Linux Process Management with Ansible

By: Er. Vikas Nehra (M. Tech, B. Tech), Experience: 15 + Years

Session - 29 Agenda:
1. Linux Process Management with Ansible

Linux Process Management:

Process:
A process is compiled source code that is currently running on the system.

```
# date
# sleep 500
# ps
# ps | grep sleep
# pidof sleep
# sleep 500 &
# pgrep sleep
# ps | grep sleep
# jobs
# fg 1
```

PID:
All processes have a process id or PID.

```
# ps
# pidof sleep
# pgrep sleep
```

PPID:
Every process has a parent process (with a PPID). The child process is often started by the parent process.

```
# pstree
```

INIT (Systemd):
The init process always has process ID 1.
The init process is started by the kernel itself so technically it does not have a parent process.

```
# ps -C systemd
# pidof systemd
```

Kill:
When a process stops running, the process dies, when you want a process to die, you kill it.

```
# sleep 500 &
# jobs
# kill 36938
# jobs
# fg 1
```

Daemon:
Processes that start at system startup and keep running forever are called daemon processes or daemons. These daemons never die.

```
# ps -C sshd
# pgrep sshd
```

Zombie:
When a process is killed, but it still shows up on the system, then the process is referred to as zombie. You cannot kill zombies, because they are already dead.

$$ and $PPID:
The $$ variable will hold your current process ID, and $PPID contains the parent PID.

```
#  echo $$ $PPID
```

Pidof:
You can find all process id's by name using the pidof command.

```
# pidof sleep
```

Parent and child:
Processes have a parent-child relationship. Every process has a parent process.

```
$ echo $$ $PPID
36714 36709
$ bash
$ echo $$ $PPID
36967 36714
$ exit
exit
$ echo $$ $PPID
36714 36709
```

Ps:
One of the most common tools on Linux to look at processes is ps.

```
# ps
# ps fx
# ps -ef
# ps -aux
```

Pgrep:
Similar to the ps -C, you can also use pgrep to search for a process by its command name.

```
$ sleep 1000 &
[1] 37010
$ pgrep sleep
37010
$ ps -C sleep
    PID TTY          TIME CMD
  37010 pts/2    00:00:00 sleep
$ pgrep -l sleep
37010 sleep
```

Top:
The top tool can order processes according to cpu usage or other properties. You can also kill processes from within top. Press h inside top for help.

```
# top
```

Kill:
The kill command will kill (or stop) a process.

List signals:
Running processes can receive signals from each other or from the users. You can have a list of signals by typing kill -l, that is a letter l, not the number 1.

```
# kill -l
```

Kill -1 (SIGHUP):
It is common on Linux to use the first signal SIGHUP (or HUP or 1) to tell a process that it should re-read its configuration file.

```
# ps -C firewalld
   PID TTY         TIME CMD
   998 ?       00:00:00 firewalld

# firewall-cmd --permanent --add-service=http
success

# firewall-cmd --list-services
cockpit dhcpv6-client ssh

# kill -1 998

# firewall-cmd --list-services
cockpit dhcpv6-client http ssh
```

Kill -15 (SIGTERM):
The SIGTERM signal is also called a standard kill. Whenever kill is executed without specifying the signal, a kill -15 is assumed.

```
# sleep 100 &
[1] 2038

# sleep 200 &
[2] 2039

# jobs
[1]-  Running                 sleep 100 &
[2]+  Running                 sleep 200 &

# kill 2038

# jobs
[1]-  Terminated              sleep 100
[2]+  Running                 sleep 200 &

# kill -15 2039

# jobs
[2]+  Terminated              sleep 200
```

Kill -9 (SIGKILL):

The SIGKILL is different from most other signals in that it is not being sent to the process, but to the Linux kernel. A kill -9 is also called a sure kill. The kernel will shoot down the process. As a developer you have no means to intercept a kill -9 signal.

```
# sleep 600 &
[1] 2044

# jobs
[1]+  Running              sleep 600 &

# kill -9 2044

# jobs
[1]+  Killed               sleep 600
```

SIGSTOP:

A running process can be suspended when it receives a SIGSTOP signal. This is the same as kill -19 on Linux, but might have a different number in other Unix systems.

