ProLUG – Unit 4 Operate Running Systems

Required Materials

Rocky or similar server

Root or sudo permissions

EXERCISES (Warmup to quickly run through your system and familiarize yourself)

1. cd ~
2. ls
3. mkdir unit4
4. mkdir unit4/test/round6 #this fails
5. mkdir -p unit4/test/round6 #this works, think about why? man (mkdir)
6. cd unit4
7. ps #read the man
8. ps –ef #what does this show differently?
9. ps –ef | grep –i root #what is the PID of the 4th line?
10. ps –ef | grep –i root | wc –l #what does this show you and why might it be useful?.
11. top #use q to exit. Inside top, use h to find commands you can use to toggle system info

Pre-LAB – Check to make sure we have a useful tool and where we get it from

1. Real quick check for a package that is useful.
   rpm –qa | grep –i iostat #should find nothing
2. Let's find what provides iostat by looking in the YUM (we'll explore more in later lab)
   dnf whatprovides iostat
   #should tell you that sysstat provides iostat
3. Let's check to see if we have it
   rpm –qa | grep –i sysstat
4. If you don't, lets install it
   dnf install sysstat
5. Re-check to verify we have it now
   rpm –qa | grep –I sysstat
   rpm –qi sysstat<version>
   iostat #we'll look at this more in a bit

Might be good to ensure you have vim on your system now too. This is the same procedure as above.
rpm –qa | grep –i vim
If it's there, good.
dnf install vim
If it's not, install it so you can use vimtutor later (if you need help with vi commands)

LAB

1. Gathering system information release and kernel information
    a. cat /etc/*release
    b. uname
    c. uname -a
    d. uname –r
       man uname to see what those options mean if you don't recognize the values
    e. rpm -qa | grep -i kernel #what is your kernel number? Highlight it (copy in putty)
    f. rpm –qi <kernel from earlier> #what does this tell you about your kernel? When was the kernel last updated? What license is your kernel released under?

2. Check the number of disks
    a. fdisk -l
    b. ls /dev/sd*
       When might this command be useful? What are we assuming about the disks for this to work? How many disks are there on this system? How do you know if it's a partition or a disk?
    c. pvs #what system are we running if we have physical volumes? What other things can we tell with vgs and lvs?
    d. Use pvdisply, vgdisplay, and lvdisplay to look at your carved up volumes. Thinking back to last week's lab, what might be interesting from each of those?
    e. Try a command like lvdisplay | egrep "Path|Size" and see what it shows. Does that output look useful? Try to egrep on some other values "separate with | between search items.

    Check some quick disk statistics

    f. iostat –d
    g. iostat –d 2 #Wait for a while, then use crtl + c to break. What did this do? Try changing this to a different number.

h. iostat –d 2 5 #Don't break this, just wait. What did this do differently? Why might this be useful?

3. Check the amount of RAM
    a. cat /proc/meminfo
    b. free
    c. free –m
    What do each of these commands show you? How are they useful?

4. Check the number of processors and processor info
    a. cat /proc/cpuinfo
    What type of processors do you have? Google to try to see when they were released. Look at the flags. Sometimes when compiling these are important to know. This is how you check what execution flags your processor works with.
    b. cat /proc/cpuinfo | grep proc | wc –l
    Does this command accurately count the processors?

    Check some quick processor statistics
    c. iostat –c
    d. iostat –c 2 #Wait for a while, then use crtl + c to break. What did this do? Try changing this to a different number.
    e. iostat –c 2 5 #Don't break this, just wait. What did this do differently? Why might this be useful?

    Does this look familiar? To what we did earlier with iostat?

5. Check the system uptime
    a. uptime
    b. man uptime
    Read the man for uptime and figure out what those 3 numbers represent. Referencing this server, do you think it is under high load? Why or why not?

6. Check who has recently logged into the server and who is currently in
    a. last
    Last is a command that outputs backwards. (Top is most recent). So it is less than useful without using the more command.
    b. last | more
    Were you the last person to log in? Who else has logged in today?
    c. w
    d. who
    e. whoami

how many other users are on this system? What does the pts/0 mean on google?

7. Check running processes and services
    a. ps –aux | more
    b. ps –ef | more
    c. ps –ef | wc –l
    d. Try to use what you've learned to see all the processes owned by your user
    e. Try to use what you've learned to count up all of those processes owned by your user

8. Looking at system usage (historical)

    a. Check processing for last day
       sar | more
    b. Check memory for the last day
       sar –r | more
       Sar is a tool that shows the 10 minute weighted average of the system for the last day. Sar is tremendously useful for showing long periods of activity and system load. It is exactly the opposite in it's usefulness of spikes or high traffic loads. In a 20 minute period of 100% usage a system may just show to averages times of 50% and 50%, never actually giving accurate info. Problems typically need to be proactively tracked by other means, or with scripts, as we will see below.

