CS615 - Aspects of System Administration

Software Installation Concepts

Department of Computer Science
Stevens Institute of Technology
Jan Schaumann
jschauma@stevens.edu
https://www.cs.stevens.edu/~jschauma/615/
Down the stack we go

Consider a website on an AWS EC2 instance...
Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions
Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions
- which make various system calls
Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions
- which make various system calls
- which the kernel handles for the OS
Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions
- which make various system calls
- which the kernel handles for the OS
- which is running in a virtual machine
Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions
- which make various system calls
- which the kernel handles for the OS
- which is running in a virtual machine
- which is running on top of a hypervisor
Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions
- which make various system calls
- which the kernel handles for the OS
- which is running in a virtual machine
- which is running on top of a hypervisor
- which uses firmware to manage various components
Down the stack we go

Consider a website on an AWS EC2 instance...

...it might:

- require a web server
- require PHP, Perl, Ruby, Javascript, ...
- which uses generic library functions
- which make various system calls
- which the kernel handles for the OS
- which is running in a virtual machine
- which is running on top of a hypervisor
- which uses firmware to manage various components
- which is running on some hardware
...and back up again

Bringin up this web service might include...

- power on hardware
...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts
...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts
- domU is started
...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts
- domU is started
- guest OS kernel starts
...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts
- domU is started
- guest OS kernel starts
- kernel initializes (virtual) hardware
...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts
- domU is started
- guest OS kernel starts
- kernel initializes (virtual) hardware
- init(8) (or similar) starts
...and back up again

Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts
- domU is started
- guest OS kernel starts
- kernel initializes (virtual) hardware
- `init(8)` (or similar) starts
- system processes / daemons start
Bringin up this web service might include...

- power on hardware
- POST and other firmware initialization
- first stage boot loader
- second stage boot loader
- hypervisor kernel dom0 starts
- domU is started
- guest OS kernel starts
- kernel initializes (virtual) hardware
- `init(8)` (or similar) starts
- system processes / daemons start
- web server runs, binds network socket, serves content
Typical Boot Sequence

AMI BIOS (C) 2007 American Megatrends, Inc.
ASUS P5KPL ACPI BIOS Revision 0603
CPU: Intel(R) Pentium(R) Dual CPU EZ100 @ 2.00GHz
  Speed: 2.51 GHz  Count: 2

Press DEL to run Setup
Press F8 for BBS POPUP
DDR2-667 in Dual-Channel Interleaved Mode
Initializing USB Controllers .. Done.
3584MB OK

(C) American Megatrends, Inc.
64-0603-000001-00101111-82298B-Bearlake-A0820000-Y2KC
Typical Boot Sequence
Typical Boot Sequence
Typical Boot Sequence: BIOS and MBR

- first sector (512 bytes) of data storage device
- last two bytes contain signature 0x55 0xAA
- 64 bytes allocated for partition table (four possible partitions at 16 bytes each)
- 446 bytes for primary boot loader code
Recall HW1

```
# fdisk /dev/xbd0
fdisk: primary partition table invalid, no magic in sector 0
Disk: /dev/xbd0d

BIOS disk geometry:
cylinders: 1023, heads: 255, sectors/track: 63 (16065 sectors/cylinder)
total sectors: 69206016

Partition table:
0: <UNUSED>
1: <UNUSED>
2: <UNUSED>
3: <UNUSED>
Bootselector disabled.
No active partition.
```
Recall HW1

```bash
# fdisk /dev/xbd4
Disk: /dev/rxbd4d

BIOS disk geometry:
cylinders: 1023, heads: 255, sectors/track: 63 (16065 sectors/cylinder)
total sectors: 69206016

Partition table:
0: NetBSD (sysid 169)
   start 63, size 69205825 (33792 MB, Cyls 0-4307/221/10), Active
1: <UNUSED>
2: <UNUSED>
3: <UNUSED>
Bootselector disabled.
First active partition: 0
```
Typical Boot Sequence

