

```
TASK [Disable root login via SSH] ****
changed: [servera.lab.example.com]

RUNNING HANDLER [Restart sshd] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com    : ok=7      changed=4      unreachable=0      failed=0
```

- 9. 以 user1 用户身份，使用 SSH 登录 servera 服务器。登录后，使用 sudo su - 命令将身份切换为 root 用户。

9.1. 以 user1 用户身份使用 SSH 并登录 servera 服务器。

```
[student@workstation system-users]$ ssh user1@servera
Activate the web console with: systemctl enable --now cockpit.socket

[user1@servera ~]$
```

9.2. 将身份切换为 root 用户。

```
[user1@servera ~]$ sudo su -
Last login: Wed Dec 19 05:39:53 EST 2018 on pts/0
root@servera ~]#
```

9.3. 从 servera 服务器注销。

```
[root@servera ~]$ exit
logout
[user1@servera ~]$ exit
logout
Connection to servera closed.
[student@workstation system-users]$
```

- 10. 尝试直接以 root 用户身份登录 servera 服务器。此步骤应该会失败，因为 SSH 配置已修改为不允许 root 用户直接登录。

10.1. 从 workstation，以 root 用户身份使用 SSH 登录 servera 服务器。

```
[student@workstation system-users]$ ssh root@servera
root@servera's password: redhat
Permission denied, please try again.
root@servera's password:
```

这确认 SSH 配置拒绝 root 用户直接访问系统。

完成

在 workstation 上，运行 lab system-users finish 脚本来清理本练习中创建的资源。

```
[student@workstation ~]$ lab system-users finish
```

本引导式练习到此结束。

管理引导过程和调度的进程

培训目标

学完本节后，您应能够管理服务启动，使用 at、cron 和 systemd 调度进程，重新引导，以及控制受管主机上的默认启动目标。

使用 AT 模块进行调度

通过 **at** 模块来进行一次性快速调度。您可以将作业创建为在将来的某个时间运行，它将等待到该时间执行。此模块附带六个参数。分别是 command、count、script_file、state、unique 和 units。

at 模块示例：

```
- name: remove tempuser.
  at:
    command: userdel -r tempuser
    count: 20
    units: minutes
    unique: yes
```

参数

参数	选项	注释
command	Null	计划运行的命令。
count	Null	单位数。（必须与 units 一起运行）
script_file	Null	要在将来执行的现有脚本文件。
state	absent、present	添加或删除命令或脚本的状态。
unique	yes、no	如果作业已在运行，则不会再次执行。
units	minutes/hours/days/weeks	时间名称。

使用 CRON 模块附加命令。

在设置作业调度任务时，可使用 **cron** 模块。cron 模块将命令直接附加到您指定的用户的 crontab 中。

cron 模块示例：

```
- cron:
  name: "Flush Bolt"
  user: "root"
  minute: 45
  hour: 11
  job: "php ./app/nu cache:clear"
```

此 play 在清空 Bolt 缓存后立即使用公司的 cache:clear 命令，在每天上午的 11:45 删除缓存的文件以及 CMS 服务器的 directories.flushes 缓存。

Ansible 将使用用户声明的正确语法将 play 写入到 crontab 中。

通过检查 crontab 来验证它是否已附加。

cron 模块的一些常用参数有：

参数

参数	选项	注释
special_time	reboot、yearly、annually、monthly、	一系列重复出现时间。
state	absent、present	如果设为 present，它将创建命令。设为 absent 则会删除命令。
cron_file	Null	如果有大量的服务器需要维护，那么有时最好有一个预先编写好的 crontab 文件。
backup	yes、no	在编辑之前备份 crontab 文件。

使用 SYSTEMD 和 SERVICE 模块管理服务。S

为管理服务或重新加载守护进程，Ansible 提供了 **systemd** 和 service 模块。service 提供了一组基本选项，即 start、stop、restart 和 enable。systemd 模块提供更多配置选项。Systemd 允许您执行守护进程重新加载，service 模块则不会。

service 模块示例：

```
- name: start nginx
  service:
    name: nginx
    state: started"
```



注意

init 守护进程正在被 systemd 取代。因此在很多情况下，systemd 将是更好的选择。

systemd 模块示例：

```
- name: reload web server
  systemd:
    name: apache2
    state: reload
    daemon-reload: yes
```

