

```

---
partitions:
- number: 1
  start: 1MiB
  end: 257MiB

volume_groups:
- name: apache-vg
  devices: /dev/vdb1

logical_volumes:
- name: content-lv
  size: 64M
  vgroup: apache-vg
  mount_path: /var/www

- name: logs-lv
  size: 128M
  vgroup: apache-vg
  mount_path: /var/log/httpd

```

此文件描述各 Web 服务器上的预期分区结构、卷组和逻辑卷。第一个分区始于 `/dev/vdb` 设备开头偏移 1 MiB 处，终于偏移 257 MiB 处，总大小为 256 MiB。

每台 Web 服务器上具有一个卷组 `apache-vg`，其包含 `/dev/vdb` 设备的第一分区。

每台 Web 服务器具有两个逻辑卷。第一逻辑卷名为 `content-lv`，大小为 64 MiB，连至 `apache-vg` 卷组，并挂载于 `/var/www`。第二逻辑卷名为 `logs-lv`，大小为 128 MiB，连至 `apache-vg` 卷组，并挂载于 `/var/log/httpd`。



### 注意

`apache-vg` 卷组的容量为 256 MiB，因为它由 `/dev/vdb1` 分区提供支持。它为两个逻辑卷提供充足的容量。

### 2.3. 执行 `storage.yml` playbook。

```

[student@workstation system-storage]$ ansible-playbook storage.yml

PLAY [Ensure Apache Storage Configuration] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] ****
ok: [servera.lab.example.com] => (item={'start': u'1MiB', u'end': u'257MiB',
u'number': 1}) => {
    "msg": "TODO"
}

...output omitted...

```

```

TASK [Each Logical Volume is mounted] ****
ok: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size': u'64M',
u'mount_path': u'/var/www', u'name': u'content-lv'}) => {
    "msg": "TODO"
}
ok: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size': u'128M',
u'mount_path': u'/var/log/httpd', u'name': u'logs-lv'}) => {
    "msg": "TODO"
}

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

```

- 3. 更改第一个任务，以使用 `parted` 模块来配置各个循环项的分区。各个项目描述每台 Web 服务器上 `/dev/vdb` 设备的预期分区：

**number**

分区编号。将此用作 `parted` 模块 `number` 关键字的值。

**start**

分区的起点，表示为块设备开头的偏移量。将此用作 `parted` 模块 `part_start` 关键字的值。

**end**

分区的终点，表示为块设备开头的偏移量。将此用作 `parted` 模块 `part_end` 关键字的值。

第一个任务的内容应该是：

```

- name: Correct partitions exist on /dev/vdb
  parted:
    device: /dev/vdb
    state: present
    number: "{{ item.number }}"
    part_start: "{{ item.start }}"
    part_end: "{{ item.end }}"
    loop: "{{ partitions }}"

```

- 4. 更改 play 的第二个任务，以使用 `lvg` 模块为各个循环项配置卷组。`volume_groups` 变量的每个项目描述各 Web 服务器上应存在的卷组：

**name**

卷组的名称。将此用作 `lvg` 模块 `vg` 关键字的值。

**devices**

组成卷组的设备或分区的逗号分隔列表。将此用作 `lvg` 模块 `pvs` 关键字的值。

第二个任务的内容应该是：

```
- name: Ensure Volume Groups Exist
  lvg:
    vg: "{{ item.name }}"
    pvs: "{{ item.devices }}"
    loop: "{{ volume_groups }}"
```

- 5. 更改 play 的第三个任务，以使用 `lvol` 模块为各个项目创建逻辑卷。使用项目的关键字来创建新逻辑卷：

**name**

逻辑卷的名称。将此用作 `lvol` 模块 `lv` 关键字的值。

**vgroup**

为逻辑卷提供存储的卷组的名称。

**size**

逻辑卷的大小。此关键字的值是 `lvcreate` 命令的 `-L` 选项的任何可接受值。

只有在逻辑卷尚不存在时才执行此任务。更新 `when` 语句，以检查是否不存在名称与项目的 `name` 关键字值匹配的逻辑卷。

5.1. 更改第三个任务，以使用 `lvol` 模块。利用每个项目的关键字，设置卷组名称、逻辑卷名称和逻辑卷大小。第三个任务的内容现在是：

```
- name: Create each Logical Volume (LV) if needed
  lvol:
    vg: "{{ item.vgroup }}"
    lv: "{{ item.name }}"
    size: "{{ item.size }}"
    loop: "{{ logical_volumes }}"
    when: true
```

5.2. Ansible 事实 `ansible_lvm` 包含各个主机上逻辑卷管理对象的信息。使用一个临时命令来查看远程主机上当前的逻辑卷集合：

```
[student@workstation system-storage]$ ansible all -m setup -a \
> "filter=ansible_lvm"
servera.lab.example.com | SUCCESS => {
  "ansible_facts": {
    "ansible_lvm": {
      "lvs": {},
      "pvs": {},
      "vgs": {}
    }
  }
}
```

```

    },
    "discovered_interpreter_python": "/usr/libexec/platform-python"
},
"changed": false
}

