Top　DAY1

1 案例 1：环境准备

1.1 问题

本案例要求准备 ansible 的基础环境：

- 启动 6 台虚拟机
- 禁用 selinux 和 firewalld
- 编辑/etc/hosts
- 配置 yum 扩展源并在管理节点安装 ansible

1.2 方案

此方案需要准备六台主机，1 台管理主机，5 台托管主机，以实现批量程序部署，批量运行命令等功能，具体要求如表-1 所示：

表-1

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：基础环境准备

1）启动 6 台虚拟机，由于已经讲过怎么创建，这里不再在案例里体现

2）真机配置 yum 仓库

```
[root@room9pc01 ~]# tar -xf ansible_soft.tar.xz
[root@room9pc01 ~]# cd ansible_soft/
[root@room9pc01 ansible_soft]# mkdir /var/ftp/ansible
[root@room9pc01 ansible_soft]# cp * /var/ftp/ansible
[root@room9pc01 ansible_soft]# createrepo   /var/ftp/ansible
Spawning worker 0 with 1 pkgs
Spawning worker 1 with 1 pkgs
Spawning worker 2 with 1 pkgs
Spawning worker 3 with 1 pkgs
Spawning worker 4 with 1 pkgs
Spawning worker 5 with 1 pkgs
Workers Finished
Saving Primary metadata
Saving file lists metadata
Saving other metadata
Generating sqlite DBs
```

```
Sqlite DBs complete
```

3）修改主机名（容易区分，6 台机器都需要修改）这里以 ansible 主机为例子

```
[root@localhost ~]# echo ansible > /etc/hostname
[root@localhost ~]# hostname ansible
```

4）配置 ip（6 台机器都需要配置），这里以 ansible 主机为例子

```
[root@localhost ~]# vim /etc/sysconfig/network-scripts/ifcfg-eth0
# Generated by dracut initrd
DEVICE="eth0"
ONBOOT="yes"
IPV6INIT="no"
IPV4_FAILURE_FATAL="no"
NM_CONTROLLED="no"
TYPE="Ethernet"
BOOTPROTO="static"
IPADDR=192.168.1.51
PREFIX=24
GATEWAY=192.168.1.254
[root@localhost ~]# systemctl restart network
[root@localhost ~]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.51  netmask 255.255.255.0  broadcast 192.168.1.255
        ether 52:54:00:b2:69:9e  txqueuelen 1000  (Ethernet)
        RX packets 234  bytes 16379 (15.9 KiB)
        RX errors 0  dropped 36  overruns 0  frame 0
        TX packets 31  bytes 2618 (2.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

5）配置 yum 客户端，在管理节点 ansible 上面配置

```
[root@ansible ~]# vim /etc/yum.repos.d/local.repo
[local_repo]
name=CentOS-$releasever - Base
baseurl="ftp://192.168.1.254/system"
enabled=1
gpgcheck=1
[local]
name=local
baseurl="ftp://192.168.1.254/ansible"
enabled=1
```

```
gpgcheck=0
[root@ansible ~]# yum clean all
[root@ansible ~]# yum repolist
[root@ansible ~]# yum -y install ansible
[root@ansible ~]# ansible --version
ansible 2.4.2.0            //显示版本说明安装成功
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.5 (default, Aug  4 2017, 00:39:18) [GCC 4.8.5 20150623 (Red Hat 4.8.5-16)]
```

6）请在 6 台主机上面配置/etc/hosts，这里以 ansible 主机为例子

```
[root@ansible ansible]# cat /etc/hosts
192.168.1.51 ansible
192.168.1.52 web1
192.168.1.53 web2
192.168.1.54 db1
192.168.1.55 db2
192.168.1.56 cache
```

2 案例 2：主机定义与分组：

2.1 问题

本案例要求：

熟悉 ansible 配置文件
定义主机，分组和子组练习
自定义文件，多配置路径练习

2.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：ansible.cfg 配置文件

```
[root@ansible ~]# cd /etc/ansible/
[root@ansible ansible]# ls
ansible.cfg   hosts   roles
[root@ansible ansible]# vim ansible.cfg
```

```
    #inventory        = /etc/ansible/hosts       //指定分组文件路径，主机的分组文件
hosts
    [selinux]           //组名称，selinux 的相关选项在这个下面配置
    ...
    [colors]            //组名称，colors 的相关选项在这个下面配置
    ...
```

步骤二：定义主机，分组和子组练习

1）静态主机的定义

```
    [root@ansible ansible]# vim hosts
    [web]
    web1
    web2
    [db]
    db[1:2]                         //1:2 为 db1 到 db2 两台主机，1:20 为 db1 到 db20
多台主机
    [other]
    cache
    [root@ansible ansible]# ansible web --list-host     //显示 web 组的主机
      hosts (2):
        web1
        web2
    [root@ansible ansible]# ansible db --list-host
      hosts (2):
        db1
        db2
    [root@ansible ansible]# ansible other --list-host
      hosts (1):
        cache
    [root@ansible ansible]# ansible all --list-host     //显示所有组的主机
      hosts (5):
        web1
        web2
        cache
        db1
        db2
```

2）直接测试

```
    [root@ansible ansible]# ansible cache -m ping
    //测试是否可以连接，若失败颜色为红色
    cache | UNREACHABLE! => {
        "changed": false,
        "msg": "Failed to connect to the host via ssh: ssh: Could not resolve hostname cache: Name or service not known\r\n",
        "unreachable": true
    }
```

3）修改后测试

```
    [root@ansible ansible]# vi hosts
    [other]
    cache ansible_ssh_user="root" ansible_ssh_pass="a"
    [root@ansible ansible]# ansible other -m ping     //测试成功，颜色为绿色
    cache | SUCCESS => {
        "changed": false,
        "ping": "pong"
    }
```

4）不检测主机的 sshkey，在第一次连接的时候不用输入 yes