REBOOT 模块

另一个很好用的 Ansible 系统模块是 **reboot**。这被认为比使用 shell 模块来发起关机更加安全。在运行 play 期间，reboot 模块将关闭受管主机，再等待它恢复运行，然后再继续执行 play。

reboot 模块示例：

```
- name: "Reboot after patching"
  reboot:
    reboot_timeout: 180

- name: force a quick reboot
  reboot:
```

SHELL 和 COMMAND 模块

与 service 和 systemd 模块一样，shell 和 command 也可以交换一些任务。command 模块被认为更安全，但某些环境变量不可用。此外，流操作符将无法运作。如果您需要流式传输命令，那么 shell 模块就可以。

shell 模块示例：

```
- name: Run a templated variable (always use quote filter to avoid injection)
  shell: cat {{ myfile|quote }}❶
```

❶ 要清理变量，建议您使用 “{{ var | quote }}” 而非 “{{ var }}”

command 模块示例：

```
- name: This command only
  command: /usr/bin/scrape_logs.py arg1 arg2
  args: ❶
  chdir: scripts/
  creates: /path/to/script
```

❶ 您可以将参数传递到表单中以提供选项。



注意

command 命令被视为更安全，因为它不受用户环境的影响。

通过在受管主机上收集事实，您可以访问环境变量。有一个名为 ansible_env 的子列表，其中包含所有的环境变量。

```
---  
- name:  
  hosts: webservers  
  vars:  
    local_shell: "{{ ansible_env }}"  
  tasks:  
    - name: Printing all the environment variables in Ansible  
      debug:  
        msg: "{{ local_shell }}"
```

- ① 您可以使用 lookup 插件确定您要返回的变量。msg:
"{{ lookup('env', 'USER', 'HOME', 'SHELL') }}"



参考文献

at - 通过 at 命令调度命令或脚本文件的执行 — Ansible 文档

https://docs.ansible.com/ansible/latest/modules/at_module.html

cron - 管理 cron.d 和 crontab 条目 — Ansible 文档

https://docs.ansible.com/ansible/latest/modules/cron_module.html

reboot - 重新引导计算机 — Ansible 文档

https://docs.ansible.com/ansible/latest/modules/reboot_module.html

service - 在计算机上运行服务 — Ansible 文档

https://docs.ansible.com/ansible/latest/modules/service_module.html

► 指导练习

管理引导过程和调度的进程

在本练习中，您将管理启动过程，调度周期性作业，并且重新引导受管主机。

成果

您应能够使用 playbook 来实现以下目标：

- 调度一个 cron 作业。
- 从 crontab 文件删除一个特定的 cron 作业。
- 调度 at 任务。
- 在受管主机上设置默认引导目标。
- 重新引导受管主机。

在你开始之前

从 workstation 运行 **lab system-process start** 脚本，以配置本练习的环境。此脚本将创建 **system-process** 工作目录，并下载练习所需的 Ansible 配置文件和主机清单文件。

```
[student@workstation ~]$ lab system-process start
```

- 1. 以 student 用户身份，在 workstation 上更改到 /home/student/system-process 工作目录。

```
[student@workstation ~]$ cd ~/system-process
[student@workstation system-process]$
```

- 2. 在当前的工作目录中，创建 playbook **create_crontab_file.yml**。配置 playbook，以使用 cron 模块创建 **/etc/cron.d/add-date-time** crontab 文件，以调度周期性 cron 作业。该作业应当以 devops 用户身份运行，在 Monday 到 Friday 的 09:00 至 16:59 之间每隔两分钟运行一次。该作业应当将当前的日期与时间附加到文件 /home/devops/my_datetime_cron_job

- 2.1. 创建新 playbook **create_crontab_file.yml**，再添加开始 play 所需的行。它应当以 webservers 组中的受管主机为目标，并启用特权升级。

```
---
- name: Recurring cron job
  hosts: webservers
  become: true
```

- 2.2. 定义一项任务，以使用 cron 模块调度周期性 cron 作业。

**注意**

cron 模块提供了一个 **name** 选项，可以唯一地描述 crontab 文件条目并确保结果与预期相符。该描述将添加到 crontab 文件中。例如，如果您要使用 **state=absent** 删除 crontab 条目，则需要 **name** 选项。此外，如果设置了默认状态 **name, state=present** 选项会防止始终创建新的 crontab 条目。

```
tasks:
  - name: Crontab file exists
    cron:
      name: Add date and time to a file
```

