0 for first hard drive and first partition
initrd	Defines the initial RAM disk file or filesystem, which contains drivers necessary for the kernel to interact with the system hardware.
rootnoverify	Defines non-Linux boot partitions, such as Windows.

Sample GRUB Legacy configuration file

```
default 0
timeout 10
color white/blue yellow/blue

title CentOS Linux
root (hd1,0)
kernel (hd1,0)/boot/vmlinuz
initrd /boot/initrd

title Windows
rootnoverify (hd0,0)
```

Installing GRUB Legacy

❖ **grub-install** - install GRUB on your drive

```
grub-install [OPTION] install_device
```

- ✓ After you build the GRUB Legacy configuration file, you must install the GRUB Legacy program in the MBR.
- ✓ The **grub-install** command uses a single parameter that indicates the partition on which to install GRUB.
- ✓ You can specify the partition using either the Linux or the GRUB Legacy format.

```
# grub-install /dev/sda
```

```
# grub-install /dev/sda1
```

```
# grub-install '(hd0)'
```

```
# grub-install 'hd(0,0)'
```

Using GRUB 2 as the Boot Loader

- ❖ Since the GRUB2 system was intended as an improvement over GRUB Legacy, many of the features are the same, with just a few twists.
- ❖ The GRUB2 system changes the configuration file name to **grub.cfg**.
- ❖ Where the file is stored depends on your system's firmware:
 - ✓ **BIOS**: The **grub.cfg** file is stored in the **/boot/grub/** or **/boot/grub2/** directory.
 - ✓ **UEFI**: The **grub.cfg** file is stored in the **/boot/efi/EFI/distro-name/** directory.
- ❖ For global commands, the **/etc/default/grub** configuration file is used.
- ❖ Although GRUB2 uses the **/boot/grub/grub.cfg** file as the configuration file, you should never modify that file.
 - ✓ Instead, there are separate configuration files stored in the **/etc/grub.d** folder.

Configuring GRUB2

Setting	Description
menuentry	The first line for each boot definition section; this is what appears in the boot menu.
set root	Defines the disk and partition where the GRUB2 /boot directory partition is located on the system. uses 0 for the first hard drive, the first partition is set to 1
linux, linux16	For BIOS systems, defines the kernel image file stored in the /boot directory to load.
linuxefi	For UEFI systems, defines the kernel image file stored in the /boot directory to load.
initrd	For BIOS systems, defines the initial RAM filesystem, which contains drivers necessary for the kernel to interact with the system hardware.
initrdefi	For UEFI systems, defines the initial RAM filesystem, which contains drivers necessary for the kernel to interact with the system hardware.

Sample GRUB2 configuration file

[...]

```
menuentry "CentOS Linux" {
```

[...]

```
set root=(hd1,1)
```

```
linux16 /vmlinuz[...]
```

```
initrd /initramfs[...]
```

```
}
```

```
menuentry "Windows" {
```

```
set root=(hd0,1)
```

[...]

Installing GRUB2

- ❖ **grub2-mkconfig** - Generate a GRUB configuration file.

```
grub-mkconfig [-o | --output=FILE]
```

- ✓ The **grub2-mkconfig** program reads configuration files stored in the **/etc/grub.d** folder and assembles the commands into the single **grub.cfg** configuration file.
- ✓ You can update the configuration file manually by using super user privileges and running the **grub2-mkconfig** command:

```
# grub2-mkconfig > /boot/grub2/grub.cfg
```

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

Adding Kernel Boot Parameters

Parameter	Description
console=	Set the console device
debug	Enable kernel debugging
init=	Execute the specified program, such as a Bash shell (/bin/bash) instead of /sbin/init
initrd=	Change the location of the initial RAM filesystem
mem	Set the total amount of available system memory
ro	Mount root filesystem as read-only
root=	Change the root filesystem
rootflags=	Set root filesystem's mount options
rw	Mount root filesystem as read-write
selinux	Disable SELinux at boot time
single, Single, 1, or S	Boot a SysVinit system to single-user mode
systemd.unit=	Boot a systemd system to specified target

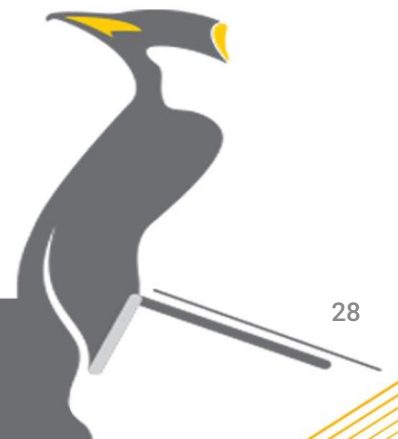
Using Alternative Boot Loaders

- ❖ The **systemd-boot** loader program is starting to gain popularity in Linux distributions that use the systemd initialization method.
- ❖ This boot loader generates a menu of boot image options and can load any UEFI boot image.
- ❖ The U-Boot boot loader (also called Das U-Boot) can boot from any type of disk.
- ❖ It can load any type of boot image.
 - ✓ **SYSLINUX**
 - ✓ **EXTLINUX**
 - ✓ **ISOLINUX**
 - ✓ **PXELINUX**
 - ✓ **MEMDISK**

THE INITIALIZATION PROCESS

After your Linux system has traversed the boot process, it enters final system initialization, where it needs to start various services.