```
    [root@ansible ansible]# vim ansible.cfg
    61 host_key_checking = False
    [root@ansible ansible]# vim hosts
    [web]
    web1
    web2
    [web:vars]       //web 组:变量(vars 不改)，web 组的多台机器共用一个用户名和密码
    ansible_ssh_user="root"
    ansible_ssh_pass="a"
    [root@ansible ansible]# ansible web -m ping
    web2 | SUCCESS => {
        "changed": false,
        "ping": "pong"
    }
    web1 | SUCCESS => {
        "changed": false,
        "ping": "pong"
    }
```

步骤三：定义子组

```
    [root@ansible ansible]# vi hosts
    [app:children]       //指定子分组(app 可改:children 不改)，web，db 是提前分好的组
```

```
web
db
[app:vars]
ansible_ssh_user="root"
ansible_ssh_pass="a"
[root@ansible ansible]# ansible app --list-host             //查看
  hosts (4):
    web1
    web2
    db1
    db2
[root@ansible ansible]# ansible app   -m ping             //测试
web1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
web2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

步骤四：多路径练习

自定义的 ansible 文件只在当前路径生效

1）多路径

```
[root@ansible ~]# mkdir aaa
[root@ansible ~]# cd aaa/
[root@ansible aaa]# vim myhost
[app1]
web1
db1
[app2]
web2
db2
```

```
[app:children]
app1
app2
[other]
cache
[app:vars]
ansible_ssh_user="root"
ansible_ssh_pass="a"
[root@ansible aaa]# touch ansible.cfg
[root@ansible aaa]# grep -Ev "^#|^$" /etc/ansible/ansible.cfg
[defaults]
roles_path      = /etc/ansible/roles:/usr/share/ansible/roles
host_key_checking = False
[inventory]
[privilege_escalation]
[paramiko_connection]
[ssh_connection]
[persistent_connection]
[accelerate]
[selinux]
[colors]
[diff]
[root@ansible aaa]# vim ansible.cfg
[defaults]
inventory = myhost
host_key_checking = False
```

2）测试结果

```
[root@ansible aaa]# ansible app1 -m ping
web1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
[root@ansible aaa]# ansible app -m ping
web1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db1 | SUCCESS => {
      "changed": false,
      "ping": "pong"
}
db2 | SUCCESS => {
      "changed": false,
      "ping": "pong"
}
web2 | SUCCESS => {
      "changed": false,
      "ping": "pong"
}
[root@ansible aaa]# ansible   app --list-host
   hosts (4):
      web1
      db1
      web2
      db2
[root@ansible aaa]# cd
[root@ansible ~]# ansible   app1 --list-host     //切换到别的目录，测试失败
  [WARNING]: Could not match supplied host pattern, ignoring: app1
  [WARNING]: No hosts matched, nothing to do
   hosts (0):
```

3 案例 3：动态主机

3.1 问题

本案例要求：

　　脚本输出主机列表

3.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：脚本输出主机列表

```
[root@ansible ~]# cd aaa
[root@ansible aaa]# vim host.py
#!/usr/bin/python
import json
hostlist = {}
hostlist["bb"] = ["192.168.1.52", "192.168.1.53"]
hostlist["192.168.1.54"] = {
         "ansible_ssh_user":"root","ansible_ssh_pass":"pwd"
```

```
                }
hostlist["aa"] = {
                "hosts" : ["192.168.1.55", "192.168.1.56"],
                "vars"   : {
                        "ansible_ssh_user":"root","ansible_ssh_pass":"pwd"
                }
}
print(json.dumps(hostlist))
[root@ansible aaa]# chmod 755 ./host.py
```

步骤二：脚本输出样例（这样写输出的结果有些乱）

```
[root@ansible aaa]# ./host.py
{"aa": {"hosts": ["192.168.1.55", "192.168.1.56"], "vars": {"ansible_ssh_user": "root", "ansible_ssh_pass": "a"}}, "192.168.1.54": {"ansible_ssh_user": "root", "ansible_ssh_pass": "a"}, "bb": ["192.168.1.52", "192.168.1.53"]}
```

步骤三：可以用 shell 脚本输出

```
 [root@ansible aaa]# vim my.sh
#!/bin/bash
echo '
{   "aa": {
            "hosts":
                        ["192.168.1.55", "192.168.1.56"],
           "vars": {
                        "ansible_ssh_user": "root",
                        "ansible_ssh_pass": "a"}
 },
}'
[root@ansible aaa]# chmod 755 my.sh
[root@ansible aaa]# ./my.sh
{   "aa": {
      "hosts":
            ["192.168.1.55", "192.168.1.56"],
           "vars": {
            "ansible_ssh_user": "root",
            "ansible_ssh_pass": "a"}
 },
}
[root@ansible aaa]# vim ansible.cfg
[defaults]
inventory = my.sh
```

```
host_key_checking = False
[root@ansible aaa]# ansible aa -m ping
192.168.1.55 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
192.168.1.56 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

步骤二：批量执行

1）查看负载

```
[root@ansible aaa]# ansible app -m command -a 'uptime'
db1 | SUCCESS | rc=0 >>
 11:35:52 up  1:59,  2 users,  load average: 0.00, 0.01, 0.01
web1 | SUCCESS | rc=0 >>
 11:35:52 up  2:00,  2 users,  load average: 0.00, 0.01, 0.02
db2 | SUCCESS | rc=0 >>
 11:35:53 up  1:59,  2 users,  load average: 0.00, 0.01, 0.03
web2 | SUCCESS | rc=0 >>
 11:35:52 up  1:59,  2 users,  load average: 0.00, 0.01, 0.02
```

2）查看时间