```

`lvs` 关键字的值指出远程主机上没有逻辑卷。

5.3. 执行 playbook 以在远程主机上创建逻辑卷。

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

PLAY [Ensure Apache Storage Configuration] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] ****
changed: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure Volume Groups Exist] ****
changed: [servera.lab.example.com] => (item={...output omitted...})

TASK [Create each Logical Volume (LV) if needed] ****
changed: [servera.lab.example.com] => (item={...output omitted...})
changed: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure XFS Filesystem exists on each LV] ****
ok: [servera.lab.example.com] => (item={...output omitted...}) => {
    "msg": "TODO"
}
...output omitted...
PLAY RECAP ****
servera.lab.example.com : ok=7      changed=3      unreachable=0      failed=0
```

5.4. 执行另一个临时命令，以查看远程主机上存在逻辑卷时 `ansible_lvm` 变量的结构。

```
[student@workstation system-storage]$ ansible all -m setup -a \
> "filter=ansible_lvm"
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_lvm": {
            "lvs": ❶{
                "content-lv": {
                    "size_g": "0.06",
                    "vg": "apache-vg"
                },
                "logs-lv": {
                    "size_g": "0.12",
                    "vg": "apache-vg"
                }
            },
            "pvs": ❷{
                "/dev/vdb1": {

```

```

        "free_g": "0.06",
        "size_g": "0.25",
        "vg": "apache-vg"
    }
},
"vgs": ❸{
    "apache-vg": {
        "free_g": "0.06",
        "num_lvs": "2",
        "num_pvs": "1",
        "size_g": "0.25"
    }
}
},
"changed": false
}

```

- ❶ `lvs` 关键字的值是一种键值对数据结构。此结构的键是主机上任何逻辑卷的名称。这表明 `content-lv` 和 `logs-lv` 逻辑卷都存在。对于每个逻辑卷，对应的卷组由 `vg` 关键字提供。
- ❷ `pvs` 关键字包含有关主机上物理卷的信息。此信息指出 `/dev/vdb1` 分区属于 `apache-vg` 卷组。
- ❸ `vgs` 关键字包含有关主机上卷组的信息。

5.5. 更新 `when` 语句，以检查是否存在名称与项目的 `name` 关键字值匹配的逻辑卷。第三个任务的内容现在是：

```

- name: Create each Logical Volume (LV) if needed
  lvol:
    vg: "{{ item.vgroup }}"
    lv: "{{ item.name }}"
    size: "{{ item.size }}"
  loop: "{{ logical_volumes }}"
  when: item.name not in ansible_lvm["lvs"]

```

- 6. 更改第四个任务以使用 `filesystem` 模块。配置该任务，以确保每个逻辑卷格式化为 XFS 文件系统。回想一下，逻辑卷与逻辑设备 `/dev/<volume group name>/<logical volume name>` 相关联。

第四个任务的内容应该是：

```

- name: Ensure XFS Filesystem exists on each LV
  filesystem:
    dev: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
  loop: "{{ logical_volumes }}"

```

- 7. 配置第五个任务以确保每个逻辑卷具有正确的存储容量。如果逻辑卷的容量增大，请务必强制扩展卷的文件系统。

**警告**

如果逻辑卷需要减小容量，则此任务将失败，因为 XFS 文件系统不支持收缩容量。

第五个任务的内容应该是：

```
- name: Ensure the correct capacity for each LV
  lvol:
    vg: "{{ item.vgroup }}"
    lv: "{{ item.name }}"
    size: "{{ item.size }}"
    resizefs: yes
    force: yes
  loop: "{{ logical_volumes }}"
```

- 8. 在第六个任务中，使用 `mount` 模块来确保每个逻辑卷挂载到对应的挂载路径，并在重新引导后保留。

第六个任务的内容应该是：

```
- name: Each Logical Volume is mounted
  mount:
    path: "{{ item.mount_path }}"
    src: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
    opts: noatime
    state: mounted
  loop: "{{ logical_volumes }}"
```

- 9. 检查完成的 `storage.yml` playbook。执行 playbook 并验证是否已挂载每个逻辑卷。

9.1. 检查 playbook：

```
---
- name: Ensure Apache Storage Configuration
  hosts: webservers
  vars_files:
    - storage_vars.yml
  tasks:
    - name: Correct partitions exist on /dev/vdb
      parted:
        device: /dev/vdb
        state: present
        number: "{{ item.number }}"
        part_start: "{{ item.start }}"
        part_end: "{{ item.end }}"
      loop: "{{ partitions }}"
    - name: Ensure Volume Groups Exist
      lvg:
        vg: "{{ item.name }}"
        pvs: "{{ item.devices }}"
```