A service, or daemon, is a program that performs a particular duty.



The Initialization Process

- ❖ The initialization daemon (init) determines which services are started and in what order.
- ❖ This daemon also allows you to stop and manage the various system services.
- ❖ There are two initialization daemons with which you should be familiar:

- ✓ **SysVinit**

- The **SysVinit (SysV)** was based on the Unix System V initialization daemon.
- Though it is not used by many major Linux distributions anymore, you still may find it lurking around that older Linux server at your company.

- ✓ **systemd**

- The **systemd** initialization method is the new kid on the block.
- Started around 2010, it is now the most popular system service initialization and management mechanism.
- This daemon reduces initialization time by starting services in a parallel manner.

Finding Initialization Program

Method #1

- ❖ Finding the init program file location

```
# which init
```

```
/sbin/init
```

- ❖ Checking the init program for links

```
# readlink -f /sbin/init
```

```
/usr/lib/systemd/system
```

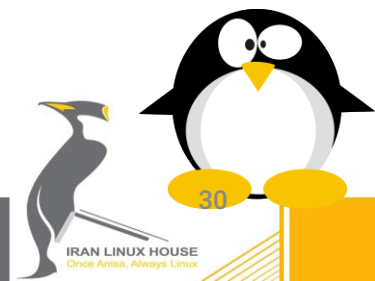
Method #2

- ❖ Checking PID 1

```
# ps -p 1
```

```
PID TTY TIME CMD
```

```
1 ? 00:00:06 systemd
```



USING THE SYSTEMD INITIALIZATION PROCESS

The systemd approach introduced a major paradigm shift in how Linux systems manage services. Services can now be started when the system boots, when a particular hardware component is attached to the system, when certain other services are started, and so on.

Some services can be started based upon a timer.



Exploring Unit Files

- ❖ The easiest way to start exploring systemd is through the **systemd units**.
- ❖ A unit defines a service, a group of services, or an action.
- ❖ Each unit consists of a name, a type, and a configuration file.
- ❖ There are currently 12 different systemd unit types:
 - ✓ automount
 - ✓ device
 - ✓ mount
 - ✓ path
 - ✓ scope
 - ✓ service
 - ✓ slice
 - ✓ snapshot
 - ✓ socket
 - ✓ swap
 - ✓ target
 - ✓ timer
- ❖ Looking at systemd units
 - `$ systemctl list-units`

Focusing on Service Unit Files

- ❖ Service unit files contain information such as which environment file to use, when a service must be started, what targets want this service started, and so on.
- ❖ These configuration files are located in different directories.
- ❖ Keep in mind that the directory location for a unit configuration file is critical, because if a file is found in two different directory locations, one will have precedence over the other.
- ❖ The following list shows the directory locations in ascending priority order:
 - ✓ `/etc/systemd/system/`
 - ✓ `/run/systemd/system/`
 - ✓ `/usr/lib/systemd/system/`

Looking at systemd unit files

```
$ systemctl list-unit-files
```

- ❖ In addition to the unit file's base name, you can see a unit file's state.
- ❖ Their states are called enablement states, and they refer to when the service is started.
- ❖ There are at least 12 different enablement states, but you'll commonly see these 3:
 - ✓ **enabled**: Service starts at system boot.
 - ✓ **disabled**: Service does not start at system boot.
 - ✓ **static**: Service starts if another unit depends on it. Can also be manually started.

Finding and displaying a systemd unit file

```
$ systemctl cat name.unit-type
```

- ❖ first displayed line shows the base name and directory location of the unit file.
- ❖ For service unit files, there are three primary configuration sections:
 - ✓ [Unit]
 - there are basic directives
 - ✓ [Service]
 - set configuration items that are specific to that service.
 - ✓ [Install]
 - determine what happens to a service if it is enabled or disabled.

Commonly used service unit file [Unit] section directives

Directive	Description
After	Sets this unit to start after the designated units.
Before	Sets this unit to start before the designated units.
Description	Describes the unit.
Documentation	Sets a list of uniform resource identifiers (URIs) that point to documentation sources. The URIs can be web locations, system files, info pages, and man pages.
Conflicts	Sets this unit to not start with the designated units. If any of the designated units start, this unit is not started. (Opposite of Requires.)
Requires	Sets this unit to start together with the designated units. If any of the designated units do not start, this unit is not started. (Opposite of Conflicts.)
Wants	Sets this unit to start together with the designated units. If any of the designated units do not start, this unit is still started.
AllowIsolate	

Commonly used service unit file [Service] section directives

Directive	Description
ExecReload	Indicates scripts or commands (and options) to run when unit is reloaded.
ExecStart	Indicates scripts or commands (and options) to run when unit is started.
ExecStop	Indicates scripts or commands (and options) to run when unit is stopped.
Environment	Sets environment variable substitutes, separated by a space.
Environment File	Indicates a file that contains environment variable substitutes.
RemainAfterExit	Set to either no (default) or yes. If set to yes, the service is left active even when the process started by ExecStart terminates. If set to no, then ExecStop is called when the process started by ExecStart terminates.
Restart	Service is restarted when the process started by ExecStart terminates. Ignored if a systemctl restart or systemctl stop command is issued. Set to no (default), on-success, on-failure, on-abnormal, on-watchdog, on-abort, or always.
Type	Sets the startup type.