<table>
<thead>
<tr>
<th>Ubuntu 8.04, kernel 2.6.24-16-generic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu 8.04, kernel 2.6.24-16-generic (recovery mode)</td>
</tr>
<tr>
<td>Ubuntu 8.04, memtest06+</td>
</tr>
</tbody>
</table>

Use the ↑ and ↓ keys to select which entry is highlighted. Press enter to boot the selected OS, 'e' to edit the commands before booting, or 'c' for a command-line.
Typical Boot Sequence

- Power-on Self-Test
- primary boot loader (e.g. BIOS, UEFI, Open Firmware / OpenBoot)
- transfer of execution to Master Boot Record or perform netbooting
- Second-stage boot loader (e.g. GRUB)
- load kernel
- kernel transfers control to init(8)

Note: in virtualized environments, some of these steps are skipped, repeated, or simulated.
Typical Boot Sequences

https://www.cs.stevens.edu/~jschauma/615/boot-sequence/

$ aws ec2 run-instances --instance-type t1.micro --image-id ami-569ed93c
$ id=$(aws ec2 describe-instances --query 'Reservations[].Instances[].InstanceId')
$ aws ec2 get-console-output --instance-id ${id} | more

Compare the dislabel on the /boot device to the output of df(1) on the mounted partition. What’s different?

Review the full console output; pay attention to the filesystem specific parts. Can you explain what’s happening?
Types of Software
Firmware
Firmware
Firmware
Incorrect configuration checksum;
Setting NVRAM parameters to default values.
Setting diag-switch? NVRAM parameter to true
Probing /sbus@1,f8000000 at 0,0 dma esp sd st le
Probing /sbus@1,f8000000 at 1,0 cgtthree
Probing /sbus@1,f8000000 at 2,0 Nothing there
Probing /sbus@1,f8000000 at 3,0 Nothing there

SPARCstation 2, Keyboard Present
ROM Rev. 2.9, 16 MB memory installed, Serial #1296.
Ethernet address 8:0:20:10:31:3, Host ID: 55000510.

Testing 16 megs of memory 14
Type b (boot), c (continue), or n (new command mode)
> n
Type help for more information
ok setenv diag-switch? false
diag-switch? = false
ok setenv selftest-megs 0
selftest-megs = 0
ok boot cdrom
Boot device: /sbus/esp@0,8000000/sd@6,0:c File and args:
>> NetBSD/sparc Secondary Boot, Revision 1.15
>> (builds@b3.netbsd.org, Tue Oct 31 08:41:58 UTC 2006)
Booting netbsd
1525520
Firmware
Kernel
Setting up new root fs
no fstab.sys, mounting internal defaults
Switching to new root and running init.
unmounting old /dev
unmounting old /proc
unmounting old /sys
INIT: version 2.86 booting
Welcome to Fedora
Press 'I' to enter interactive startup.
Setting clock: Fri Feb 11 19:17:31 EST 2011 [ OK ]
Starting udev: [ OK ]
Setting hostname localhost: [ OK ]
No devices found
Setting up Logical Volume Management: File descriptor 7 left open
   No volume groups found
   [ OK ]
Checking filesystems
Checking all file systems.
[/sbin/fsck.ext3 (1) -- /] fsck.ext3 -a /dev/sda1
myroot: clean, 51198/1966080 files, 470903/3932160 blocks
   [ OK ]
Remounting root filesystem in read-write mode: [ OK ]
Mounting local filesystems: mount: special device /dev/mapper/storageVG-storage
FS does not exist
System Software

```
$ ls /bin
[ df launchctl pwd tcsh
bash domainname link rcp test
cat echo ln rm unlink
chmod ed ls rmdir wait4path
cp expr mkdf sh

csh hostname mv sleep
date kill pax stty

dd ksh ps sync

$ ls -C /etc | head
6to4.conf master.passwd
CiscoSystemsVPNClient memberd.conf
Product.Catalog.JavaLiveUpdate moduli
Symantec.conf named.conf
afpovertcp.cfg nanorc
aliases networks
aliases.db newsyslog.conf
amavisd.conf newsyslog.d
apache2 notify.conf
asl.conf ntp-restrict.conf

$ 
```
Applications
Applications
Types of Software

- **Add-on or Third-Party Applications**
  - (web browser, database, programming languages, ...)