```

loop: "{{ volume_groups }}"

- name: Create each Logical Volume (LV) if needed
  lvol:
    vg: "{{ item.vgroup }}"
    lv: "{{ item.name }}"
    size: "{{ item.size }}"
  loop: "{{ logical_volumes }}"
  when: item.name not in ansible_lvm["lvs"]

- name: Ensure XFS Filesystem exists on each LV
  filesystem:
    dev: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
  loop: "{{ logical_volumes }}"

- name: Ensure the correct capacity for each LV
  lvol:
    vg: "{{ item.vgroup }}"
    lv: "{{ item.name }}"
    size: "{{ item.size }}"
    resizefs: yes
    force: yes
  loop: "{{ logical_volumes }}"

- name: Each Logical Volume is mounted
  mount:
    path: "{{ item.mount_path }}"
    src: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
    opts: noatime
    state: mounted
  loop: "{{ logical_volumes }}"

```

## 9.2. 执行 playbook。

```

[student@workstation system-storage]$ ansible-playbook storage.yml

PLAY [Ensure Apache Storage Configuration] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] ****
ok: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure Volume Groups Exist] ****
ok: [servera.lab.example.com] => (item={...output omitted...})
...output omitted...

TASK [Create each Logical Volume (LV) if needed] ****
skipping: [servera.lab.example.com] => (item={...output omitted...})
skipping: [servera.lab.example.com] => (item={...output omitted...})

```

```

TASK [Ensure XFS Filesystem exists on each LV] ****
changed: [servera.lab.example.com] => (item={...output omitted...})
changed: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure the correct capacity for each LV] ****
ok: [servera.lab.example.com] => (item={...output omitted...})
ok: [servera.lab.example.com] => (item={...output omitted...})

TASK [Each Logical Volume is mounted] ****
changed: [servera.lab.example.com] => (item={...output omitted...})
changed: [servera.lab.example.com] => (item={...output omitted...})

PLAY RECAP ****
servera.lab.example.com      : ok=6    changed=2    unreachable=0    failed=0
skipped=1      rescued=0    ignored=0

```

执行期间跳过了一个任务，因为之前使用相同的变量值执行过该 playbook。不需要创建逻辑卷。

### 9.3. 使用 Ansible 临时命令对远程主机运行 **lsblk** 命令。输出中指示逻辑卷的挂载点。

```
[student@workstation system-storage]$ ansible all -a lsblk
servera.lab.example.com | CHANGED | rc=0 >>
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0        11:0    1 1024M  0 rom
vda       252:0    0   10G  0 disk
└─vda1     252:1    0   10G  0 part /
vdb       252:16   0    1G  0 disk
└─vdb1     252:17   0  256M  0 part
├─apache--vg-content--lv 253:0    0   64M  0 lvm  /var/www
└─apache--vg-logs--lv    253:1    0  128M  0 lvm  /var/log/httpd
```

- 10. 将 **content-lv** 逻辑卷的容量增大到 128 MiB，**logs-lv** 逻辑卷的容量增大到 256 MiB。这需要增大 **apache-vg** 卷组的容量。

创建一个容量为 256 MiB 的新分区，并将它添加到 **apache-vg** 卷组。

10.1. 编辑 **storage\_vars.yml** 文件中的 **partitions** 变量定义，以向 **/dev/vdb** 设备添加第二分区。**partitions** 变量的内容应当是：

```
partitions:
- number: 1
  start: 1MiB
  end: 257MiB
- number: 2
  start: 257MiB
  end: 513MiB
```

10.2. 编辑 **storage\_vars.yml** 文件中 **volume\_groups** 变量定义。将第二分区添加到为卷组提供支持的设备列表中。**volume\_groups** 变量的内容应当是：

```

volume_groups:
  - name: apache-vg
    devices: /dev/vdb1,/dev/vdb2

```

10.3. 将 **storage\_vars.yml** 文件中定义的每个逻辑卷的容量翻倍。**logical\_volumes** 变量的内容应当是：

```

logical_volumes:
  - name: content-lv
    size: 128M
    vgroup: apache-vg
    mount_path: /var/www

  - name: logs-lv
    size: 256M
    vgroup: apache-vg
    mount_path: /var/log/httpd

```

10.4. 执行 playbook。验证每个逻辑卷的新容量。

```

[student@workstation system-storage]$ ansible-playbook storage.yml

PLAY [Ensure Apache Storage Configuration] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] ****
ok: [servera.lab.example.com] => (item={...output omitted...})
changed: [servera.lab.example.com] => (item={'start': u'257MiB', 'end': u'513MiB', 'number': 2})

TASK [Ensure Volume Groups Exist] ****
changed: [servera.lab.example.com] => (item={'name': u'apache-vg', 'devices': u'/dev/vdb1,/dev/vdb2'})
...output omitted...