Commonly used service unit file [Install] section directives

Directive	Description
Alias	Sets additional names that can denote the service in systemctl commands.
Also	Sets additional units that must be enabled or disabled for this service. Often the additional units are socket type units.
RequiredBy	Designates other units that require this service.
WantedBy	Designates which target unit manages this service.

Focusing on Target Unit Files

- ❖ For systemd, you need to understand the service unit files as well as the target unit files.
- ❖ The primary purpose of target unit files is to group together various services to start at system boot time.
- ❖ The default target unit file, `default.target`, is symbolically linked to the target unit file used at system boot.
- ❖ Looking at the `default.target` link:

```
$ systemctl get-default
```

Focusing on Target Unit Files

```
$ systemctl get-default
graphical.target
$ systemctl cat graphical.target
# /usr/lib/systemd/system/graphical.target
[...]
[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
Requires=multi-user.target
Wants=display-manager.service
Conflicts=rescue.service rescue.target
After=multi-user.target rescue.service rescue.target display-manager.service
AllowIsolate=yes
```

Commonly used system boot target unit files

❖ graphical.target

- ✓ Provides multiple users access to the system via local terminals and/or through the network.
- ✓ Graphical user interface (GUI) access is offered.

❖ multi-user.target

- ✓ Provides multiple users access to the system via local terminals and/or through the network.
- ✓ No GUI access is offered.

❖ runlevel n .target

- ✓ Provides backward compatibility to SysVinit systems, where n is set to 1–5 for the desired SysV runlevel equivalence.

Commonly used system boot target unit files

❖ Rescue Target

- ✓ When you jump your system to the rescue target, the system mounts all the local filesystems, only the root user is allowed to log into the system, networking services are turned off, and only a few other services are started.
- ✓ The `systemctl is-system-running` command will return the maintenance status.
- ✓ Running disk utilities to fix corrupted disks is a useful task in this particular target.
- ✓ This target is similar to the SysVinit single-user mode.

❖ Emergency Target

- ✓ When your system goes into emergency mode, the system mounts only the root filesystem, and it mounts it as read-only.
- ✓ Similar to rescue mode, it only allows the root user to log into the system, networking services are turned off, and only a few other services are started.
- ✓ The `systemctl is-system-running` command will return the maintenance status.
- ✓ If your system goes into emergency mode by itself, there are serious problems.
- ✓ This target is used for situations where even rescue mode cannot be reached.

Managing System Using Systemctl

❖ **systemctl** - Control the systemd system and service manager

`systemctl [OPTIONS...] COMMAND [NAME...]`

✓ **--faild**

- Finding failed units

✓ **cat PATTERN...**

- Show backing files of one or more units

✓ **get-default**

- Return the default target to boot into

✓ **set-default**

- Set the default target to boot into

✓ **isolate**

- Start the unit specified on the command line and its dependencies and stop all others.
- The isolate command is handy for jumping between system targets.
- When this command is used along with a target name for an argument, all services and processes not enabled in the listed target are stopped.
- Any services and processes enabled and not running in the listed target are started.

Managing System Using Systemctl

```
# systemctl get-default
```

```
graphical.target
```

```
# systemctl isolate multi-user.target
```

```
# systemctl status graphical.target
```

```
[...]
```

```
Active: inactive (dead) since Thu 2018-09-13 16:57:00 EDT; 4min 24s ago
```

```
Docs: man:systemd.special(7)
```

Managing System Using Systemctl

❖ Commonly used systemctl service management commands

✓ start

✓ stop

✓ status

✓ restart

✓ reload

✓ disable

✓ enable

✓ daemon-reload

✓ mask

✓ unmask

Managing System Using Systemctl

❖ Convenient systemctl service status commands

- ✓ is-active
- ✓ is-enabled
- ✓ is-failed
- ✓ is-system-running

❖ Operational statuses provided by systemctl is-system-running

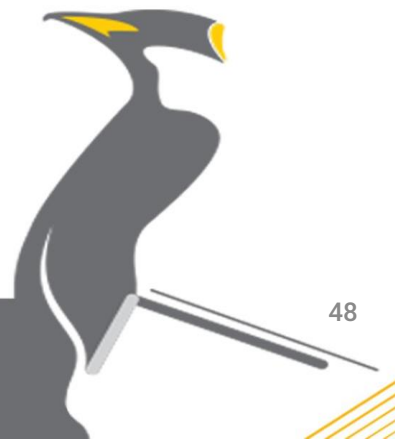
- ✓ **running** System is fully in working order.
- ✓ **degraded** System has one or more failed units.
- ✓ **maintenance** System is in emergency or recovery mode.
- ✓ **initializing** System is starting to boot.
- ✓ **starting** System is still booting.
- ✓ **stopping** System is starting to shut down.