- **System Software**
  - (device drivers, loadable modules, libraries, ...)

- **Applications/Utilities**
  - (shell, common unix tools, daemons, compiler, ...)

- **Kernel**

- **Firmware**

- **Hardware**
...and then there are unikernels and containers.
Where do we put all these files?

Layout of filesystem *should* be standardized. Some UNIX versions adhere to these standards, some are strongly influenced by tradition.

`man hier`
File System Hierarchy

/ root directory of the system
/bin/ utilities used in both single and multi-user environments
/dev/ block, character and other special device files
/etc/ system configuration files and scripts
/lib/ dynamic linked libraries used by dynamic linked programs (such as those in /bin/ and /sbin/) that cannot rely upon /usr/lib/ being available.
/sbin/ system programs and administration utilities used in both single-user and multi-user environments
/tmp/ temporary files, usually a mfs(8) memory-based filesystem (the contents of /tmp are usually not preserved across a system reboot)
/usr/ contains the majority of the system utilities and files

bin/ common utilities, programming tools, and applications
lib/ archive, profiled, position independent archive, and shared libraries
/sbin/ system daemons and system utilities (normally executed by the super-user)
/share/ architecture-independent text files
Software Installation Concepts

Operating System Installation
OS Installation

NetBSD/amd64 6.1.3

This menu-driven tool is designed to help you install NetBSD to a hard disk, or upgrade an existing NetBSD system, with a minimum of work.

In the following menus type the reference letter (a, b, c, ...) to select an item, or type CTRL+N/CTRL+P to select the next/previous item.

The arrow keys and Page-up/Page-down may also work.

Activate the current selection from the menu by typing the enter key.

Thank you for using NetBSD!

```plaintext
NetBSD-6.1.3 Install System

- Install NetBSD to hard disk
- Upgrade NetBSD on a hard disk
- Re-install sets or install additional sets
- Reboot the computer
- Utility menu
- Config menu
- Exit Install System
```
OS Installation

Before installing, consider

- purpose of machine
  - choice of hardware
  - disk partitioning scheme
  - choice of filesystem
  - which software to install

- installation media
  - network installation
  - installation CD-ROMs
  - customized boot media
OS Installation

High-level overview:

- hardware identification, provisioning, and registration
- base OS installation
- installation of add-on applications
- initial minimum system configuration [*]
- system registration
- system restart

[*] system deployment $\cap$ system configuration

$\Rightarrow$ configuration management
Base OS Installation

General steps:

- boot from boot media (CD, network, ...)
- identify root device
- optionally identify additional devices
- create partition table / disklabel
- create filesystem(s)
- install MBR, bootblocks etc.
- install / copy / extract OS
- optionally add application software
- perform basic system configuration
- reboot
OS Installation

```bash
# fdisk -f -u 0 -s 169/63/4194241 /dev/rwd0d
# fdisk -f -c /usr/mdec/mbr /dev/rwd0d
# fdisk -f -a -0 /dev/rwd0d
# disklabel -e -I wd0

[...]  
4 partitions:
#  size  offset ftype [fsize bsize cpg/sgs]
a: 4194241 63 4.2BSD 0 0 0 # (Cyl. 0*-4161*)
c: 4194241 63 4.2BSD 0 0 0 # (Cyl. 0*-4161*)
d: 4194304 0 unused 0 0 0 # (Cyl. 0-4161*)

# /sbin/newfs -0 2 /dev/rwd0a
/dev/rwd0a: 2048.0MB (4194240 sectors) block size 16384,
fragment size 2048 using 12 cylinder groups of
170.67MB, 10923 blks, 21504 inodes.
super-block backups (for fsck_ffs -b #) at:
32, 349568, 699104, 1048640, 1398176, 1747712, 2097248, 2446784,

# mount -o async /dev/wd0a /mnt
# for pkg in base comp etc games man misc modules text kern GENERIC; do
tar zxf /i386/binary/sets/${pkg}.tgz -C /mnt done
# cp /mnt/usr/mdec/boot /mnt/boot
# /usr/sbin/installboot -v -o timeout=5 /dev/rwd0a 
   /mnt/usr/mdec/bootxx_ffsv2

File system: /dev/rwd0a
Primary bootstrap: /usr/mdec/bootxx_ffsv2
Boot options: timeout 5, flags 0, speed 9600, ioaddr 0, console pc

# cd /mnt/dev k & './MAKEDEV all'
# shutdown -r now
```
Post Installation
Post Installation
Post Installation
Hooray!