TASK [Create each Logical Volume (LV) if needed] ****
skipping: [servera.lab.example.com] => (item={'vgroup': u'apache-vg', 'size': u'128M', 'mount_path': u'/var/www', 'name': u'content-lv'})
skipping: [servera.lab.example.com] => (item={'vgroup': u'apache-vg', 'size': u'256M', 'mount_path': u'/var/log/httpd', 'name': u'logs-lv'})

TASK [Ensure XFS Filesystem exists on each LV] ****
ok: [servera.lab.example.com] => (item={...output omitted...})
ok: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure the correct capacity for each LV] ****
changed: [servera.lab.example.com] => (item={'vgroup': u'apache-vg', 'size': u'128M', 'mount_path': u'/var/www', 'name': u'content-lv'})
changed: [servera.lab.example.com] => (item={'vgroup': u'apache-vg', 'size': u'256M', 'mount_path': u'/var/log/httpd', 'name': u'logs-lv'})

```

```
TASK [Each Logical Volume is mounted] ****
ok: [servera.lab.example.com] => (item={...output omitted...})
ok: [servera.lab.example.com] => (item={...output omitted...})

PLAY RECAP ****
servera.lab.example.com      : ok=6    changed=3    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

输出中指出远程主机上分区和卷组已更改，并且两个逻辑卷都已调整大小。

10.5 使用 Ansible 临时命令对远程主机运行 **lsblk** 命令。

```
[student@workstation system-storage]$ ansible all -a lsblk
servera.lab.example.com | CHANGED | rc=0 >>
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0           11:0    1 1024M  0 rom
vda           252:0    0   10G  0 disk
└─vda1        252:1    0   10G  0 part /
vdb           252:16   0   1G  0 disk
└─vdb1        252:17   0  256M 0 part
└─apache--vg-content--lv 253:0    0  128M 0 lvm  /var/www
└─apache--vg-logs--lv   253:1    0  256M 0 lvm  /var/log/httpd
└─vdb2        252:18   0  256M 0 part
└─apache--vg-content--lv 253:0    0  128M 0 lvm  /var/www
└─apache--vg-logs--lv   253:1    0  256M 0 lvm  /var/log/httpd
```

输出中指出每个逻辑卷大小正确并挂载到正确的目录。每个逻辑卷存在两个条目，因为存储在逻辑卷的文件实际上位于任一分区（/dev/vdb1 或 /dev/vdb2）。

## 完成

运行 **lab system-storage finish** 命令，以清理受管主机。

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

本引导式练习到此结束。

# 管理网络配置

---

## 培训目标

学完本节后，您应能够在受管主机上配置网络设置和名称解析，并收集网络相关的 Ansible 事实。

## 使用网络系统角色配置网络

红帽企业 Linux 8 中包含一系列 Ansible 系统角色，可用于配置基于 RHEL 的系统。rhel-system-roles 软件包可安装这些系统角色，这些角色可以支持时间同步或网络等配置。您可以使用 **ansible-galaxy list** 命令列出当前安装的系统角色。

```
[user@controlnode ~]$ ansible-galaxy list
- linux-system-roles.kdump, (unknown version)
- linux-system-roles.network, (unknown version)
- linux-system-roles.postfix, (unknown version)
- linux-system-roles.selinux, (unknown version)
- linux-system-roles.timesync, (unknown version)
- rhel-system-roles.kdump, (unknown version)
- rhel-system-roles.network, (unknown version)
- rhel-system-roles.postfix, (unknown version)
- rhel-system-roles.selinux, (unknown version)
- rhel-system-roles.timesync, (unknown version)
```

角色位于 **/usr/share/ansible/roles** 目录中。以 **linux-system-roles** 开头的角色是匹配的 **rhel-system-roles** 角色的符号链接。

网络系统角色支持在受管主机上配置网络。此角色支持以太网接口、网桥接口、绑定接口、VLAN 接口、MacVLAN 接口和 Infiniband 接口的配置。网络角色通过 **network\_provider** 和 **network\_connections** 这两个变量配置。

```
---
network_provider: nm
network_connections:
  - name: ens4
    type: ethernet
    ip:
      address:
        - 172.25.250.30/24
```

**network\_provider** 变量配置后端提供程序，即 **nm** (NetworkManager) 或 **initscripts**。在红帽企业 Linux 8 中，网络角色使用 **nm** (NetworkManager) 作为默认的网络提供程序。**initscripts** 提供程序用于 RHEL 6 系统，需要 **network** 服务处于可用状态。**network\_connections** 变量使用接口名称作为连接名称来配置指定为字典列表的不同连接。