```
[root@ansible aaa]# ansible app -m command -a 'date +%F\ %T'
db1 | SUCCESS | rc=0 >>
2018-09-06 11:42:18
web1 | SUCCESS | rc=0 >>
2018-09-06 11:42:18
web2 | SUCCESS | rc=0 >>
2018-09-06 11:42:18
db2 | SUCCESS | rc=0 >>
2018-09-06 11:42:19
```

4 案例 4：批量部署证书文件

4.1 问题

本案例要求：

创建一对密钥
给所有主机部署密钥

4.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：批量部署证书文件，给所有主机部署密钥

1）创建密钥

```
 [root@ansible aaa]# cd /root/.ssh/
[root@ansible .ssh]# vi /etc/ansible/hosts
[web]
web1
web2
[db]
db[1:2]
[other]
cache
[root@ansible .ssh]# ansible all -m ping   //直接 ping 会报错
 [root@ansible .ssh]# ssh-keygen -t rsa -b 2048 -N ''   //创建密钥
```

2）给所有主机部署密钥

```
[root@ansible .ssh]# ansible all -m authorized_key -a "user=root exclusive=true manage_dir=true key='$(< /root/.ssh/id_rsa.pub)'" -k
SSH password:           //输入密码
[root@ansible .ssh]# ansible all -m ping   //成功
web2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
web1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
cache | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
[root@ansible .ssh]# ssh web1            //不需要输入密码,可以直接登陆
Last login: Thu Sep  6 11:49:00 2018 from 192.168.1.51
[root@web1 ~]#
```

5 案例 5：练习模块

5.1 问题

本案例要求：

练习使用 command , shell , raw, script 模块

5.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：练习模块

ansible-doc //模块的手册，相当于 man

ansible-doc -l //列出所有模块

ansible-doc 模块名 //查看指定模块的帮助信息

1）ping 模块

```
    [root@ansible .ssh]# ansible web1 -m ping
    web1 | SUCCESS => {
        "changed": false,
        "ping": "pong"
    }
```

2）command 模块

```
    [root@ansible .ssh]# ansible web1 -m command -a 'chdir=/tmp touch f1'
//创建成功
    [root@web1 ~]# cd /tmp/
    [root@web1 tmp]# ls           //在 web1 上面查看
    f1
```

3）shell 模块

```
    [root@ansible .ssh]# ansible web1 -m shell -a 'chdir=/tmp touch f2'    //创建成功
    [root@web1 ~]# cd /tmp/
    [root@web1 tmp]# ls       //在 web1 上面查看
    f2
```

4）raw 模块

```
    [root@ansible .ssh]# ansible web1 -m raw -a 'chdir=/tmp touch f3'
    //文件可以创建，但无法切换目录，文件在用户家目录下生成
    web1 | SUCCESS | rc=0 >>
    Shared connection to web1 closed.
    [root@web1 tmp]# cd /root/
    [root@web1 ~]# ls             //在 web1 上面查看
    f3
```

5）script 模块

对于太复杂的命令，可以写个脚本，然后用 script 模块执行

在 web1 主机上创建 zhangsan3 用户，修改 zhangsan3 的密码为 123456，设置 zhangsan3 第一次登陆必须修改密码

用命令写：

```
    [root@ansible .ssh]# ansible web1 -m shell -a 'useradd zhangsan3'
    [root@ansible .ssh]# ansible web1 -m shell -a 'echo 123456 | passwd --stdin zhangsan3'
    [root@ansible .ssh]# ssh -l zhangsan3 web1
    zhangsan3@web1's password:     //输入 zhangsan3 的密码
    [root@ansible .ssh]# ansible web1 -m shell -a 'chage -d 0 zhangsan3'
    [root@ansible .ssh]# ssh -l zhangsan3 web1
```

用脚本写，script 模块执行：

```
    [root@ansible .ssh]# vim user.sh
    #!/bin/bash
    useradd zhangsan3
    echo 123456 | passwd --stdin zhangsan3
    chage -d 0 zhangsan3
    echo
    [root@ansible .ssh]# ansible web1 -m script -a './user.sh'
    web1 | SUCCESS => {
        "changed": true,
            "rc": 0,
            "stderr": "Shared connection to web1 closed.\r\n",
            "stdout": "Changing password for user zhangsan3.\r\npasswd: all authentication tokens updated successfully.\r\n\r\n",
            "stdout_lines": [
                "Changing password for user zhangsan3.",
                "passwd: all authentication tokens updated successfully.",
                ""
            ]
        }
    [root@ansible .ssh]# ssh   -l lisi web1
    lisi@web1's password:
    You are required to change your password immediately (root enforced)
    Last login: Thu Sep   6 14:51:33 2018 from 192.168.1.51
    WARNING: Your password has expired.
    You must change your password now and login again!
    Changing password for user lisi.
    Changing password for lisi.
    (current) UNIX password:
```

6 案例 6：模块练习

6.1 问题

本案例要求：

- 使用 copy 模块同步数据
- 使用 lineinfile 模块编辑文件
- 使用 replace 模块修改文件

6.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：模块练习

1）使用 copy 模块同步数据

src：要复制到进程主机的文件在本地的地址,可以是绝对路径,也可以是相对路径。如果路径是一个目录,它将递归复制。在这种情况下,如果路径使用"/"来结尾,则只复制目录里的内容,如果没有使用"/"来结尾,则包含目录在内的整个内容全部复制,类似于 rsync

dest：必选项。进程主机的绝对路径,如果源文件是一个目录,那么该路径也必须是个目录

backup：在覆盖之前将原文件备份,备份文件包含时间信息。有两个选项:yes|no force：如果目标主机包含该文件,但内容不同,如果设置为 yes,则强制覆盖,如果为 no,则只有当目标主机的目标位置不存在该文件时,才复制。默认为 yes