2.3. 将作业配置为在 **Monday** 到 **Friday** 的 **09:00** 至 **16:59** 之间每隔两分钟运行一次。

```
minute: */2"
hour: 9-16
weekday: 1-5
```

2.4. 使用 **cron_file** 参数来使用 **/etc/cron.d/add-date-time** crontab 文件，而不使用 **/var/spool/cron/** 中个别用户的 crontab。相对路径会将文件放到 **/etc/cron.d** 目录中。如果使用了 **cron_file** 参数，您还必须指定 **user** 参数。

```
user: devops
job: date >> /home/devops/my_date_time_cron_job
cron_file: add-date-time
state: present
```

2.5. 完成时，该 playbook 应当如下所示。检查 playbook 的准确性。

```
---
  - name: Recurring cron job
    hosts: webservers
    become: true

    tasks:
      - name: Crontab file exists
        cron:
          name: Add date and time to a file
          minute: */2"
          hour: 9-16
          weekday: 1-5
          user: devops
          job: date >> /home/devops/my_date_time_cron_job
          cron_file: add-date-time
          state: present
```

2.6. 通过运行 **ansible-playbook --syntax-check create_crontab_file.yml** 命令来验证 playbook 语法。更正所有错误，再继续下一步。

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> create_crontab_file.yml

playbook: create_crontab_file.yml
```

2.7. 运行 playbook。

```
[student@workstation system-process]$ ansible-playbook create_crontab_file.yml

PLAY [Recurring cron job] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Crontab file exists] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

2.8. 运行临时命令来验证 **/etc/cron.d/add-date-time** cron 文件存在并且其内容正确。

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "cat /etc/cron.d/add-date-time"
servera.lab.example.com | CHANGED | rc=0 >>
#Ansible: Add date and time to a file
*/2 9-16 * * 1-5 devops date >> /home/devops/my_date_time_cron_job
```

- 3. 在当前的工作目录中，创建 playbook **remove_cron_job.yml**。配置 playbook，以使用 cron 模块从 **/etc/cron.d/add-date-time** crontab 文件中删除 Add date and time to a file cron 作业

3.1. 创建新 playbook **remove_cron_job.yml**，并添加以下几行：

```
---
- name: Remove scheduled cron job
  hosts: webservers
  become: true

  tasks:
    - name: Cron job removed
      cron:
        name: Add date and time to a file
        user: devops
        cron_file: add-date-time
        state: absent
```

3.2. 通过运行 **ansible-playbook --syntax-check remove_cron_job.yml** 命令来验证 playbook 语法。更正所有错误，再继续下一步。

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> remove_cron_job.yml

playbook: remove_cron_job.yml
```

3.3. 运行 playbook。

```
[student@workstation system-process]$ ansible-playbook remove_cron_job.yml

PLAY [Remove scheduled cron job] ****

TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Cron job removed] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

3.4. 运行临时命令来验证 `/etc/cron.d/add-date-time` cron 文件仍然存在，但 cron 作业已被删除。

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "cat /etc/cron.d/add-date-time"
servera.lab.example.com | CHANGED | rc=0 >>
```

- 4. 在当前的工作目录中，创建 playbook `schedule_at_task.yml`。配置 playbook，以使用 `at` 模块来调度一分钟后运行的任务。该任务应当运行 `date` 命令，并将其输出重定向到 `/home/devops/my_at_date_time` 文件。使用 `unique: yes` 选项，以确保 `at` 队列中已存在该命令时不添加新的任务。

4.1. 创建新 playbook `schedule_at_task.yml`，并添加以下几行：

```
---
- name: Schedule at task
hosts: webservers
become: true
become_user: devops

tasks:
- name: Create date and time file
  at:
    command: "date > ~/my_at_date_time\n"
    count: 1
    units: minutes
    unique: yes
    state: present
```

4.2. 通过运行 `ansible-playbook -syntax-check schedule_at_task.yml` 命令来验证 playbook 语法。更正所有错误，再继续下一步。