下表列出了 **network\_connections** 变量的选项。

选项名称	描述
name	标识连接配置集。
state	连接配置集的运行时状态。如果连接配置集为活跃，状态为 <b>up</b> ，否则为 <b>down</b> 。
persistent_state	标识连接配置集是否持久。如果连接配置集持久，则为 <b>present</b> ，否则为 <b>absent</b> 。
type	标识连接类型。有效的值有 <b>etherne</b> t、 <b>bridge</b> 、 <b>bond</b> 、 <b>team</b> 、 <b>vlan</b> 、 <b>macvlan</b> 和 <b>infiniband</b> 。
autoconnect	确定连接是否自动启动。
mac	将连接限制为在具有特定 MAC 地址的设备上使用。
interface_name	将连接配置集显示为供特定接口使用。
zone	为接口配置 FirewallD 区域。
ip	确定连接的 IP 配置。支持诸如 <b>address</b> （指定静态 IP 地址）或 <b>dns</b> （配置 DNS 服务器）等选项。

以下示例使用了上述部分选项：

```
network_connections:
- name: eth0 ①
  persistent_state: present ②
  type: ethernet ③
  autoconnect: yes ④
  mac: 00:00:5e:00:53:5d ⑤
  ip:
    address:
      - 172.25.250.40/24 ⑥
  zone: external ⑦
```

- ① 使用 **eth0** 作为连接名称。
- ② 使连接持久。这是默认值。
- ③ 将连接类型设置为 **etherne**t。
- ④ 在引导时自动启动连接。这是默认值。
- ⑤ 限制为具有此 MAC 地址的设备使用该连接。
- ⑥ 为连接配置 **172.25.250.40/24** IP 地址。
- ⑦ 将 **external** 区域配置为连接的 FirewallD 区域。

要使用网络系统角色，您需要在 playbook 的 **roles** 子句下指定角色名称，如下方所示：

```
- name: NIC Configuration
hosts: webservers
vars:
```

```

network_connections:
  - name: ens4
    type: ethernet
    ip:
      address:
        - 172.25.250.30/24
    roles:
      - rhel-system-roles.network

```

您可以使用 **vars** 子句指定网络角色的变量，如上例中所示，也可在 **group\_vars** 或 **host\_vars** 目录下创建包含这些变量的 YAML 文件，具体取决于您的使用情形。

## 使用模块配置网络

作为 **network** 系统角色的替代选择，红帽 Ansible 引擎包含了支持系统上网络配置的一系列模块。**nmcli** 模块支持管理网络连接和设备。此模块支持配置网络接口组合和绑定，以及 IPv4 和 IPv6 寻址。

下表列出了 **nmcli** 模块的一些参数。

参数名称	描述
conn_name	配置连接名称。
autoconnect	启用在引导时自动激活连接。
dns4	配置 IPv4 的 DNS 服务器（最多 3 个）。
gw4	为接口配置 IPv4 网关。
ifname	要绑定到连接的接口。
ip4	接口的 IP 地址 (IPv4)。
state	启用或禁用该网络接口。
type	设备或网络连接的类型。

以下示例为网络连接和设备配置静态 IP 配置。

```

- name: NIC configuration
  nmcli:
    conn_name: ens4-conn ①
    ifname: ens4 ②
    type: ethernet ③
    ip4: 172.25.250.30/24 ④
    gw4: 172.25.250.1 ⑤
    state: present ⑥

```

- ① 配置 **ens4-conn** 作为连接名称。
- ② 将 **ens4-conn** 连接绑定到 **ens4** 网络接口。
- ③ 将网络接口配置为 **ethernet**。
- ④ 在接口上配置 **172.25.250.30/24** IP 地址。
- ⑤ 将网关设置为 **172.25.250.1**。

⑥ 确保连接可用。

`hostname` 模块可在不修改 `/etc/hosts` 文件的前提下设置受管主机的主机名。此模块使用 `name` 参数来指定新的主机名，如下方任务所示：

```
- name: Change hostname
  hostname:
    name: managedhost1
```

`firewalld` 模块支持受管主机上的 FirewallD 管理。此模块支持为服务和端口配置 FirewallD 规则。它还支持区域管理，包括将网络接口和规则与特定区域关联。

以下任务演示了如何为默认区域 (`public`) 中的 `http` 服务创建 FirewallD 规则。该任务将规则配置为永久规则，并确保它处于活动状态。

```
- name: Enabling http rule
  firewalld:
    service: http
    permanent: yes
    state: enabled
```

此任务在 `external` FirewallD 区域中配置 `eth0`。

```
- name: Moving eth0 to external
  firewalld:
    zone: external
    interface: eth0
    permanent: yes
    state: enabled
```

下表列出了 `firewalld` 模块的一些参数。

参数名称	描述
<code>interface</code>	要使用 FirewallD 管理的接口名称。
<code>port</code>	端口或端口范围。使用端口/协议或端口-端口/协议格式。
<code>rich_rule</code>	FirewallD 的富规则。
<code>service</code>	要使用 FirewallD 管理的服务名称。
<code>source</code>	要使用 FirewallD 管理的来源网络。
<code>zone</code>	FirewallD 区域。
<code>state</code>	启用或禁用 FirewallD 配置。
<code>type</code>	设备或网络连接的类型。