```
 [root@ansible .ssh]# ansible all -m shell -a 'cat /etc/resolv.conf'
//查看/etc/resolv.conf
cache | SUCCESS | rc=0 >>
; generated by /usr/sbin/dhclient-script
nameserver 192.168.1.254
search localhost
db2 | SUCCESS | rc=0 >>
; generated by /usr/sbin/dhclient-script
nameserver 192.168.1.254
search localhost
web1 | SUCCESS | rc=0 >>
; generated by /usr/sbin/dhclient-script
nameserver 192.168.1.254
search localhost
web2 | SUCCESS | rc=0 >>
; generated by /usr/sbin/dhclient-script
nameserver 192.168.1.254
search localhost
db1 | SUCCESS | rc=0 >>
; generated by /usr/sbin/dhclient-script
nameserver 192.168.1.254
search localhost
[root@ansible .ssh]# vi /etc/resolv.conf
nameserver 172.40.1.10
[root@ansible .ssh]# ansible all -m copy -a 'src=/etc/resolv.conf dest=/etc/resolv.conf'    //复制本机的 resolv.conf 到其他主机
[root@ansible .ssh]# ansible all -m shell -a 'cat /etc/resolv.conf'
//查看有 nameserver 172.40.1.10
[root@ansible ~]# mkdir aa
[root@ansible ~]# ansible all -m copy -a 'src=/root/aa dest=/root/a.log'
//复制本机的目录/root/aa 到其他机器的/root/a.log，复制目录只能少数批量执行同步
[root@ansible ~]# ansible all -m shell -a 'ls -ld /root'
db2 | SUCCESS | rc=0 >>
dr-xr-x---. 4 root root 167 Sep  6 11:48 /root
web2 | SUCCESS | rc=0 >>
dr-xr-x---. 4 root root 167 Sep  6 11:48 /root
cache | SUCCESS | rc=0 >>
dr-xr-x---. 4 root root 177 Sep  6 14:35 /root
db1 | SUCCESS | rc=0 >>
dr-xr-x---. 4 root root 167 Sep  6 11:48 /root
```

```
web1 | SUCCESS | rc=0 >>
dr-xr-x---. 4 root root 177 Sep   6 14:35 /root
```

2）使用 lineinfile 模块编辑文件

以行为基础，整行修改(整行被替换掉)

```
[root@ansible ~]# ansible cache -m lineinfile \
-a   'path=/etc/sysconfig/network-scripts/ifcfg-eth0 \
regexp="^ONBOOT=" line="ONBOOT=\"no\""'
cache | SUCCESS => {
      "backup": "",
      "changed": true,
      "msg": "line replaced"
}
```

3）使用 replace 模块修改文件

修改文件的某一部分(替换一行中匹配的内容)，以正则表达式匹配为基础修改

```
[root@ansible ~]# ansible cache -m replace   -a \
  'path=/etc/sysconfig/network-scripts/ifcfg-eth0 \
regexp="^(ONBOOT=).*" replace="\1\"yes\""'
cache | SUCCESS => {
      "changed": true,
      "msg": "1 replacements made"
}
```

7 案例 7：综合练习

7.1 问题

本案例要求：

安装 Apache 并修改监听端口为 8080
修改 ServerName 配置，执行 apachectl -t 命令不报错
设置默认主页 hello world
启动服务并设开机自启

7.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：熟悉模块

1）yum 模块

```
 [root@ansible ~]# ansible other -m yum -a 'name="lrzsz" state=removed'
//lrzsz 软件包名，removed=absent 删除
[root@ansible ~]# ansible other -m yum -a 'name="lrzsz,lftp" state=installed'
//安装多个软件包，不写 state 默认为安装
```

2)service 模块

```
[root@ansible ~]# ansible other -m service -a 'name="sshd" enabled="yes"  state="started"' //sshd 服务名，开机启动同时启动这个服务
```

3）setup 模块

filter 过滤指定的关键字（可以过滤到我们需要的信息）

```
[root@ansible ~]# ansible cache -m setup -a 'filter=os'
cache | SUCCESS => {
    "ansible_facts": {},
    "changed": false
}
[root@ansible ~]# ansible cache -m setup -a 'filter=ansible_distribution'
cache | SUCCESS => {
    "ansible_facts": {
        "ansible_distribution": "CentOS"
    },
    "changed": false
}
```

步骤二：安装 Apache

1）安装 Apache 服务设置开机自启

```
[root@ansible ~]# ansible cache -m yum -a 'name=httpd state=installed'
[root@ansible ~]# ansible cache -m service -a 'name=httpd enabled=yes state=started'
```

2）修改端口号为 8080

```
[root@ansible ~]# ssh cache
Last login: Thu Sep  6 15:30:33 2018 from 192.168.1.51
[root@cache ~]# cat  /etc/httpd/conf/httpd.conf | grep Listen
Listen 80
```

```
[root@ansible ~]# ansible cache -m lineinfile -a 'path="/etc/httpd/conf/httpd.conf" regexp="^Listen " line="Listen 8080"'cache | SUCCESS => {
        "backup": "",
        "changed": true,
        "msg": "line replaced"
    }
    [root@ansible ~]# ssh cache
    Listen 8080
```

步骤三：修改 ServerName 配置，执行 apachectl -t 命令不报错

1）没有修改之前

```
[root@cache ~]# apachectl -t   //有报错
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 192.168.1.56. Set the 'ServerName' directive globally to suppress this message
Syntax OK
```

2）修改之后

```
[root@ansible ~]# ansible cache -m lineinfile -a 'path="/etc/httpd/conf/httpd.conf" regexp="^ServerName " line="ServerName 0.0.0.0"'
cache | SUCCESS => {
    "backup": "",
    "changed": true,
    "msg": "line added"
}
[root@ansible ~]# ssh cache
Last login: Thu Sep  6 15:36:08 2018 from 192.168.1.51
[root@cache ~]# apachectl -t
Syntax OK
```

步骤四：设置默认主页为 hello world