SIGCONT:

A suspended process does not use any cpu cycles, but it stays in memory and can be re-animated with a SIGCONT signal (kill -18 on Linux).

```
# sleep 700 &
[1] 2055

# jobs
[1]+  Running              sleep 700 &

# kill -19 2055

# jobs
[1]+  Stopped              sleep 700

# kill -18 2055

# jobs
[1]+  Running              sleep 700 &
```

Pkill:

You can use the pkill command to kill a process by its command name.

```
# sleep 1000 &
[2] 2056

# pkill sleep
[1]-  Terminated           sleep 700
[2]+  Terminated            sleep 1000
```

Killall:

The killall command will send a signal 15 to all processes with a certain name until you don't mention the signal with killall.

```
# sleep 1100 &
[1] 2069
# sleep 1200 &
[2] 2070

# jobs
[1]-  Running              sleep 1100 &
[2]+  Running              sleep 1200 &

# killall sleep
[1]-  Terminated           sleep 1100
[2]+  Terminated           sleep 1200

# sleep 1100 &
[1] 2072

# sleep 1200 &
[2] 2074

# jobs
[1]-  Running              sleep 1100 &
[2]+  Running              sleep 1200 &

# killall -9 sleep
[1]-  Killed               sleep 1100
[2]+  Killed               sleep 1200
```

Top:
Inside top the k key allows you to select a signal and pid to kill.

Priority and nice values:
Priority value: The priority value is the process's actual priority which is used by the Linux kernel to schedule a task. In Linux system priorities are 0 to 139 in which 0 to 99 for real-time and 100 to 139 for users.

Nice value: Nice values are user-space values that we can use to control the priority of a process. The nice value range is -20 to +19 where -20 is highest, 0 default and +19 is lowest.

The relation between nice value and priority is:
Priority_value = Nice_value + 20

To see how these works together let us take a process that takes a lot of processing power continuously. I'll use a shell script (infinite.sh) which has an infinite loop in it to demonstrate how this works.

```
# vim infinite.sh
#! /bin/bash
for (( ; ; ))
do
continue
done
```

Linux Process Management with Ansible

By: Er. Vikas Nehra (M. Tech, B. Tech), Experience: 15 + Years

```
# chmod +x infinite.sh

# ./infinite.sh &

# top
   PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  2088 root      20   0  222520   1444   1292 R  95.0   0.1   0:05.07 infinite.sh

# sh infinite.sh &
[2] 2092

# jobs
[1]-  Running                 ./infinite.sh &
[2]+  Running                 sh infinite.sh &

# top
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  2088 root      20   0  222520   1444   1292 R  93.7   0.1   1:43.81 infinite.sh
  2092 root      20   0  222520   1472   1320 R  93.7   0.1   0:22.62 sh

# nice -n 10 sh infinite.sh &
[3] 2095

# jobs
[1]   Running                 ./infinite.sh &
[2]-  Running                 sh infinite.sh &
[3]+  Running                 nice -n 10 sh infinite.sh &

# top
PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  2088 root      20   0  222520   1444   1292 R  93.7   0.1   3:24.20 infinite.sh
  2095 root      30  10  222520   1460   1304 R  93.7   0.1   1:07.32 sh
  2092 root      20   0  222520   1472   1320 R  93.4   0.1   2:02.77 sh

# renice -5 2088
2088 (process ID) old priority 0, new priority -5

# top
   PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  2088 root      15  -5  222520   1444   1292 R  93.7   0.1   4:44.38 infinite.sh

# renice -n -8 -p 2092
2092 (process ID) old priority 0, new priority -8

# top
   PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  2095 root      30  10  222520   1460   1304 R  93.7   0.1   6:16.23 sh
  2106 root       2 -18  222520   1444   1292 R  93.4   0.1   0:05.81 infinite.sh
```

Lower will be the PR value, higher will be the CPU time for the process.

You can also use renice to set nice values to all processes by a user using.

```
# renice -n nice_val -u [user]
# renice -n 10 -u vikasnehra
```

Normal users can attribute a nice value from zero to 20 to processes they own. Only the root user can use negative nice values. Be very careful with negative nice values, since they can make it impossible to use the keyboard or ssh to a system. Important to remember is to always make less important processes nice to more important processes. Using negative nice values can have a severe impact on a system's usability.