## 网络配置的 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...
    }
}
```

受管主机的所有网络接口位于 `ansible_interfaces` 元素下。您可以使用 `setup` 模块的 `gather_subset=network` 参数，将事实限制为 `network` 子集中包含的事实。`setup` 模块的 `filter` 选项支持根据 shell 样式的通配符进行精细过滤。

```
[user@controlnode ~]$ ansible webservers -m setup \
> -a 'gather_subset=network filter=ansible_interfaces'
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_interfaces": [
            "ens4",
            "lo",
            "ens3"
        ],
        "changed": false
    }
}
```

上一命令显示了受管主机 `host.lab.example.com` 上可用的三个网络接口：`lo`、`ens3` 和 `ens4`。

您可以使用 `setup` 模块的 `ansible_NIC_name` 过滤器检索有关网络接口配置的其他信息。例如，要检索置 `ens4` 网络接口的配置，可使用 `ansible_ens4` 过滤器。

```
[user@controlnode ~]$ ansible webservers -m setup \
> -a 'gather_subset=network filter=ansible_ens4'
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_ens4": {
            "active": true,
            "device": "ens4",
            "features": {},
            "hw_timestamp_filters": [],
            "ipv4": {
                "address": "172.25.250.30",
                "broadcast": "172.25.250.255",
                "netmask": "255.255.255.0",
                "network": "172.25.250.0"
            },
            "ipv6": [
                {
                    "address": "fe80::5b42:8c94:1fc7:4ae"
                }
            ]
        }
    }
}
```

```

        "prefix": "64",
        "scope": "link"
    },
],
"macaddress": "52:54:00:01:fa:0a",
"module": "virtio_net",
"mtu": 1500,
"pciid": "virtio1",
"promisc": false,
"speed": -1,
"timestamping": [
    "tx_software",
    "rx_software",
    "software"
],
"type": "ether"
},
},
"changed": false
}

```

上一命令显示其他配置详细信息，如 IPv4 和 IPv6 的 IP 地址配置、关联的设备及其类型。

下表列出了 `network` 子集的一些可用事实。

事实名称	描述
<code>ansible_dns</code>	包括 DNS 服务器 IP 地址和搜索域。
<code>ansible_domain</code>	包括受管主机的子域。
<code>ansible_all_ipv4_addresses</code>	包括受管主机上配置的所有 IPv4 地址。
<code>ansible_all_ipv6_addresses</code>	包括受管主机上配置的所有 IPv6 地址。
<code>ansible_fqdn</code>	包括受管主机的 FQDN。
<code>ansible_hostname</code>	包括非限定主机名，即 FQDN 中第一个句点之前的字符串。
<code>ansible_nodename</code>	包括系统报告的受管主机的主机名。



### 注意

Ansible 还提供 `inventory_hostname` 变量，它可包含 Ansible 清单文件中配置的主机名。



### 参考文献

#### 知识库：红帽企业 Linux (RHEL) 系统角色

<https://access.redhat.com/articles/3050101>

#### Linux 系统角色

<https://linux-system-roles.github.io/>

#### nmcli 模块文档

[https://docs.ansible.com/ansible/latest/modules/nmcli\\_module.html/](https://docs.ansible.com/ansible/latest/modules/nmcli_module.html/)

#### hostname 模块文档

[https://docs.ansible.com/ansible/latest/modules/hostname\\_module.html/](https://docs.ansible.com/ansible/latest/modules/hostname_module.html/)

#### firewalld 模块文档

[https://docs.ansible.com/ansible/latest/modules/firewalld\\_module.html/](https://docs.ansible.com/ansible/latest/modules/firewalld_module.html/)

## ► 指导练习

# 管理网络配置

在本练习中，您将调整受管主机的网络配置，并在由模板创建的文件中收集相关的信息。

## 成果

您应能够在受管主机上配置网络设置和名称解析，并收集网络相关的 Ansible 事实。

## 在你开始之前

从 workstation 运行 **lab system-network start** 脚本，以配置本练习的环境。此脚本将创建 **system-network** 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-network start
```

### ► 1. 检查 /home/student/system-network 目录中的清单文件。

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

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

- 1.2. 验证 **servera.lab.example.com** 是否是 **webservers** 主机组的一部分。此服务器具有一个备用网络接口。

```
[student@workstation system-network]$ cat inventory  
[webservers]  
servera.lab.example.com
```