```
[root@ansible ~]# ansible cache -m copy -a 'src=/root/index.html dest=/var/www/html/index.html'    ///root/index.html 这个页面可以自己写
cache | SUCCESS => {
    "changed": true,
    "checksum": "22596363b3de40b06f981fb85d82312e8c0ed511",
    "dest": "/var/www/html/index.html",
    "gid": 0,
        "group": "root",
        "md5sum": "6f5902ac237024bdd0c176cb93063dc4",
        "mode": "0644",
        "owner": "root",
        "size": 12,
        "src": 
"/root/.ansible/tmp/ansible-tmp-1536219767.29-30682157793478/source",
        "state": "file",
        "uid": 0
    }
```

Top
NSD ARCHITECTURE DAY02

- 练习 1：playbook 练习
- 案例 2：变量练习
- 案例 3：handlers 练习
- 案例 4：编写 playbook

1 练习 1：playbook 练习

1.1 问题

本案例要求：

- 安装 Apache 并修改监听端口为 8080
- 修改 ServerName 配置，执行 apachectl -t 命令不报错
- 设置默认主页 hello world
- 启动服务并设开机自启

1.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：playbook 的 ping 脚本检测

```
[root@ansible ansible]# vim ping.yml
---
- hosts: all
  remote_user: root
  tasks:
     - ping:
[root@ansible ansible]# ansible-playbook ping.yml   //输出结果
```

```
    PLAY                                                                    [all]
*******************************************************************
    TASK                          [Gathering                              Facts]
*******************************************************
    ok: [web1]
    ok: [web2]
    ok: [cache]
    ok: [db1]
    ok: [db2]
    TASK                                                                   [ping]
******************************************************************
    ok: [db1]
    ok: [web2]
    ok: [cache]
    ok: [web1]
    ok: [db2]
    PLAY                                                                  RECAP
*******************************************************************
    cache                          : ok=2      changed=0      unreachable=0
failed=0
    db1                            : ok=2      changed=0      unreachable=0
failed=0
    db2                            : ok=2      changed=0      unreachable=0
failed=0
    web1                           : ok=2      changed=0      unreachable=0
failed=0
    web2                           : ok=2      changed=0      unreachable=0
failed=0
```

注意：如果检测的时候出错，会在当前的目录生成一个新的文件（以.retry 结尾），可以去这个文件里面看是哪个主机的错

步骤二：用 playbook 安装 Apache,修改端口，配置 ServerName，修改主页，设置开机自启

```
    [root@ansible ansible]# vim http.yml
    ---
    - hosts: cache
      remote_user: root
      tasks:
        - name: install one specific version of Apache
          yum:
            name: httpd          //安装 Apache
            state: installed
        - lineinfile:
            path: /etc/httpd/conf/httpd.conf
            regexp: '^Listen '
            line: 'Listen 8080'          //修改端口为8080
        - replace:
            path: /etc/httpd/conf/httpd.conf
            regexp: '^#(ServerName).*'          //配置ServerName
            replace: '\1 localhost'
        - service:
            name: httpd
            enabled: yes          //开机自启
            state: restarted
        - copy:
            src: /root/index.html          //修改主页，可以自己写个页面
            dest: /var/www/html/index.html
    [root@ansible ansible]# curl 192.168.1.56:8080
    hello world
    [root@ansible ansible]# ssh cache
    Last login: Fri Sep  7 09:32:05 2018 from 192.168.1.51
    [root@cache ~]# apachectl -t
    Syntax OK
```

2 案例2：变量练习

2.1 问题

本案例要求熟悉playbook进阶：

练习使用user模块添加用户
练习使用变量简化task，让play通用性更强
练习使用过滤器

2.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：使用user模块添加用户，并修改密码

```
    [root@ansible ansible]# vim user.yml
    ---
    - hosts: cache
      remote_user: root
      vars:
        username: xiaoming
      tasks:
```

```
      - name: create user "{{username}}"
        user: group=wheel uid=1000 name={{username}}
      - shell: echo 123456 | passwd --stdin xiaoming
      - shell: chage -d 0 {{username}}
[root@ansible ansible]# ansible-playbook user.yml    //执行结果
PLAY                                                              [cache]
*****************************************************************
TASK                          [Gathering                          Facts]
*******************************************************
ok: [cache]
TASK        [create        user        "        xiaoming        "]
**********************************************
changed: [cache]
TASK                                                            [command]
***************************************************************
changed: [cache]
TASK                                                            [command]
***************************************************************
changed: [cache]
PLAY                                                              RECAP
*******************************************************************
cache                         : ok=4     changed=3     unreachable=0
failed=0
```

步骤二：变量过滤器，创建一个用户，设置密码

```
[root@ansible ansible]# vim user1.yml
---
- hosts: cache
  remote_user: root
  tasks:
   - user:
        name: lisi
        group: root
        password: "{{'123456' | password_hash('sha512')}}"
   - shell: chage -d 0 lisi
[root@ansible ansible]# ansible-playbook user1.yml
PLAY                                                              [cache]
*****************************************************************
TASK                          [Gathering                          Facts]
*******************************************************
ok: [cache]
TASK                                                               [user]
******************************************************************
```

```
    changed: [cache]
    TASK                                                    [command]
*****************************************************************
    changed: [cache]
    PLAY                                                        RECAP
*********************************************************************
    cache                      : ok=3     changed=2     unreachable=0
failed=0
```

步骤三：定义一个变量创建用户

```
    [root@ansible ansible]# vim user2.yml
    ---
    - hosts: cache
      remote_user: root
      vars:
        user: zhangs
      tasks:
        - user:
            name: "{{user}}"
            group: root
            password: "{{'123456' | password_hash('sha512')}}"
        - shell: chage -d 0 "{{user}}"
    [root@ansible ansible]# ansible-playbook user2.yml
    PLAY                                                        [cache]
*******************************************************************
    TASK                        [Gathering                       Facts]
*********************************************************
    ok: [cache]
    TASK                                                         [user]
********************************************************************
    changed: [cache]
    TASK                                                    [command]
*****************************************************************
    changed: [cache]
    PLAY                                                        RECAP
*********************************************************************
    cache                      : ok=3     changed=2     unreachable=0
failed=0
```

3 案例 3：handlers 练习
3.1 问题

本案例要求：

安装 Apache 软件
配置文件，重新载入配置文件让服务生效
使用 handlers 来实现

3.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：error

playbook 从上往下顺序执行，若报错，后面的命令不会在执行，若想解决有两种方法：

1）当返回值为假时，显示 true： - shell: setenforce 0 || true