Managing System Using Systemctl

```
# systemctl isolate emergency  
# systemctl is-system-running  
# systemctl list-units --type=target  
# systemctl default
```

USING THE SYSV INITIALIZATION PROCESS

Many server administrators have gone through the process of moving from a SysVinit system to a systemd system. Recall that systemd is backward compatible with SysVinit, so understanding SysVinit is important.



Understanding Runlevels

- ❖ At system boot time, instead of targets to determine what groups of services to start, SysVinit uses runlevels.

Runlevel	RedHat-Based Description	Debian-Based Description
0	Shut down the system.	
1, s, or S	Single-user mode used for system maintenance. (Similar to system rescue target.)	
2	Multi-user mode without networking services enabled.	Multi-user mode with GUI available.
3	Multi-user mode with networking services enabled.	-
4	Custom.	-
5	Multi-user mode with GUI available.	-
6	Reboot the system.	

Jumping Through runlevels

❖ **runlevel** - Print previous and current SysV runlevel

```
runlevel [options...]
```

```
# runlevel
```

```
N 5
```

❖ **init, telinit** - Change SysV runlevel

```
init [OPTIONS...] {COMMAND}
```

```
telinit [OPTIONS...] {COMMAND}
```

```
# init 3
```


SysVinit systems configuration file - inittab

- ❖ SysVinit systems employ a configuration file, **/etc/inittab**.
- ❖ In the past, this file started many different services, but in later years it started only terminal services and defined the default runlevel for a system.
- ❖ The **/etc/inittab** file line that sets the default runlevel:

```
# grep :initdefault: /etc/inittab
```

```
id:5:initdefault:
```

SysVinit systems configuration file - init.d

- ❖ Setting the default runlevel is the first step in configuring certain services to start at system initialization.
- ❖ Next, each service must have an initialization script located typically in the `/etc/init.d/` directory.

```
# ls -lF /etc/init.d/  
anacron*  
atd*  
[...]
```
- ❖ These initialization scripts are responsible for starting, stopping, restarting, reloading, and displaying the status of various system services.
- ❖ The program that calls these initialization scripts is the `rc` script, and it can reside in either the `/etc/init.d/` or the `/etc/rc.d/` directory.

SysVinit systems configuration file - rc

- ❖ The **rc** script runs the scripts in a particular directory.
- ❖ The directory picked depends on the desired runlevel.
- ❖ Each runlevel has its own subdirectory in the **/etc/rc.d/** directory.

```
# ls /etc/rc.d/
```

```
init.d rc0.d rc2.d rc4.d rc6.d rc.sysinit rc rc1.d rc3.d rc5.d rc.local
```

- ❖ Files in the **/etc/rc.d/rc3.d** directory:

```
# ls -1F /etc/rc.d/rc3.d/
```

```
K01smolt@
```

```
K02avahi-dnsconfd@
```

```
[...]
```

```
S00microcode_ctl@
```

```
S04readahead_early@
```

```
[...]
```



Manage SysVinit Services

❖ **service** - run a System V init script

`service SCRIPT COMMAND [OPTIONS]`

✓ **--status-all** runs all init scripts, in alphabetical order, with the status command.

❖ Commonly used **service** utility commands

✓ start

✓ stop

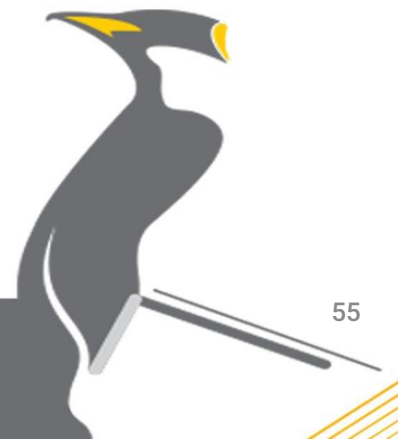
✓ status

✓ restart

✓ reload

STOPPING THE SYSTEM

Besides the various SysVinit and systemd commands you can use to shut down or reboot a system, there are a few additional utilities you can employ to enact these tasks no matter what system initialization your system uses.



Stopping the System

❖ **halt**: Stops all processes and shuts down the CPU.

✓ -p, --poweroff

✓ --reboot

❖ **poweroff**: Stops all processes, shuts down the CPU, and sends signals to the hardware to power down.

✓ --halt

✓ --reboot

❖ **reboot**: Stops all processes, shuts down the CPU, and then restarts the system.

✓ --halt

✓ -p, --poweroff

❖ **shutdown**: Stops all processes, shuts down the CPU, and sends signals to the hardware to power down.