### ► 2. 使用 **ansible-galaxy** 命令验证系统角色是否可用。如果没有角色可用，您需要安装 **rhel-system-roles** 软件包。

```
[student@workstation system-network]$ ansible-galaxy list  
# /usr/share/ansible/roles  
- linux-system-roles.kdump, (unknown version)  
- linux-system-roles.network, (unknown version)  
- linux-system-roles.postfix, (unknown version)  
- linux-system-roles.selinux, (unknown version)  
- linux-system-roles.timesync, (unknown version)  
- rhel-system-roles.kdump, (unknown version)  
- rhel-system-roles.network, (unknown version)  
- rhel-system-roles.postfix, (unknown version)  
- rhel-system-roles.selinux, (unknown version)
```

```
- rhel-system-roles.timesync, (unknown version)
# /etc/ansible/roles
[WARNING]: - the configured path /home/student/.ansible/roles does not exist.
```

- 3. 创建一个 playbook，以使用 `linux-system-roles.network` 角色为 `servera.lab.example.com` 上的备用网络接口 `ens4` 配置 **172.25.250.30** IP 地址。

- 3.1. 创建一个 playbook `playbook.yml`，其中包含一个以 `webservers` 主机组为目标的 play。将 `rhel-system-roles.network` 角色包含到 play 的 `roles` 部分。

```
---
- name: NIC Configuration
  hosts: webservers

  roles:
    - rhel-system-roles.network
```

- 3.2. 检查 `README.md` 文件的 Role Variables 部分中的 `rhel-system-roles.network` 角色。确定相关的角色变量，以便为 `ens4` 网络接口配置 **172.25.250.30** IP 地址。

```
[student@workstation system-network]$ cat \
> /usr/share/doc/rhel-system-roles/network/README.md
...output omitted...
Setting the IP configuration:
...output omitted...
```

- 3.3. 创建 `group_vars/webservers` 子目录。

```
[student@workstation system-network]$ mkdir -pv group_vars/webservers
mkdir: created directory 'group_vars'
mkdir: created directory 'group_vars/webservers'
```

- 3.4. 创建一个新文件 `network.yml` 来定义角色变量。由于这些变量值将应用到 `webservers` 主机组中的主机，您需要在 `group_vars/webservers` 目录中创建该文件。添加变量定义以支持配置 `ens4` 网络接口。该文件现在包含：

```
---
network_connections:
  - name: ens4
    type: ethernet
    ip:
      address:
        - 172.25.250.30/24
```

- 3.5. 运行 playbook，以在 `servera` 上配置第二网络接口。

```
[student@workstation system-network]$ ansible-playbook playbook.yml
PLAY [NIC Configuration] ****
TASK [Gathering Facts] ****
```

```

ok: [servera.lab.example.com]

TASK [rhel-system-roles.network : Check which services are running] ****
ok: [servera.lab.example.com]

TASK [rhel-system-roles.network : Check which packages are installed] ****
ok: [servera.lab.example.com]

TASK [rhel-system-roles.network : Print network provider] ****
ok: [servera.lab.example.com] => {
    "msg": "Using network provider: nm"
}

TASK [rhel-system-roles.network : Install packages] ****
skipping: [servera.lab.example.com]

TASK [rhel-system-roles.network : Enable network service] ****
ok: [servera.lab.example.com]

TASK [rhel-system-roles.network : Configure networking connection profiles] ****
...output omitted...

changed: [servera.lab.example.com]

TASK [rhel-system-roles.network : Re-test connectivity] ****
ok: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=7    changed=1    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0

```

- 4. 在一个Ansible临时命令中，使用Ansible `setup`模块来验证 `servera` 上的 `ens4` 网络接口配置是否正确。

4.1. 使用 `setup` Ansible 模块，列出可供 `servera` 使用的所有 Ansible 事实。使用 `-a 'filter=ansible_ens4'` 选项，过滤 `ens4` 网络接口的结果。验证 `ens4` 网络接口是否使用了 **172.25.250.30** IP 地址。配置 IP 地址可能需要最多一分钟。

```
[student@workstation system-network]$ ansible webservers -m setup \
> -a 'filter=ansible_ens4'
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_ens4": {
            ...output omitted...
            "ipv4": {
                "address": "172.25.250.30",
                "broadcast": "172.25.250.255",
                "netmask": "255.255.255.0",
                "network": "172.25.250.0"
            },
            ...output omitted...
        }
    }
}
```

## 完成

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

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

本引导式练习到此结束。

## ▶ 开放研究实验

# 自动执行 LINUX 管理任务

### 任务执行清单

在本实验中，您将使用 playbook 配置和执行对受管主机的管理任务。

### 成果

您应能够创建用于在受管主机上配置软件存储库、用户和组、逻辑卷、cron 作业和其他网络接口的 playbook。

### 在你开始之前

在 `workstation` 上运行实验起始脚本，以确认环境已就绪，可开始实验。该脚本将创建名为 `system-review` 的工作目录，并为它填充 Ansible 配置文件、主机清单和实验文件。