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> schedule_at_task.yml

playbook: schedule_at_task.yml
```

4.3. 运行 playbook。

```
[student@workstation system-process]$ ansible-playbook schedule_at_task.yml

PLAY [Schedule at task] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Create date and time file] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

4.4. 等待一分钟，以便 `at` 命令完成，然后，运行临时命令来验证 `/home/devops/my_at_date_time` 文件存在并且具有正确的内容。

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "ls -l my_at_date_time"
servera.lab.example.com | CHANGED | rc=0 >>
-rw-rw-r--. 1 devops devops 30 abr 17 06:15 my_at_date_time

[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "cat my_at_date_time"
servera.lab.example.com | CHANGED | rc=0 >>
mié abr 17 06:15:00 EDT 2019
```

- 5. 在当前的工作目录中，创建 playbook `set_default_boot_target_graphical.yml`。配置 playbook，以使用 `file` 模块更改受管主机上的符号链接，使其引用 `graphical-target` 引导目标。



注意

在以下 `file` 模块中，`src` 参数值是符号链接引用的对象。`dest` 参数值是符号链接。

5.1. 创建新 playbook `set_default_boot_target_graphical.yml`，并添加以下几行：

```
---
- name: Change default boot target
  hosts: webservers
  become: true

  tasks:
```

```
- name: Default boot target is graphical
  file:
    src: /usr/lib/systemd/system/graphical.target
    dest: /etc/systemd/system/default.target
    state: link
```

5.2. 通过运行 `ansible-playbook --syntax-check`

`set_default_boot_target_graphical.yml` 命令来验证 playbook 语法。更正所有错误，再继续下一步。

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> set_default_boot_target_graphical.yml

playbook: set_default_boot_target_graphical.yml
```

5.3. 在运行 playbook 前，运行临时命令来验证当前的默认引导目标是 `multi-user.target`:

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
multi-user.target
```

5.4. 运行 playbook。

```
[student@workstation system-process]$ ansible-playbook \
> set_default_boot_target_graphical.yml

PLAY [Change default boot target] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Default boot target is graphical] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

5.5. 通过 add hoc 的方式来确认默认的引导目标是 `graphical.target`。

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
graphical.target
```

- 6. 在当前的工作目录中，创建用于重新引导受管主机的 playbook `reboot_hosts.yml`。更改默认目标后无需重新引导服务器。但是，知道如何创建重新引导受管主机的 playbook 可能会很有用。

6.1. 创建新 playbook `reboot_hosts.yml`，并添加以下几行：

```
---  
- name: Reboot hosts  
  hosts: webservers  
  become: true  
  
  tasks:  
    - name: Hosts are rebooted  
      reboot:
```

- 6.2. 通过运行 **ansible-playbook --syntax-check reboot_hosts.yml** 命令来验证 playbook 语法。更正所有错误，再继续下一步。

```
[student@workstation system-process]$ ansible-playbook --syntax-check \  
> reboot_hosts.yml  
  
playbook: reboot_hosts.yml
```

- 6.3. 在运行 playbook 前，运行临时命令来确定最近一次系统重新引导的时间戳。

```
[student@workstation system-process]$ ansible webservers -u devops -b \  
> -a "who -b"  
servera.lab.example.com | CHANGED | rc=0 >>  
    system boot 2019-04-12 06:01
```

- 6.4. 运行 playbook。

```
[student@workstation system-process]$ ansible-playbook reboot_hosts.yml  
  
PLAY [Reboot hosts] *****  
  
TASK [Gathering Facts] *****  
ok: [servera.lab.example.com]  
  
TASK [Hosts are rebooted] *****  
changed: [servera.lab.example.com]  
  
PLAY RECAP *****  
servera.lab.example.com : ok=2     changed=1     unreachable=0     failed=0
```

- 6.5. 运行临时命令来确定最近一次系统重新引导的时间戳。运行 playbook 后显示的时间戳应该更晚。

```
[student@workstation system-process]$ ansible webservers -u devops -b \  
> -a "who -b"  
servera.lab.example.com | CHANGED | rc=0 >>  
    system boot 2019-04-12 06:20
```

- 6.6. 运行第二个临时命令，以确定 **graphical.target** 引导目标在重新引导后仍然有效。

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
graphical.target
```