5 Minute Break
Software Installation Concepts

System Software vs. Third Party Software
What's what?

Diagram showing the layers of computer systems:
- Hardware
- Kernel
- Shells
- Utilities

Layers include:
- Hardware
- Kernel
- Shells: csh, ksh, sh, sort, ls
- Utilities: vi, cat, who, awk, cc, device interfaces, file system, multitasking
What's what?

[Images of various software logos: Python, Oracle, another logo, and PHP]
Types of Software

- Add-on or Third-Party Applications
  (web browser, database, programming languages, ...)

- System Software
  (device drivers, loadable modules, libraries, ...)

- Applications/Utilities
  (shell, common UNIX tools, daemons, compiler, ...)

- Kernel

- Firmware

- Hardware

Package Management
System Software vs. Third Party Software

Consider:

- OS upgrades vs. software upgrades
- location of configuration files
- duplicates or conflicting versions in the base system vs. the add-ons
- startup scripts, daemons
- location of third party software
- dependencies
- installation by hand and/or installation using a package manager
- proprietary third party software
Binary vs. Source installation

Benefits of binary installation:

- packaged by "vendor" → support, ease of installation
- faster
- uses less space
- may be only possibility
- able to integrate into your full OS image build
- may be possible to deploy across large numbers of hosts
Binary vs. Source installation

Disadvantages of binary installation:
- complex dependencies
- installation procedure may be cumbersome
- your OS may not be officially supported
- installation scripts may be busted
- limited control over where files are installed
- missing or not-needed features enabled
- you have to trust the package provider
Binary vs. Source installation

Benefits of source installation:
- full control over
  - installation location
  - compiler flags, optimization, enabled features
  - dependencies
- make things work even if your OS is not officially supported
- ability to patch source (features, security etc.)
- able to integrate into your full OS image build
Binary vs. Source installation

Disadvantages of source installation:

- complex dependencies
- may take time
- requires more detailed knowledge
- a lot of software is done poorly
- not all software is available in source form
- you have to trust the source code provider
Why use a Package Management System?

- easy and scalable installation of software
- automatic resolution of software dependencies
- package and file inventory

```
linux-lab$ dpkg -l
[...]
linux-lab$ dpkg -L tcpdump
[...]
linux-lab$ dpkg-query -S /usr/lib/libsqlite.so.0.8.6 /usr/bin/sqlite3
[...]
```
Why use a Package Management System?

- easy and scalable installation of software
- automatic resolution of software dependencies
- package and file inventory
- integration into OS
- package and file integrity checks

```
$ rpm -Va
[...]
missing /etc/pki/CA/private (Permission denied)
S.5..... c /etc/pki/tls/certs/ca-bundle.crt
.......T c /etc/libuser.conf
..?..... c /etc/tcsd.conf
missing c /etc/logrotate.d/syslog
[...]
```
Managing Security Patches and Software Upgrades

How many known vulnerabilities (unique CVEs and affected packages) exist in each of the Fedora and Debian instances?

debian$ sudo apt-get install debsecan
debian$ debsecan
debian$ sudo apt-get update
debian$ sudo apt-get upgrade
debian$ debsecan

defedora$ yum list-security
defedora$ yum info-security
Managing Security Patches and Software Upgrades