       Sar can also be run interactively. Run the command `yum whatprovides sar` and you will see that it is sysstat. You may have guessed that sar runs almost exactly like iostat.

       Try the same commands from earlier, but with their interactive information

       sar 2 #crtl + c to break
       sar 2 5

       or

       sar –r 2
       sar –r 2 5

    c. Check sar logs for previous daily usage
       cd /var/log/sa/
       # ls
       sa01 sa02 sa03 sa04 sa05 sar01 sar02 sar03 sar04

       sar –f sa03 | head

sar –r –f sa03 | head
#should output just the beginning of 3 July (whatever month you're in). Most Sar data is kept for just one month but is very configurable. Read man sar for more info.

d. Sar logs are not kept in a readable format, they are binary. So if you needed to dump all the sar logs from a server, you'd have to output it to a file that is readable. You could do something like this:
    i. Gather information and move to the right location
    cd /var/log/sa
    pwd
    ls
    We know the files we want are in this directory and all look like this sa*
    ii. Build a loop against that list of files
    for file in `ls /var/log/sa/sa??`; do echo "reading this file $file"; done
    iii. Execute that loop with the output command of sar instead of just saying the filename
    for file in `ls /var/log/sa/sa?? | sort -n`; do sar -f $file ; done
    iv. But that is too much scroll, so let's also send it to a file for later viewing
    for file in `ls /var/log/sa/sa?? | sort -n`; do sar -f $file | tee -a /tmp/sar_data_`hostname`; done
    v. Let's verify that file is as long as we expect it to be:
    ls –l /tmp/sar_data*
    cat /tmp/sar_data_<yourhostname> | wc –l

    Is it what you expected? You can also visually check it a number of ways
    cat /tmp/<filename>
    more /tmp/<filename>

Exploring Cron

Your system is running the cron daemon. You can check with:

ps –ef | grep –i cron

systemctl status crond

This is a tool that wakes up between the 1st and 5th second of every minute and checks to see if it has any tasks it needs to run. It checks in a few places in the system for these tasks. It can either read from a crontab or it can execute tasks from files found in the following locations.

/var/spool/cron is one location you can ls to check if there are any crontabs on your system

The other locations are directories found under

ls -ld /etc/cron*

These should be self-explanatory in their use.

If you want to see if the user you are running has a crontab use the command ` crontab –l `. If you want to edit, (using your default editor, probably vi) use `crontab –e `

We'll make a quick crontab entry and I'll point you here (https://en.wikipedia.org/wiki/Cron) if you're interested in learning more.

Crontab format looks like this picture so let's do these steps

```
┌──────────── minute (0 - 59)
│ ┌────────── hour (0 - 23)
│ │ ┌──────── day of month (1 - 31)
│ │ │ ┌────── month (1 - 12)
│ │ │ │ ┌──── day of week (0 - 6) (Sunday to Saturday;
│ │ │ │ │                         7 is also Sunday on some systems)
│ │ │ │ │
│ │ │ │ │
* * * * *  command to execute
```

1. crontab –e
2. Add this line (using vi commands – Revisit vimtutor if you need help with them)
   * * * * * echo 'this is my cronjob runnig at' `date` | wall
3. Verify with crontab –l
4. Wait to see if it runs and echos out to wall.
5. cat /var/spool/cron/root to see that it is actually stored where I said it was.
6. This will quickly become very annoying, so I recommend removing that line, or commenting it out (#) from that file.

I can change all kinds of things about this to execute at different times. The one above, we executed every minute through all hours, of every day, of every month.

We could also have done some other things:

Every 2 minutes (divisible by any number you need):

    */2 * * * *

The first and 31st minute of each hour:

    1,31 * * * *

The first minute of every 4th hour:

1 */4 * * *

There's a lot there to explore, I recommend looking into https://en.wikipedia.org/wiki/Cron or http://www.tldp.org/LDP/lame/LAME/linux-admin-made-easy/using-cron.html for more information.

That's all for this week's lab. There are a lot of uses for all of these tools above. Most of what I've shown here, I'd liken to showing you around a tool box. Nothing here is terribly useful in itself, the value comes from knowing the tool exists and then being able to properly apply it to the problem at hand. I hope you enjoyed this lab.