- 7. 为保持其余练习过程中的一致性, 请将默认引导目标重新更改为原先的设置 `multi-user.target`。在当前的工作目录中, 创建 playbook `set_default_boot_target_multi-user.yml`。配置 playbook, 以使用 `file` 模块更改受管主机上的符号链接, 使其引用 `multi-user.target` 引导目标。

7.1. 创建新 playbook `set_default_boot_target_multi-user.yml`, 并添加以下几行:

```
---
- name: Change default runlevel target
  hosts: webservers
  become: true

  tasks:
    - name: Default runlevel is multi-user target
      file:
        src: /usr/lib/systemd/system/multi-user.target
        dest: /etc/systemd/system/default.target
        state: link
```

7.2. 通过运行 `ansible-playbook --syntax-check set_default_boot_target_multi-user.yml` 命令来验证 playbook 语法。更正所有错误, 再继续下一步。

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> set_default_boot_target_multi-user.yml

playbook: set_default_boot_target_multi-user.yml
```

7.3. 运行 playbook。

```
[student@workstation system-process]$ ansible-playbook \
> set_default_boot_target_multi-user.yml

PLAY [Change default runlevel target] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Default runlevel is multi-user target] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

7.4. 运行临时命令来验证默认引导目标现在是 `multi-user.target`。

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
multi-user.target
```

完成

在 workstation 上，运行 **lab system-process finish** 脚本来清理本练习。

```
[student@workstation ~]$ lab system-process finish
```

本引导式练习到此结束。

管理存储

培训目标

学完本节后，您应能够对存储设备进行分区，配置 LVM，格式化分区或逻辑卷，挂载文件系统，以及添加交换文件或空间。

使用 ANSIBLE 模块配置存储

红帽 Ansible 引擎提供了一组模块，可用于在受管主机上配置存储设备。这些模块支持对设备进行分区，创建逻辑卷，以及创建和挂载文件系统。

parted 模块

parted 模块支持块设备的分区。该模块包含有 **parted** 命令的功能，允许创建具有特定大小、标志和对齐的分区。下表列出了 **parted** 模块的一些参数。

参数名称	描述
align	配置分区对齐。
device	块设备。
flags	分区的标志。
number	分区编号。
part_end	以 parted 支持单位指定的从磁盘开头开始的分区大小。
state	创建或删除分区。
unit	分区信息的大小单位。

以下示例创建了一个 10 GB 的新分区。

```
- name: New 10GB partition
  parted:
    device: /dev/vdb ①
    number: 1 ②
    state: present ③
    part_end: 10GB ④
```

- ① 使用 **vdb** 作为要分区的块设备。
- ② 创建分区号 1。
- ③ 确保该分区可用。
- ④ 将分区大小设置为 10 GB。

lvg 和 lvol 模块

lvg 和 **lvol** 模块支持创建逻辑卷，包括物理卷和卷组的配置。**lvg** 将块设备取为参数，将其配置为卷组的后端物理卷。下表列出了 **lvg** 模块的一些参数。

参数名称	描述
pesize	物理区块的大小。必须是 2 的幂，或 128 KiB 的倍数。
pvs	逗号分隔的设备列表，这些设备将配置为卷组的物理卷。
vg	卷组的名称。
state	创建或删除卷。

以下任务使用块设备作为后端，创建具有特定物理区块大小的卷组。

```
- name: Creates a volume group
  lvg:
    vg: vg1 ①
    pvs: /dev/vda1 ②
    pesize: 32 ③
```

- ① 卷组名称是 **vg1**。
- ② 使用 **/dev/vda1** 作为卷组的后端物理卷。
- ③ 将物理区块大小设置为 **32**。

在以下示例中，如果 **vg1** 卷组已可用且将 **/dev/vdb1** 用作物理卷，则通过添加使用 **/dev/vdc1** 的一个新物理卷来扩大该卷。

```
- name: Resize a volume group
  lvg:
    vg: vg1
    pvs: /dev/vdb1,/dev/vdc1
```

lvol 模块可创建逻辑卷，并且支持放大和收缩这些卷的大小，以及其上的文件系统。此模块还支持为逻辑卷创建快照。下表列出了 **lvol** 模块的一些参数。

参数名称	描述
lv	逻辑卷的名称。
resizesfs	调整逻辑卷的文件系统大小。
shrink	启用逻辑卷收缩。
size	逻辑卷的大小。
snapshot	逻辑卷快照的名称。
state	创建或删除逻辑卷。
vg	逻辑卷的父卷组。

以下任务创建一个 2 GB 的逻辑卷。