How many known vulnerabilities (unique CVEs and affected packages) exist in each of the Fedora and Debian instances?

debian$ sudo apt-get install debsecan
debian$ debsecan
debian$ sudo apt-get update
debian$ sudo apt-get upgrade
debian$ debsecan

defora$ yum list-security
defora$ yum info-security
defora$ sudo yum update
defora$ yum list-security

Excellent! Now what about all the stuff you installed that wasn’t packaged?
“What’s pip?”
“A python package manager”
“How do I install it?”
“easy_install pip”
“What’s easy_install?”
“A python package manager”
'What is Bower?'
"A package manager"
"How do I install it?"
"Use npm"
"What's npm?"
"A package manager"
"...."
Special Purpose Package Managers

Most programming languages or environments come with their own "package management" solutions, often integrating/mixing with a "build system".

- Common Lisp => quicklisp
- Go => go get
- NodeJS => npm
- Perl => CPAN
- Python => easy-install, pip, pants, setuptools, ...
- Ruby => gems, rvm, rake
- Scala => sbt

...
You don’t get to choose.

You routinely have to build from source and (re-)package your software.
 Dependencies, Integrity, and Trust

OS provider repositories:
- `yum update`/`yum install`
- `apt-get`

Language-specific community repositories:
- `gem install foo`
- `go get github.com/randomAccount/randomRepository`
- `npm install -g foo`
- `perl -MCPAN -e 'install Something::YouWant'`
- `pip install foo`

What could possibly go wrong?
Dependencies, Integrity, and Trust

Fun fact:

$ wget http://somewhere/some.tar.gz
$ tar zxf some.tar.gz
$ cd some
$ ./configure
$ make
$ sudo make install

is not inherently better than

$ curl http://somewhere/script.sh | sudo bash
Dependencies, Integrity, and Trust

Mirroring untrusted, unverified dependencies does not solve any of your problems.

Integrity verification is meaningless without assurance of trust.

Dependencies are called dependencies because you depend on them.

Dependency trust and integrity is recursive.
Dependencies, Integrity, and Trust

Mirroring untrusted, unverified dependencies does not solve any of your problems.

Integrity verification is meaningless without assurance of trust.

Dependencies are called dependencies because you depend on them.

Dependency trust and integrity is recursive.

*Remember Left-Pad!*
Left-Pad

function leftPad (str, len, ch) {
    str = str + ' ';
    // convert 'str' to a 'string'
    len = len - str.length;
    // 'len' is the 'pad'’s length now
    if (len <= 0) return str;
    // doesn’t need to pad
    if (!ch && ch !== 0) ch = ' '\n    // 'ch' defaults to ' ' ' \
    ch = ch + ' ';
    // convert 'ch' to a 'string' cuz it could be a number
    var pad = ' ';
    // 'pad' starts with an empty string
    while (true) {
        // loop
        if (len & 1) pad += ch;
        // add 'ch' to 'pad' if 'len' is odd
        len >>= 1;
        // divide 'len' by 2, ditch the remainder
        if (len) ch += ch;
        else break;
        // 'len' is 0, exit the loop
    }
    return pad + str;
    // pad 'str'!
HW #3

Package management basics.

Detailed homework assignment posted at
Links

Booting virtual machines:
- https://wiki.xen.org/wiki/PvGrub
- https://is.gd/JnD9jM
- https://is.gd/TAVCQF
- https://is.gd/EiOu6v
Links

NPM and LeftPad:
http://blog.npmjs.org/post/141577284765/kik-left-pad-and-npm
http://www.haneycodes.net/npm-left-pad-have-we-forgotten-how-to-program/
http://www.businessinsider.com/npm-left-pad-controversy-explained-2016-3
Links

- http://www.pathname.com/fhs/
- hier(7)
- your package managers’ manual pages
  - pkg_info(1)
  - pkginfo(1), pkgadd(1M)
  - rpm(1)
  - ...
- http://www.pkgsrc.org/