Linux Process Management with Ansible:
Linux process management can be easily done by using ansible. There are two methods of managing processes on the managed nodes with ansible:
1. Ansible Ad-hoc Commands
2. Ansible Playbooks

1. Ansible Ad-hoc Commands:
We can manage a process on the managed node by using ansible ad-hoc command. Let’s take an example of httpd service to do the process management using ansible ad-hoc commands.

```
$ ansible node1 -m command -a 'sudo dnf install -y httpd'
$ ansible node1 -m command -a 'sudo systemctl start httpd'
$ ansible node1 -m command -a 'pgrep httpd'
$ ansible node1 -m command -a 'sudo pkill httpd'
$ ansible node1 -m command -a 'pgrep httpd'
$ ansible node1 -m command -a 'sudo nice -5 systemctl start httpd'
$ ansible node1 -m command -a 'pidof httpd'
$ ansible node1 -m command -a 'ps -C httpd'
$ ansible node1 -m command -a 'pgrep httpd'
$ ansible node1 -m command -a 'ps -elf'
$ ansible node1 -m command -a 'ps -elf' | grep httpd
$ ansible node1 -m command -a 'sudo renice +5 37561'
$ ansible node1 -m command -a 'ps -elf' | grep httpd
$ ansible node1 -m command -a 'sudo kill -9 37561'
```

```
# vim infinite.sh
#! /bin/bash
for (( ; ; ))
do
continue
done
```

```
$ ansible node1 -m copy -a 'src=/home/vikasnehra/infinite.sh dest=/tmp/'
$ ansible node1 -m command -a 'sh /tmp/infinite.sh &' &
$ ansible node1 -m shell -a 'ps -elf' | grep infinite
$ ansible node1 -m command -a 'sudo renice +5 69302'
$ ansible node1 -m shell -a 'ps -elf' | grep infinite
$ ansible node1 -m command -a 'sudo kill -9 69302'
$ ansible node1 -m shell -a 'ps -elf' | grep infinite
```

2. Ansible Playbooks:

Ansible playbooks are always the better option to do the automation in an effective manner.

Example1:

Let's create an ansible playbook to start and manage the processes on the node1.

```
$ vim process.yml
---
- name: process management using ansible playbook
  hosts: node1
  become: yes
  tasks:
    - name: copying the infinite loop script file to the managed node.
      copy:
        src: /home/vikasnehra/infinite.sh
        dest: /home/vikasnehra/infinite.sh
        owner: vikasnehra
        group: vikasnehra
        mode: '0755'
    - name: start the process by running the script file in the background.
      shell: nohup sh /home/vikasnehra/infinite.sh &
...
```

Execute the playbook to start the sleep process on node1 for 1000 seconds.

```
$ ansible-playbook process.yml
```

Execute the ansible ad-hoc command to verify the same.

```
$ ansible node1 -m command -a 'ps -few' | grep infinite
$ ansible node1 -m command -a 'ps -elf' | grep infinite
```

Managing a process by running one command within other.

```
$ sleep 1000 &
[2] 2872

$ ps -few | grep sleep
vikasne+   2770   2202  0 12:05 pts/0   00:00:00 sleep 1000
vikasne+   2872   2202  0 12:09 pts/0   00:00:00 sleep 1000
vikasne+   2874   2202  0 12:10 pts/0   00:00:00 grep --color=auto sleep

$ ps -few | grep sleep | awk '{print $2}'
2770
2872
2876

$ ps -few | grep sleep | awk '{print $2}' | head -1
2770

$ kill -9 `ps -few | grep sleep | awk '{print $2}' | head -1`

$ ps -few | grep sleep | awk '{print $2}' | head -1
2872
```

```
[1]-  Killed              sleep 1000

$ sleep 1000 &
[3] 2893

$ pgrep sleep
2872
2893

$ kill -9 `pgrep sleep | head -1`
[2]-  Killed              sleep 1000
```

To manage a process with ansible playbook we would require a variable which can hold the process id of the process which we want to manage. So, we have to capture and store the process id in a variable and we can print the information by using debug module if required.

```
$ vim process.yml
---
- name: process management using ansible playbook
  hosts: node1
  become: yes
  tasks:
    - name: copying the infinite loop script file to the managed node.
      copy:
         src: /home/vikasnehra/infinite.sh
         dest: /home/vikasnehra/infinite.sh
         owner: vikasnehra
         group: vikasnehra
         mode: '0755'
    - name: start the process by running the script file in the background.
      shell: nohup sh /home/vikasnehra/infinite.sh &
    - name: Get running processes list from remote host
      ignore_errors: yes
      shell: "ps -few | grep infinite | awk '{print $2}' | head -1"
      register: running_processes
    - name: Printing the process ID of the running process
      debug:
        msg: "{{ running_processes }}"
...
```

Execute the playbook to see the process ID.

```
$ ansible-playbook process.yml
```

Execute the ansible ad-hoc command to verify the same.

```
$ ansible node1 -m command -a 'ps -few' | grep infinite
$ ansible node1 -m command -a 'ps -elf' | grep infinite
```