```
- name: Create a logical volume of 2GB
  lvol:
    vg: vg1 ①
    lv: lv1 ②
    size: 2g ③
```

- ① 父卷组名称是 `vg1`。
- ② 逻辑卷名称是 `lv1`。
- ③ 逻辑卷的大小是 2 GB。

filesystem 模块

`filesystem` 模块支持创建文件系统和调整文件系统大小。此模块支持调整 `ext2`、`ext3`、`ext4`、`ext4dev`、`f2fs`、`lvm`、`xfs` 和 `vfat` 的文件系统大小。下表列出了 `filesystem` 模块的一些参数。

参数名称	描述
<code>dev</code>	块设备名称。
<code>fstype</code>	文件系统类型。
<code>resizesfs</code>	将文件系统大小增加到块设备大小。

以下示例在分区中创建一个文件系统。

```
- name: Create an XFS filesystem
  filesystem:
    fstype: xfs ①
    dev: /dev/vdb1 ②
```

- ① 使用 `XFS` 文件系统。
- ② 使用 `/dev/vdb1` 设备。

mount 模块

`mount` 模块支持在 `/etc/fstab` 上配置挂载点。下表列出了 `mount` 模块的一些参数。

参数名称	描述
<code>fstype</code>	文件系统类型。
<code>opts</code>	挂载选项。
<code>path</code>	挂载点路径。
<code>src</code>	要挂载的设备。
<code>state</code>	指定挂载状态。如果设为 <code>mounted</code> ，系统将挂载设备并使用该挂载信息配置 <code>/etc/fstab</code> 。要取消挂载设备并将它从 <code>/etc/fstab</code> 移除，可使用 <code>absent</code> 。

以下示例使用特定 ID 挂载设备。

```
- name: Mount device with ID
  mount:
    path: /data ①
    src: UUID=a8063676-44dd-409a-b584-68be2c9f5570 ②
    fstype: xfs ③
    state: present ④
```

- ① 使用 `/data` 作为挂载点路径。
- ② 使用 `a8063676-44dd-409a-b584-68be2c9f5570` ID 挂载设备。
- ③ 使用 XFS 文件系统。
- ④ 挂载设备并相应地配置 `/etc/fstab`。

以下示例将 `172.25.250.100:/share` 上可用的 NFS 共享挂载到受管主机上的 `/nfsshare` 目录。

```
- name: Mount NFS share
  mount: name=/nfsshare src=172.25.250.100:/share fstype=nfs
    opts=defaults,nobootwait dump=0 passno=2 state=mounted
```

使用模块配置交换

红帽 Ansible 引擎目前不包含管理交换内存的模块。若要通过 Ansible 使用逻辑卷向系统添加交换内存，您需要使用 `lvcreate` 和 `lvconvert` 模块创建新的卷组和逻辑卷。准备就绪后，您需要使用 `command` 模块及 `mkswap` 命令将新逻辑卷格式化为交换。最后，您需要使用 `command` 模块及 `swapon` 命令激活新的交换设备。红帽 Ansible 引擎包含了 `ansible_swaptotal_mb` 变量，其含有总交换内存大小。当交换内存不足时，您可以使用此变量来触发交换配置和启用。以下任务为交换内存创建卷组和逻辑卷，将该逻辑卷格式化为交换，并将它激活。

```
- name: Create new swap VG
  lvcreate:
    vg: vgvgswap
    pvs: /dev/vda1
    state: present

- name: Create new swap LV
  lvcreate:
    vg: vgvgswap
    lv: lvswap
    size: 10g

- name: Format swap LV
  command: mkswap /dev/vgvgswap/lvswap
  when: ansible_swaptotal_mb < 128

- name: Activate swap LV
  command: swapon /dev/vgvgswap/lvswap
  when: ansible_swaptotal_mb < 128
```

存储配置的 ANSIBLE 事实

Ansible 使用事实向控制节点检索有关受管主机配置的信息。您可以使用 `setup` Ansible 模块来检索受管主机的所有 Ansible 事实。