```
[root@ansible ansible]# vim user5.yml
---
- hosts: cache
  remote_user: root
  vars:
    user: bb
  tasks:
   - shell: setenforce 0 || true
   - user:
       name: "{{user}}"
       group: root
       password: "{{'123456' | password_hash('sha512')}}"
   - shell: chage -d 0 "{{user}}"
[root@ansible ansible]# ansible-playbook user5.yml
PLAY                                                                    [cache]
*****************************************************************
TASK                            [Gathering                               Facts]
*******************************************************
ok: [cache]
TASK                                                                  [command]
***************************************************************
changed: [cache]
TASK                                                                     [user]
******************************************************************
changed: [cache]
TASK                                                                  [command]
***************************************************************
changed: [cache]
PLAY                                                                      RECAP
```

```
******************************************************************
    cache                            : ok=4      changed=3      unreachable=0
failed=0
```

2、忽略：ignoring_errors: True(推荐使用这个，会有报错信息，告诉你错误忽略，继续执行下面的命令)

```
    [root@ansible ansible]# vim user6.yml
    ---
    - hosts: cache
      remote_user: root
      vars:
        user: bb
      tasks:
       - shell: setenforce 0
         ignore_errors: True
       - user:
           name: "{{user}}"
           group: root
           password: "{{'123456' | password_hash('sha512')}}"
       - shell: chage -d 0 "{{user}}"
    [root@ansible ansible]# ansible-playbook user6.yml
    PLAY                                                                  [cache]
****************************************************************
    TASK                        [Gathering                                 Facts]
******************************************************
    ok: [cache]
    TASK                                                              [command]
**************************************************************
    fatal: [cache]: FAILED! => {"changed": true, "cmd": "setenforce 0", "delta":
"0:00:00.004198", "end": "2018-09-07 11:08:14.936959", "msg": "non-zero
return code", "rc": 1, "start": "2018-09-07 11:08:14.932761", "stderr":
"setenforce: SELinux is disabled", "stderr_lines": ["setenforce: SELinux is
disabled"], "stdout": "", "stdout_lines": []}
    ...ignoring
    TASK                                                                   [user]
*****************************************************************
    changed: [cache]
    TASK                                                              [command]
**************************************************************
    changed: [cache]
    PLAY                                                                  RECAP
******************************************************************
    cache                            : ok=4      changed=3      unreachable=0
```

failed=0

步骤二： handlers

关注的资源发生变化时采取的操作

1) 使用 handlers 来配置文件，重新载入配置文件让服务生效

```
[root@ansible ansible]# vim adhttp.yml
---
- hosts: cache
  remote_user: root
  tasks:
    - copy:
        src: /root/httpd.conf
        dest:   /etc/httpd/conf/httpd.conf
        owner: root
        group: root
        mode: 0644
      notify:
        - restart httpd
  handlers:
     - name: restart httpd
       service: name=httpd state=restarted
[root@ansible ansible]# ansible-playbook adhttp.yml
PLAY                                                                    [cache]
*****************************************************************
TASK                          [Gathering                                 Facts]
*******************************************************
ok: [cache]
TASK                                                                     [copy]
******************************************************************
ok: [cache]
PLAY                                                                     RECAP
*******************************************************************
cache                          : ok=2     changed=0     unreachable=0
failed=0
[root@ansible ansible]# ssh cache apachectl -t
Syntax OK
[root@ansible ansible]# curl 192.168.1.56:8080
hello world
```

2）使用脚本调用变量更改服务

```
[root@ansible ansible]# vim adhttp2.yml
---
- hosts: cache
  remote_user: root
  vars:
    server: httpd
  tasks:
    - copy:
        src: /root/httpd.conf
        dest:   /etc/httpd/conf/httpd.conf
        owner: root
        group: root
        mode: 0644
      notify:
        - restart "{{server}}"
  handlers:
     - name: restart "{{server}}"
       service: name=httpd state=restarted
[root@ansible ansible]# ansible-playbook adhttp2.yml
PLAY                                                                    [cache]
*******************************************************************************************************
TASK                          [Gathering                                 Facts]
**************************************************************************************************
ok: [cache]
TASK                                                                     [copy]
********************************************************************************************************
ok: [cache]
PLAY                                                                    RECAP
*********************************************************************************************************
cache                         : ok=2      changed=0      unreachable=0
failed=0
[root@ansible ansible]#
```

4 案例 4：编写 playbook

4.1 问题

本案例要求：

把所有监听端口是 8080 的 Apache 服务全部停止

4.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：把监听端口是8080的Apache服务全部停止

```
[root@ansible ansible]# vim ad.yml
---
- hosts: cache
  remote_user: root
  tasks:
    - shell: netstat -atunlp   | awk   '{print $4}'| awk '-F:' '{print $2}'
      register: result
    - service:
        name: httpd
        state: stopped
[root@ansible ansible]# ansible-playbook ad.yml
PLAY                                                                  [cache]
***************************************************************************************************************
TASK                       [Gathering                                 Facts]
**********************************************************************************************************
ok: [cache]
TASK                                                                [command]
**************************************************************************************************************
changed: [cache]
TASK                                                                [service]
**************************************************************************************************************
changed: [cache]
PLAY                                                                  RECAP
*****************************************************************************************************************
cache                        : ok=3     changed=2     unreachable=0
failed=0
```

步骤二：when条件判断

1）当系统负载超过0.7时，则关掉httpd