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

1. 创建一个 playbook 并在 `webservers` 主机组中执行，以配置位于 `http://materials.example.com/yum/repository` 的 Yum 内部存储库，并安装该存储库中提供的 `example-motd` 软件包。所有 RPM 软件包都使用组织 GPG 密钥对签名。GPG 公钥位于 `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`。
2. 创建一个 playbook 并在 `webservers` 主机组中执行，以创建 `webadmin` 用户组并向该组添加 `ops1` 和 `ops2` 两个用户。
3. 创建一个 playbook 并在 `webservers` 主机组中执行，以使用 `/dev/vdb` 设备创建名为 `apache-vg` 的卷组。此 playbook 还将创建两个逻辑卷，取名为 `content-lv` 和 `logs-lv`，它们都由 `apache-vg` 卷组支持。最后，它在每个逻辑卷上创建 XFS 文件系统，并将 `content-lv` 和 `logs-lv` 逻辑卷分别挂载到 `/var/www` 和 `/var/log/httpd`。实验室脚本在 `~/system-review` 中填充两个文件，`storage.yml` 提供 playbook 的初始框架，`storage_vars.xml` 则为不同模块所需的所有变量提供值。
4. 创建一个 playbook 并在 `webservers` 主机组中执行，以使用 `cron` 模块来创建调度周期性 cron 作业的 `/etc/cron.d/disk_usage` crontab 文件。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。该作业应将当前磁盘使用量附加到 `/home/devops/disk_usage` 文件中。
5. 创建一个 playbook 并在 `webservers` 主机组中执行，以使用 `linux-system-roles.network` 角色将 **172.25.250.40/24** IP 地址配置给备用网络接口 `ens4`。
6. 在 `workstation` 上运行 `lab system-review grade` 以评测您的工作。

```
[student@workstation ~]$ lab system-review grade
```

### 完成

从 `workstation`，运行 `lab system-review finish` 脚本来清理本实验中创建的资源。

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

本实验到此结束。

## ► 解决方案

# 自动执行 LINUX 管理任务

### 任务执行清单

在本实验中，您将使用 playbook 配置和执行对受管主机的管理任务。

### 成果

您应能够创建用于在受管主机上配置软件存储库、用户和组、逻辑卷、cron 作业和其他网络接口的 playbook。

### 在你开始之前

在 `workstation` 上运行实验起始脚本，以确认环境已就绪，可开始实验。该脚本将创建名为 `system-review` 的工作目录，并为它填充 Ansible 配置文件、主机清单和实验文件。

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

1. 创建一个 playbook 并在 `webservers` 主机组中执行，以配置位于 `http://materials.example.com/yum/repository` 的 Yum 内部存储库，并安装该存储库中提供的 `example-motd` 软件包。所有 RPM 软件包都使用组织 GPG 密钥对签名。GPG 公钥位于 `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`。
  - 1.1. 以 `student` 用户身份，在 `workstation` 上更改到 `/home/student/system-review` 工作目录。

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

- 1.2. 创建 `repo_playbook.yml` playbook，它将在 `webservers` 主机组中的受管主机上运行。添加一个任务，以使用 `yum_repository` 模块来确保远程主机上配置了内部 yum 存储库。确保：

- 存储库的配置存储在文件 `/etc/yum.repos.d/example.repo` 中
- 存储库 ID 为 `example-internal`
- 基础 URL 为 `http://materials.example.com/yum/repository`
- 存储库已配置为检查 RPM GPG 签名
- 存储库描述为 `Example Inc. Internal YUM repo`

该 playbook 包含如下内容：

```
---
- name: Repository Configuration
  hosts: webservers
```

```

tasks:
  - name: Ensure Example Repo exists
    yum_repository:
      name: example-internal
      description: Example Inc. Internal YUM repo
      file: example
      baseurl: http://materials.example.com/yum/repository/
      gpgcheck: yes

```

- 1.3. 在 play 中添加第二个任务，以使用 `rpm_key` 模块来确保远程主机上存在存储库公钥。存储库公钥 URL 为 `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`。

第二个任务如下所示：

```

- name: Ensure Repo RPM Key is Installed
  rpm_key:
    key: http://materials.example.com/yum/repository/RPM-GPG-KEY-example
    state: present

```

- 1.4. 添加第三个任务，以安装 Yum 内部存储库中提供的 `example-motd` 软件包。

第三个任务如下所示：

```

- name: Install Example motd package
  yum:
    name: example-motd
    state: present

```

- 1.5. 执行 playbook：

```

[student@workstation system-review]$ ansible-playbook repo_playbook.yml

PLAY [Repository Configuration] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]

TASK [Ensure Example Repo exists] ****
changed: [serverb.lab.example.com]

TASK [Ensure Repo RPM Key is Installed] ****
changed: [serverb.lab.example.com]

TASK [Install Example motd package] ****
changed: [serverb.lab.example.com]

PLAY RECAP ****
serverb.lab.example.com : ok=4    changed=3    unreachable=0    failed=0

```

2. 创建一个 playbook 并在 `webservers` 主机组中执行，以创建 `webadmin` 用户组并向该组添加 `ops1` 和 `ops2` 两个用户。