```
[user@controlnode ~]$ ansible webservers -m setup
host.lab.example.com | SUCCESS => {
  "ansible_facts": {
    ...output omitted...
  }
}
```

setup 模块的 **filter** 选项支持根据 shell 样式的通配符进行精细过滤。

ansible_devices 元素包括受管主机上可用的所有存储设备。每个存储设备的元素包括分区或总大小等其他信息。以下示例显示受管主机的 **ansible_devices** 元素，该主机具有三个存储设备：**sr0**、**vda** 和 **vdb**。

```
[user@controlnode ~]$ ansible webservers -m setup -a 'filter=ansible_devices'
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_devices": {
            "sr0": {
                "holders": [],
                "host": "IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]",
                "links": {
                    "ids": [
                        "ata-QEMU_DVD-ROM_QM00003"
                    ],
                    "labels": [],
                    "masters": [],
                    "uuids": []
                },
                "model": "QEMU DVD-ROM",
                "partitions": {},
                "removable": "1",
                "rotational": "1",
                "sas_address": null,
                "sas_device_handle": null,
                "scheduler_mode": "mq-deadline",
                "sectors": "2097151",
                "sectorsize": "512",
                "size": "1024.00 MB",
                "support_discard": "0",
                "vendor": "QEMU",
                "virtual": 1
            },
            "vda": {
                "holders": [],
                "host": "SCSI storage controller: Red Hat, Inc. Virtio block device",
                "links": {
                    "ids": [],
                    "labels": [],
                    "masters": [],
                    "uuids": []
                },
                "model": null,
                "partitions": {
                    "vda1": {
                        "holders": [],
                        "links": {
                            "ids": [],
                            "labels": [],
                            "masters": []
                        }
                    }
                }
            }
        }
    }
}
```

```

        "uuids": [
            "a8063676-44dd-409a-b584-68be2c9f5570"
        ],
        },
        "sectors": "20969439",
        "sectorsize": 512,
        "size": "10.00 GB",
        "start": "2048",
        "uuid": "a8063676-44dd-409a-b584-68be2c9f5570"
    }
},
"removable": "0",
"rotational": "1",
"sas_address": null,
"sas_device_handle": null,
"scheduler_mode": "mq-deadline",
"sectors": "20971520",
"sectorsize": "512",
"size": "10.00 GB",
"support_discard": "0",
"vendor": "0x1af4",
"virtual": 1
},
"vdb": {
    "holders": [],
    "host": "SCSI storage controller: Red Hat, Inc. Virtio block
device",
    "links": {
        "ids": [],
        "labels": [],
        "masters": [],
        "uuids": []
    },
    "model": null,
    "partitions": {},
    "removable": "0",
    "rotational": "1",
    "sas_address": null,
    "sas_device_handle": null,
    "scheduler_mode": "mq-deadline",
    "sectors": "10485760",
    "sectorsize": "512",
    "size": "5.00 GB",
    "support_discard": "0",
    "vendor": "0x1af4",
    "virtual": 1
}
},
"changed": false
}

```

`ansible_device_links` 元素包括各个存储设备的所有可用链接。以下示例显示单个主机的 `ansible_device_links` 元素，该主机具有 `sr0` 和 `vda1` 两个存储设备，其带有关联的心。

```
[user@controlnode ~]$ ansible webservers -m setup -a 'filter=ansible_device_links'
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_device_links": {
            "ids": {
                "sr0": [
                    "ata-QEMU_DVD-ROM_QM00003"
                ]
            },
            "labels": {},
            "masters": {},
            "uuids": {
                "vda1": [
                    "a8063676-44dd-409a-b584-68be2c9f5570"
                ]
            }
        }
    },
    "changed": false
}
```

`ansible_mounts` 元素包括受管主机上当前挂载的设备的信息，如挂载的设备、挂载点和选项。以下输出显示受管主机的 `ansible_mounts` 元素，该主机具有一个活跃挂载，即 / 目录上的 /dev/vda1。