```
[root@ansible ansible]# vim when.yml
---
```

```
- hosts: cache
  remote_user: root
  tasks:
    - shell: uptime | awk '{printf("%.2f",$(NF-2))}'
      register: result
    - service:
        name: httpd
        state: stopped
      when: result.stdout|float > 0.7
[root@ansible ansible]# ansible-playbook when.yml
PLAY                                                                     [cache]
*******************************************************************************
*****************************
TASK                          [Gathering                                Facts]
*******************************************************************************
*******************
ok: [cache]
TASK                                                                 [command]
*******************************************************************************
***************************
changed: [cache]
TASK                                                                 [service]
*******************************************************************************
***************************
changed: [cache]
PLAY                                                                     RECAP
*******************************************************************************
*******************************
cache                          : ok=3      changed=2      unreachable=0
failed=0
```

步骤三：with_items 标准循环

1）为不同用户定义不同组

```
[root@ansible ansible]# vim add.yml
---
- hosts: web2
  remote_user: root
  tasks:
    - user:
        name: "{{item.name}}"
        group: "{{item.group}}"
        password: "{{'123456'|password_hash('sha512')}}"
```

```
          with_items:
            - {name: "aa", group: "users"}
            - {name: "bb", group: "mail" }
            - {name: "cc", group: "wheel"}
            - {name: "dd", group: "root" }
    [root@ansible ansible]# ansible-playbook add.yml
    PLAY                                                                          [web2]
*******************************************************************************
******************************
    TASK                          [Gathering                                   Facts]
*******************************************************************************
*******************
    ok: [web2]
    TASK                                                                          [user]
*******************************************************************************
******************************
    changed: [web2] => (item={u'group': u'users', u'name': u'aa'})
    changed: [web2] => (item={u'group': u'mail', u'name': u'bb'})
    changed: [web2] => (item={u'group': u'wheel', u'name': u'cc'})
    changed: [web2] => (item={u'group': u'root', u'name': u'dd'})
    PLAY                                                                          RECAP
*******************************************************************************
*******************************
    web2                          : ok=2      changed=1      unreachable=0
failed=0
```

2）嵌套循环，循环添加多用户

```
    [root@ansible ansible]# vim add1.yml
    ---
    - hosts: web2
      remote_user: root
      vars:
        un: [a, b, c]
        id: [1, 2, 3]
      tasks:
        - name: add users
          shell: echo {{item}}
          with_nested:
            - "{{un}}"
            - "{{id}}"
     [root@ansible ansible]# ansible-playbook add1.yml
    PLAY                                                                          [web2]
*******************************************************************************
```

```
******************************
    TASK                          [Gathering                          Facts]
********************************************************************************
*******************
    ok: [web2]
    TASK                          [add                                users]
********************************************************************************
*************************
    changed: [web2] => (item=[u'a', 1])
    changed: [web2] => (item=[u'a', 2])
    changed: [web2] => (item=[u'a', 3])
    changed: [web2] => (item=[u'b', 1])
    changed: [web2] => (item=[u'b', 2])
    changed: [web2] => (item=[u'b', 3])
    changed: [web2] => (item=[u'c', 1])
    changed: [web2] => (item=[u'c', 2])
    changed: [web2] => (item=[u'c', 3])
    PLAY                                                                  RECAP
********************************************************************************
*******************************
    web2                          : ok=2      changed=1      unreachable=0
failed=0
```

步骤四：tags 给指定的任务定义一个调用标识

1）tags 样例

```
    [root@ansible ansible]# vim adhttp.yml
    ---
    - hosts: cache
      remote_user: root
      tasks:
        - copy:
            src: /root/httpd.conf
            dest:   /etc/httpd/conf/httpd.conf
            owner: root
            group: root
            mode: 0644
          tags: config_httpd
          notify:
            - restart httpd
      handlers:
         - name: restart httpd
           service: name=httpd state=restarted
```

2）调用方式

```
[root@ansible ansible]# ansible-playbook adhttp.yml   --tags=config_httpd
PLAY [cache] ****************************************************************
TASK [Gathering Facts] ******************************************************
ok: [cache]
TASK [copy] *****************************************************************
ok: [cache]
PLAY RECAP ******************************************************************
cache                          : ok=2    changed=0    unreachable=0    failed=0
```

3）include and roles

在编写 playbook 的时候随着项目越来越大，playbook 越来越复杂。可以把一些 play、task 或 handler 放到其他文件中，通过包含进来是一个不错的选择

roles 像是加强版的 include，它可以引入一个项目的文件和目录

一般所需的目录层级有

vars：变量层

tasks：任务层

handlers：触发条件

files：文件

template：模板

default：默认，优先级最低

```
...
tasks:
    - include: tasks/setup.yml
    - include: tasks/users.yml user=plj
//users.yml 中可以通过{{ user }}来使用这些变量
handlers:
    - include: handlers/handlers.yml
```

步骤五：debug 检测

```
[root@ansible ansible]# ansible-playbook  --syntax-check  http.yml  //检测语法
playbook: http.yml
[root@ansible ansible]# ansible-playbook  -C  http.yml   //测试运行
[root@ansible ansible]# ansible-playbook   http.yml  --list-tasks
//显示要执行的工作
playbook: http.yml
  play #1 (cache): cache    TAGS: []
    tasks:
      install one specific version of Apache    TAGS: []
      lineinfile    TAGS: []
      replace    TAGS: []
      service    TAGS: []
      copy    TAGS: []
[root@ansible ansible]# vim debug.yml
---
- hosts: cache
  remote_user: root
  tasks:
    - shell: uptime |awk '{printf("%f\n",$(NF-2))}'
      register: result
    - shell: touch /tmp/isreboot
      when: result.stdout|float > 0.5
    - name: Show debug info
      debug: var=result
[root@ansible ansible]# ansible-playbook debug.yml          //运行
PLAY                                                                    [cache]
****************************************************************************************************************
TASK                          [Gathering                                Facts]
*******************************************************************************************************
ok: [cache]
TASK                                                                [command]
**************************************************************************************************************
changed: [cache]
TASK                                                                [command]
**************************************************************************************************************
```