Stopping the System

- ❖ When powering off a system, after the processes are stopped and the CPU is shut down, signals are sent to the hardware telling the various components to power down.
- ❖ For operating systems using Advanced Configuration and Power Interface (ACPI)–compliant chipsets, these are ACPI signals.
- ❖ These special communications are handled by the ACPI daemon, acpid.
- ❖ This daemon manages signals sent to various hardware devices via predefined settings for particular events, such as pressing the system's power button or closing a laptop system's lid.

Stopping the System

❖ **shutdown** - Halt, power-off or reboot the machine

shutdown [OPTIONS...] [TIME] [WALL...]

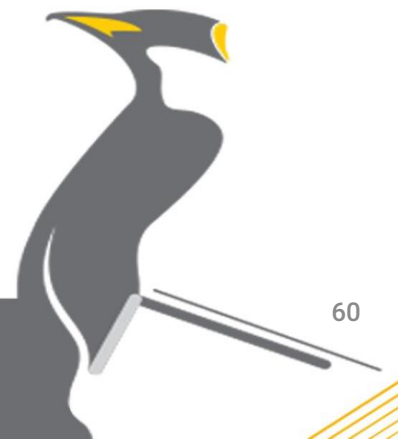
- ✓ **-H, --halt** Halt the machine.
- ✓ **-P, --poweroff** Power-off the machine (the default).
- ✓ **-r, --reboot** Reboot the machine.
- ✓ **--no-wall** Do not send wall message before halt, power-off, reboot.
- ✓ **-c** Cancel a pending shutdown.
- ✓ **TIME** now, hh:mm, +m

Stopping the System

- ❖ For any utility used to shut down the system, the processes are sent a SIGTERM signal.
- ❖ This allows the various running programs to close their files properly and gracefully terminate.
- ❖ However, in unusual cases, you may have a situation where a process refuses to shutdown.
 - ✓ You can use the `lsof -p PID` command to see if the running program has any files open.
 - ✓ If not, then you can attempt to use the `kill -9 PID` command.
 - ✓ However, always tread cautiously in these cases.

NOTIFYING THE USERS

When you perform any function that changes the system's state for logged-in users, it's a good idea to let them know ahead of time so that they can wrap up any work before being kicked out.



Notifying the Users

- ❖ `/etc/issue`: Contains text to be displayed on the tty terminal login screens (prior to logging into the system).
- ❖ `/etc/issue.net`: Contains logon screen messages for remote logins.
- ❖ `/etc/motd`: Called the Message of the Day file, contains text that is displayed after a user has logged into a tty terminal.
- ❖ `/bin/notify-send` (or `/usr/bin/notify-send`): Sends messages to a user employing the GUI but who is not logged into a tty terminal or does not have a GUI terminal emulator open.
- ❖ `/bin/wall` (or `/usr/bin/wall`): Sends messages (called wall messages) to users logged into a tty terminal or who have a GUI terminal emulator open and have their message status set to “yes.”
- ❖ `mesg`: Viewing your message status. (`mesg y` command to turn on messaging and `mesg n` to turn it off)

Notifying the Users

❖ By default, the systemctl utility will send a wall message when any of its following commands are issued:

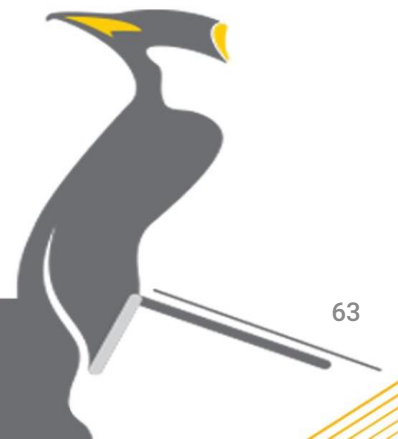
- ✓ emergency,
- ✓ halt,
- ✓ power-off,
- ✓ reboot,
- ✓ rescue.

❖ To prevent a wall message from being sent while using systemctl, include the **--no-wall** option in its command line.

VIRTUALIZING LINUX

When something is virtual, it does not physically exist but instead is simulated. In the information technology world, this simulation can apply to computer systems, which is accomplished through special software.

In this section, we'll take a look at the various types of simulations available, the terminology associated with them, as well as some of the various tools involved.




Looking at Virtual Machines

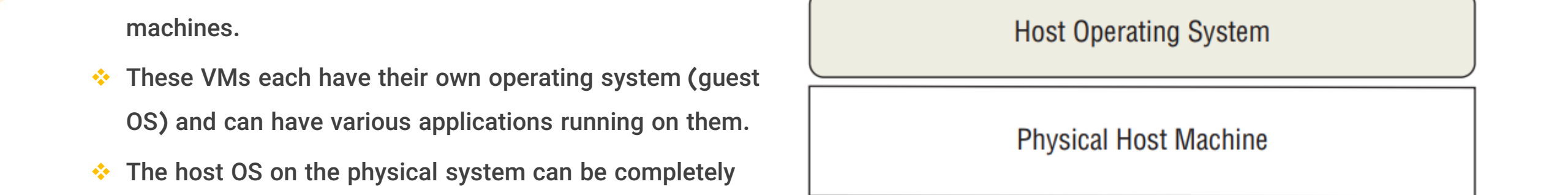
- ❖ Virtual machines (VMs) are simulated computer systems that appear and act as physical machines to their users.
- ❖ The process of creating these virtual machines is called virtualization.