Let’s change the process priority of the first infinite process which is already running on node1.

```
$ vim process.yml
---
- name: process management using ansible playbook
```

```
  hosts: node1
  become: yes
  tasks:
    - name: copying the infinite loop script file to the managed node.
      copy:
        src: /home/vikasnehra/infinite.sh
        dest: /home/vikasnehra/infinite.sh
        owner: vikasnehra
        group: vikasnehra
        mode: '0755'
    - name: start the process by running the script file in the background.
      shell: nohup sh /home/vikasnehra/infinite.sh &
    - name: Get running processes list from remote host
      ignore_errors: yes
      shell: "ps -few | grep infinite | awk '{print $2}' | head -1"
      register: running_processes
    - name: Printing the process ID of the infinite process
      debug:
        msg: "{{ running_processes }}"
    - name: Changing the priority of the infinite process
      shell: "renice +5 {{ running_processes.stdout }}"
...
```

Execute the playbook.

```
$ ansible-playbook process.yml
```

Execute the ansible ad-hoc command to verify the same.

```
$ ansible node1 -m command -a 'ps -elf' | grep infinite
```

Also verify the same by running the top command on the node1 machine.

```
[vikasnehra@node1 ~]$ top
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
79655 root      25   5  222520   1416   1264 R  18.8   0.1   9:09.61 sh
```

You can see that for the first infinite process CPU time value is less as compared to others. Change the nice value of the first infinite process using ansible ad-hoc command and see the changes in the CPU time in the output of top command.

```
$ ansible node1 -m command -a 'sudo renice -10 79655'
```

Now, this process will use more CPU time as compared to others.

```
[vikasnehra@node1 ~]$ top
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  79655 root     10 -10  222520   1416   1264 R  86.8   0.1   8:24.82 sh
```

Let’s kill all the infinite processes using ansible playbook.

```
$ vim kill.yml
---
- name: process management using ansible playbook
  hosts: node1
  become: yes
  tasks:
    - name: Get running processes list from remote host
      ignore_errors: yes
      shell: "ps -few | grep infinite | awk '{print $2}'"
      register: running_processes
    - name: Printing the process IDs of the infinite processes
      debug:
        msg: "{{ running_processes }}"
    - name: Kill all the running infinite processes
      ignore_errors: yes
      shell: "kill {{ item }}"
      with_items: "{{ running_processes.stdout_lines }}"
...
```

Execute the playbook to kill all the running infinite processes.

$ ansible-playbook kill.yml

Execute the ansible ad-hoc command to verify the same or using the top command on node1.

$ ansible node1 -m command -a 'ps -elf' | grep infinite

Example2:

Let’s consider another example of httpd process, manage the processes now using ansible playbook for the httpd processes. (This is for the educational purpose only, please don’t perform this on the production machines.)

$ vim process_httpd.yml

```
---
- name: process management using ansible playbook
  hosts: node1
  become: yes
  tasks:
    - name: installing the httpd packages on the managed node
      yum:
        name: httpd
        state: latest

    - name: starting the httpd service on the managed node
      service:
        name: httpd
        state: started

    - name: Get running processes list from remote host
      ignore_errors: yes
      shell: "ps -few | grep httpd | awk '{print $2}'"
      register: running_processes

    - name: Printing the process IDs of the httpd processes
      debug:
        msg: "{{ running_processes }}"
...
```

Execute the playbook.

```
$ ansible-playbook process_httpd.yml
```

Verify the same you using the ansible ah-hoc command.

```
$ ansible node1 -m command -a 'ps -few' | grep httpd
```

Let's manage and kill the running httpd processes.

```
$ vim kill_httpd.yml
---
- name: process management using ansible playbook
  hosts: node1
  become: yes
  tasks:
    - name: Get running processes list from remote host
      ignore_errors: yes
      shell: "ps -few | grep httpd | awk '{print $2}'"
      register: running_processes

    - name: Printing the process IDs of the httpd processes
      debug:
        msg: "{{ running_processes }}"

    - name: Kill the running httpd processes
      ignore_errors: yes
      shell: "kill {{ item }}"
      with_items: "{{ running_processes.stdout_lines }}"

    - wait_for:
        path: "/proc/{{ item }}/status"
        state: absent
      with_items: "{{ running_processes.stdout_lines }}"
      ignore_errors: yes
      register: crunchify_processes

    - name: Force kill stuck processes
      ignore_errors: yes
      shell: "kill -9 {{ item }}"
      with_items: "{{ crunchify_processes.results | select('failed') | map(attribute='item') | list
}}"
...
```

Execute the playbook.

```
$ ansible-playbook process_httpd.yml
```

Verify the same you using the ansible ah-hoc command.

```
$ ansible node1 -m command -a 'ps -few' | grep httpd
```

Thank You