```
[user@controlnode ~]$ ansible webservers -m setup -a 'filter=ansible_mounts'
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_mounts": [
            {
                "block_available": 2225732,
                "block_size": 4096,
                "block_total": 2618619,
                "block_used": 392887,
                "device": "/dev/vda1",
                "fstype": "xfs",
                "inode_available": 5196602,
                "inode_total": 5242304,
                "inode_used": 45702,
                "mount": "/",
                "options": "rw,seclabel,relatime,attr2,inode64,noquota",
                "size_available": 9116598272,
                "size_total": 10725863424,
                "uuid": "a8063676-44dd-409a-b584-68be2c9f5570"
            }
        ],
        "changed": false
    }
}
```



参考文献

parted - 配置块设备分区 — Ansible 文档

https://docs.ansible.com/ansible/latest/modules/parted_module.html

lvg - 配置 LVM 卷组 — Ansible 文档

https://docs.ansible.com/ansible/latest/modules/lvg_module.html

lvol - 配置 LVM 逻辑卷 — Ansible 文档

https://docs.ansible.com/ansible/latest/modules/lvol_module.html

filesystem - 制作文件系统 — Ansible 文档

https://docs.ansible.com/ansible/latest/modules/filesystem_module.html

mount - 控制活跃和配置的挂载点 — Ansible 文档

https://docs.ansible.com/ansible/latest/modules/mount_module.html

► 指导练习

管理存储

在本练习中，您将对新磁盘进行分区，创建逻辑卷并用 XFS 文件系统进行格式化，然后在引导时立即在受管主机上自动挂载它们。

成果

您应能够：

- 使用 `parted` 模块来配置块设备分区。
- 使用 `lvg` 模块来管理 LVM 卷组。
- 使用 `lvol` 模块来管理 LVM 逻辑卷。
- 使用 `filesystem` 模块来创建文件系统。
- 使用 `mount` 模块在 `/etc/fstab` 中控制和配置挂载点。

在你开始之前

从 `workstation` 运行 `lab system-storage start` 脚本，以配置本练习的环境。该脚本将创建 `system-storage` 项目目录，并下载练习所需的 Ansible 配置文件和主机清单文件。

```
[student@workstation ~]$ lab system-storage start
```

场景概述

您负责管理一组 Web 服务器。Web 服务器配置的建议做法是将 Web 服务器数据存储在单独的分区或逻辑卷中。

您将编写 playbook 来实现以下目标：

- 管理 `/dev/vdb` 设备的分区
- 管理用于存储 Web 服务器数据的卷组 `apache-vg`
- 创建两个逻辑卷，取名为 `content-lv` 和 `logs-lv`，它们都由 `apache-vg` 卷组支持
- 在两个逻辑卷上创建 XFS 文件系统
- 将 `content-lv` 逻辑卷挂载到 `/var/www`
- 将 `logs-lv` 逻辑卷挂载到 `/var/log/httpd`

如果 Web 服务器的要求出现变化，则更新相应的 playbook 变量并重新执行 playbook。该 playbook 应具有幂等性。

- 1. 以 `student` 用户身份，在 `workstation` 上更改到 `/home/student/system-storage` 工作目录。

```
[student@workstation ~]$ cd ~/system-storage
[student@workstation system-storage]$
```

- 2. 检查项目目录中的框架 playbook 文件 **storage.yml**, 以及关联的变量文件 **storage_vars.yml**。执行 playbook。

2.1. 检查 **storage.yml** playbook。

```
---
- name: Ensure Apache Storage Configuration
  hosts: webservers
  vars_files:
    - storage_vars.yml
  tasks:
    - name: Correct partitions exist on /dev/vdb
      debug:
        msg: TODO
      loop: "{{ partitions }}"
    - name: Ensure Volume Groups Exist
      debug:
        msg: TODO
      loop: "{{ volume_groups }}"
    - name: Create each Logical Volume (LV) if needed
      debug:
        msg: TODO
      loop: "{{ logical_volumes }}"
      when: true
    - name: Ensure XFS Filesystem exists on each LV
      debug:
        msg: TODO
      loop: "{{ logical_volumes }}"
    - name: Ensure the correct capacity for each LV
      debug:
        msg: TODO
      loop: "{{ logical_volumes }}"
    - name: Each Logical Volume is mounted
      debug:
        msg: TODO
      loop: "{{ logical_volumes }}"
```

每个任务的名称充当计划要实施的预期步骤的概要。在后续步骤中，您将更新并更改这六个任务。

2.2. 检查 **storage_vars.yml** 变量文件。