Managing VMs

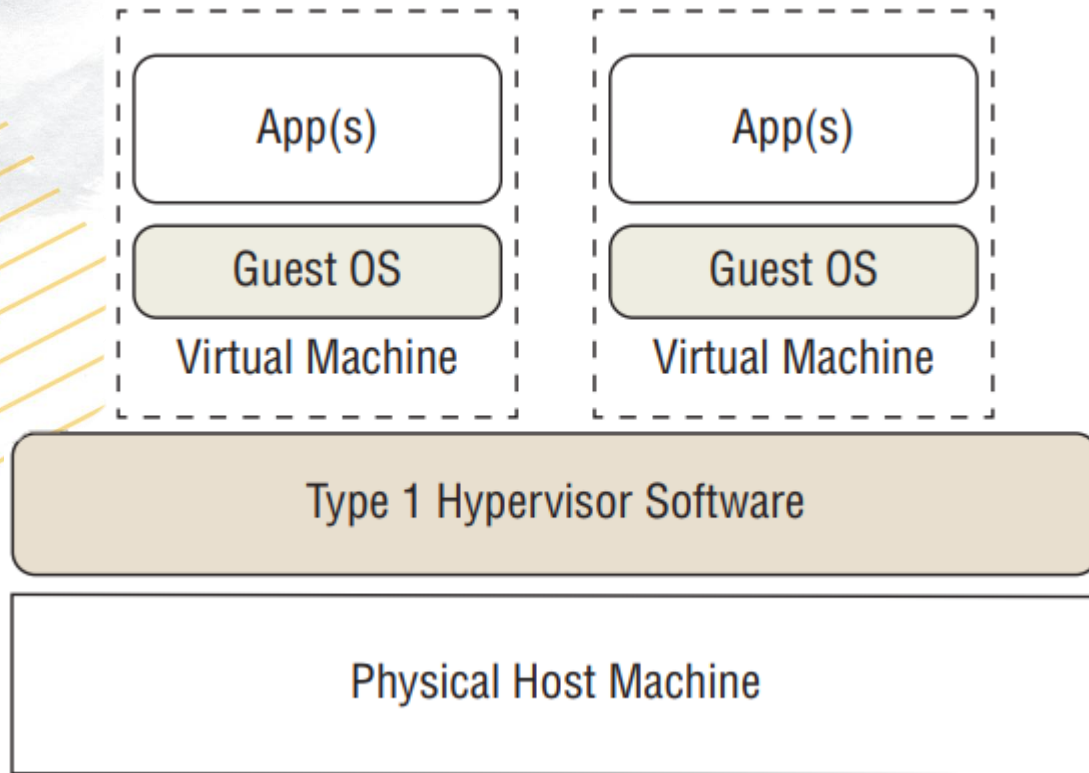
- ❖ The primary software tool used to create and manage VMs is a **hypervisor**, which has been historically called either a virtual machine monitor or a virtual machine manager (VMM).
- ❖ Hypervisors come in two basic flavors:
 - ✓ Type 1
 - ✓ Type 2

A Type 2 hypervisor

- ❖ A Type 2 hypervisor acts as a typical software application in that it interacts with the host's operating system.
 - ❖ However, its distinction lies in the fact that it provides one or more virtualized environments or virtual
- 
- The diagram illustrates the architecture of a Type 2 Hypervisor. It shows a single layer of 'Type 2 Hypervisor Software' (represented by a solid tan box) running on top of the host operating system. Above this layer, two separate 'Virtual Machine' environments are shown, each containing a 'Guest OS' (represented by dashed-line boxes). This configuration allows the hypervisor to manage multiple virtual machines, each with its own operating system, while sharing the underlying hardware resources of the host.



A Type 1 hypervisor



- ❖ A Type 1 hypervisor eliminates the need for the physical host's OS.
- ❖ This software runs directly on the physical system and due to this is sometimes called a bare-metal hypervisor.
- ❖ A few options include:
 - ✓ KVM
 - ✓ Xen
 - ✓ Hyper-V

Creating a Virtual Machine

❖ The following is a brief list of items that may need to be modified for a Linux clone:

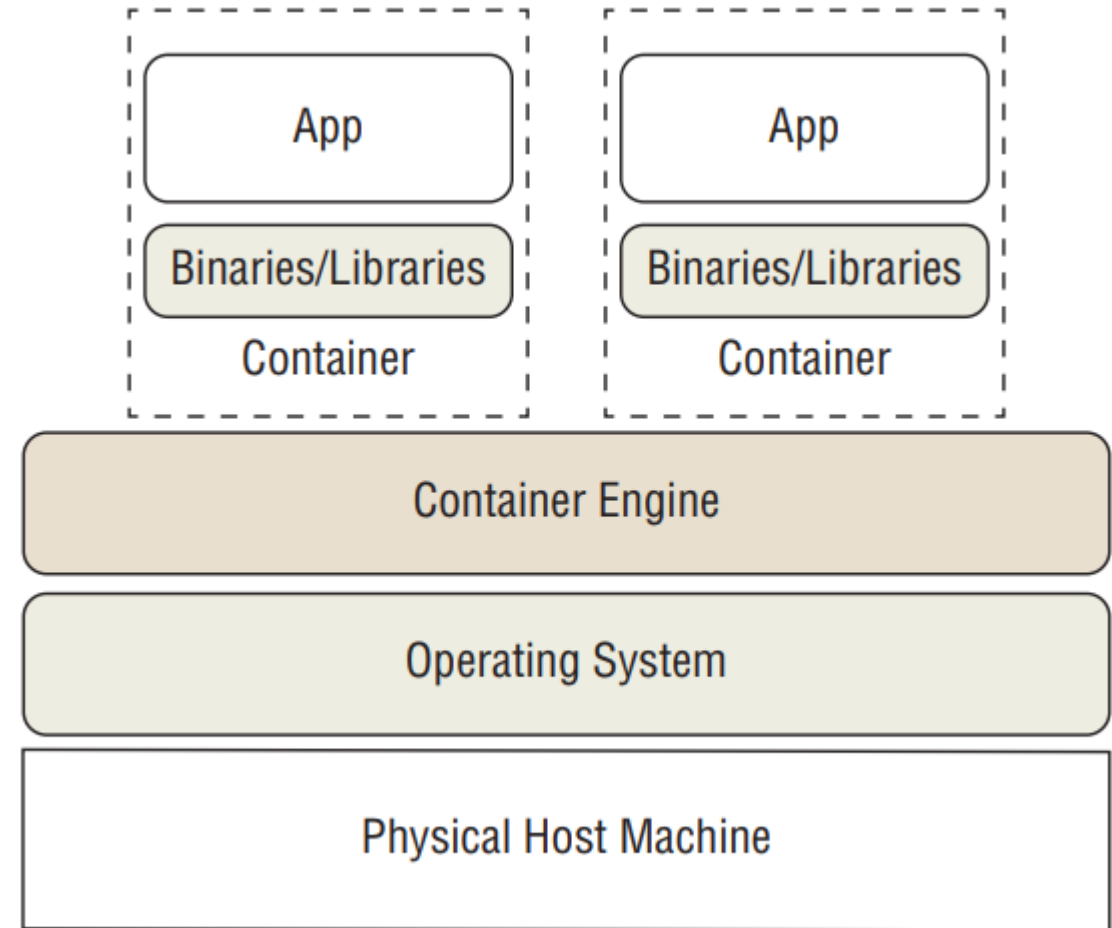
- ✓ Host name
- ✓ NIC MAC address
- ✓ NIC IP address, if using a static IP
- ✓ Machine ID
- ✓ Any items employing a universally unique identifier (UUID)
- ✓ Configuration settings on the clone that employ any item in this list

Creating a Virtual Machine

- ❖ Your system's machine ID is a unique hexadecimal 32-character identifier.
- ❖ The ID is stored in the `/etc/machine-id` file.
- ❖ D-Bus will use this ID, if its own machine ID file, `/var/lib/dbus/machine-id`, does not exist.
- ❖ Typically on modern distributions, the D-Bus machine ID file will not exist or will be symbolically linked to the `/etc/machine-id` file.
- ❖ Problems can ensue if you clone a machine and boot it so that the two machines share the same ID.
- ❖ These problems may include not being able to get an IP address if your network manager is configured to use the machine's ID instead of a NIC's MAC address for DHCP services.
- ❖ To prevent this problem, after you clone a VM, you'll need to address the duplicate machine ID.
- ❖ Typically, you can do this on the clone by performing the following steps:
 1. Delete the machine ID file: `rm /etc/machine-id`
 2. Delete the D-Bus ID file: `rm /var/lib/dbus/machine-id`
 3. Regenerate the ID: `dbus-uuidgen --ensure`
- ❖ Keep in mind that your distribution may require additional steps, such as linking the `/var/lib/dbus/machine-id` file to `/etc/machine-id`.

Understanding Containers

- ❖ Containers are virtual entities, but they are different from virtual machines and serve distinct purposes.
- ❖ Whereas a VM provides an entire guest operating system, a container's focus is typically on a single app, application stack, or environment. Instead of a hypervisor, a container is managed by a container engine.



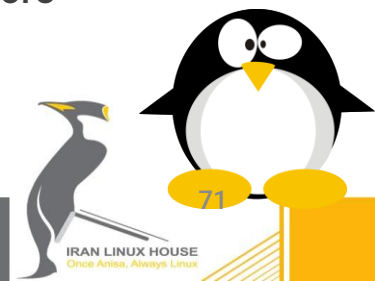
Two Container Types

Application Containers

- ❖ These containers focus on a single application, or an application stack, such as a web server.
- ❖ Application containers are heavily used in development and operations (DevOps).
- ❖ Software developers can modify their company's app in a newly created container.
- ❖ This same container, with the modified app, is then tested and eventually moved into production on the host machine.
- ❖ The old production container is destroyed.
- ❖ Using containers in this way eliminates production and development environment differences and provides little to no downtime for app users.
- ❖ Thus, containers are very popular in continuous software deployment environments.

Operating System Containers

- ❖ While containers are useful for developers, system admins can love them too.
- ❖ You can use a container that provides a fully functioning Linux OS space and is isolated from your host machine.
- ❖ Some in TechOps use containers to test their applications and needed libraries on various Linux distributions.
- ❖ Other system admins, prior to upgrading their host system distro, try out their environment on an upgraded Linux distribution.
- ❖ You can also employ VMs for these different evaluations, but containers provide a faster-to-deploy and more lightweight test area.



Looking at Infrastructure as a Service

- ❖ With the high expenses of maintaining hardware and providing the electricity to run servers as well as cool them, many companies are looking to cloud-based data centers.
- ❖ For virtualization, companies turn to Infrastructure as a Service (IaaS) providers.
- ❖ With IaaS, the provider furnishes not only the hypervisor but the data center(s) as well servers, disks, and networking infrastructure.
- ❖ Your virtualized environment is reached over the Internet.
- ❖ Many of these IaaS providers also offer various data center locations around the world, allowing you to pick the one closest to your customers.
- ❖ Often you can select additional data centers to run your VMs or containers if needed for performance or to act as backup locations should your chosen primary data center experience a disaster.

Cloud-Based Virtualized Environment

- ❖ When using a cloud-based virtualized environment, you should know a few additional terms that will assist in selecting a CSP.
- ❖ They are as follows:
- ❖ **Computing Instance**
 - ✓ A computing instance, sometimes called a cloud instance, is a single virtual machine or container running on a cloud platform. When an instance is started, this is called provisioning an instance.

Cloud-Based Virtualized Environment

❖ Cloud Type

- ✓ A CSP (Cloud Service Provider) will offer public, private, and hybrid clouds.
- ✓ Public clouds are just as they sound open to the public via the Internet or application programming interfaces (APIs).
- ✓ Access to their instances is controlled via virtual firewalls, gateways, and network routers.
- ✓ These clouds reside solely on the CSP's infrastructure.
- ✓ Private cloud instances are only accessible to those who own and manage the cloud instances.
- ✓ Access to these private cloud instances is often controlled via the same methods as public clouds.
- ✓ They too reside solely on the CSP's infrastructure.
- ✓ Hybrid clouds are interesting in that they are typically a combination of instances on the CSP's infrastructure as well as instances or physical computers at a company's local data center.
- ✓ Hybrid clouds are popular with organizations who do not need all the features (or the price) of a CSP's cloud infrastructure.

Cloud-Based Virtualized Environment

❖ Elasticity

- ✓ Elasticity allows an instance to automatically scale up or scale down, depending on current demand.
- ✓ This may mean that additional instances are created to meet bursts of traffic (called **horizontal** scaling), and when traffic wanes, then the instances are deprovisioned.
- ✓ It can also mean that additional resources are provided (or removed from) an instance to meet demand (called **vertical** scaling).

❖ Load Balancing

- ✓ Load balancing occurs when a virtualized environment has multiple instances.
- ✓ The current demand is spread across the multiple instances to provide balanced loads to each instance, instead of hitting one instance with a majority of the traffic

Cloud-Based Virtualized Environment

❖ Management Console or Portal

- ✓ To set up your instances and choose the various CSP features, many providers offer a web browser-based graphical utility called a management console or portal.
- ✓ Through these consoles, you can modify access to private and public clouds, choose storage types, start or stop instances, and perform monitoring of your running VMs or containers.

❖ Block and Object Storage

- ✓ **Block storage** is familiar to most system admins.
- ✓ Typically the underlying hardware is configured as disk drives in RAID configurations.
- ✓ Fixed size storage volumes can be permanently or temporarily attached as physical devices to instances.
- ✓ From there, they can be mounted and used as needed.
- ✓ However, if block storage is not mounted, it cannot be accessed.
- ✓ **Object storage** is different in that it can be accessed through the management console or through the web.
- ✓ Typically, the CSP provides data protection services for object storage, and you can store any kinds of data you desire.

Cloud-Based Virtualized Environment

❖ Remote Instance Access

- ✓ While the console or portal gives you the high-level view of your various cloud instances, there are times you need to log directly into an instance.
- ✓ A preferred method employs using OpenSSH to access your instance's IP address.
- ✓ But instead of using a username and password, many CSPs provide an SSH host key.
- ✓ This method furnishes a secured encrypted connection and access method to log directly into your remote cloud instances.