```
skipping: [cache]
TASK [Show debug info] *****************************************************************************************************
ok: [cache] => {
    "result": {
        "changed": true,
        "cmd": "uptime |awk '{printf(\"%f\\n\",$(NF-2))}'",
        "delta": "0:00:00.005905",
        "end": "2018-09-07 12:57:51.371013",
        "failed": false,
        "rc": 0,
        "start": "2018-09-07 12:57:51.365108",
        "stderr": "",
        "stderr_lines": [],
        "stdout": "0.000000",
        "stdout_lines": [
            "0.000000"
        ]
    }
}
PLAY RECAP ***************************************************************************************************************
cache                      : ok=3    changed=1    unreachable=0    failed=0
```

Top
NSD ARCHITECTURE DAY03

1 案例 1：ES 集群安装

1.1 问题

本案例要求：

准备 1 台虚拟机
部署 elasticsearch 第一个节点
访问 9200 端口查看是否安装成功

1.2 方案

1）ELK 是日志分析平台，不是一款软件,而是一整套解决方案,是三个软件产品的首字母缩写，ELK 分别代表：

Elasticsearch:负责日志检索和储存

Logstash:负责日志的收集和分析、处理

Kibana:负责日志的可视化

2) ELK 组件在海量日志系统的运维中,可用于解决分布式日志数据集中式查询和管理系统监控等，故障排查，安全信息和事件管理，报表功能

部署 Elasticsearch 分布式集群安装，Kibana 作为可视化平台，实时总结流量和数据的图表，Logstash 用来收集处理日志，如表-1 所示：

表-1

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：先准备一台虚拟机

1）更改主机名，配置 IP，搭建第三方 yum 源(之前已经搭建过几次,这里不再赘述)

```
[root@se1 ~]# echo se1 > /etc/hostname
[root@se1 ~]# vim /etc/sysconfig/network-scripts/ifcfg-eth0
# Generated by dracut initrd
DEVICE="eth0"
ONBOOT="yes"
IPV6INIT="no"
IPV4_FAILURE_FATAL="no"
NM_CONTROLLED="no"
TYPE="Ethernet"
BOOTPROTO="static"
IPADDR=192.168.1.61
PREFIX=24
GATEWAY=192.168.1.254
[root@se1 ~]# vim /etc/yum.repos.d/local.repo
```

```
[local_repo]
name=CentOS-$releasever - Base
baseurl="ftp://192.168.1.254/system"
enabled=1
gpgcheck=1
[elk]
name=elk
baseurl="ftp://192.168.1.254/elk"
enabled=1
gpgcheck=0
```

2）部署 elasticsearch 第一个节点

```
[root@se1 ~]# vim /etc/hosts
192.168.1.61 se1
192.168.1.62 se2
192.168.1.63 se3
192.168.1.64 se4
192.168.1.65 se5
[root@se1 ~]# yum -y install java-1.8.0-openjdk.x86_64
[root@se1 ~]# java -version
openjdk version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-b12)
OpenJDK 64-Bit Server VM (build 25.131-b12, mixed mode)
[root@se1 ~]# sestatus        //查看 selinux 状态
SELinux status:                         disabled
[root@se1 ~]# yum -y install elasticsearch
[root@se1 ~]# vim /etc/elasticsearch/elasticsearch.yml
17 cluster.name: myelk            //配置集群名字
23 node.name: se1            //当前主机名称
54 network.host: 0.0.0.0        // 0.0.0.0（监听所有地址）
68 discovery.zen.ping.unicast.hosts: ["se1", "se2","se3"]
//声明集群里的主机成员有谁，不需要全部写进去
[root@se1 ~]# systemctl restart elasticsearch
[root@se1 ~]# systemctl enable elasticsearch
[root@se1 ~]# ss -antup | grep 9200
tcp        LISTEN          0          50          :::9200                          :::*
users:(("java",pid=23231,fd=110))
```

3）访问 9200 端口查看是否安装成功，如图-1 所示：

图-1

2 案例 2：ES 集群安装配置

2.1 问题

本案例要求：

    一共安装 5 台虚拟机
    在所有机器中部署 ES
    启动服务查看验证集群状态

2.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：安装 elasticsearch 和 java-1.8.0-openjdk，同步配置文件

备注：在步骤一已经安装了一台 elasticsearch，这里只需再准备四台即可

1）更改对应的主机名、 ip 地址以及搭建 yum 源（以案例 1 为例子）

2）安装 elasticsearch 四台主机同样操作（以 se2 为例子）

```
    [root@se2 ~]# yum -y install java-1.8.0-openjdk.x86_64
    [root@se2 ~]# yum -y install elasticsearch
```

3）同步配置/etc/hosts 和/etc/elasticsearch/elasticsearch.yml, 修改 node.name 字段（以 se2 为例子）

```
    [root@se1 ~]# for i in {62..65} ; do scp /etc/hosts 192.168.1.$i:/etc/hosts; 
done
    [root@se1 ~]# for i in {62..65} ; do scp   \
    /etc/elasticsearch/elasticsearch.yml \
    192.168.1.$i:/etc/elasticsearch/elasticsearch.yml; done
    [root@se2 ~]# vim /etc/elasticsearch/elasticsearch.yml
    node.name: se2      //另外三台修改为对应 se3，se4，se5
    [root@se2 ~]# systemctl restart elasticsearch
    [root@se2 ~]# systemctl enable elasticsearch
```

4）访问测试，如图-2 所示：

可以访问 61-65 的任意一台主机， 集群的节点都是 5 台，若先启动的是 se4 或 se5，这两个会自动成为各自的集群，解决办法，先启动集群里的 se1 或 se2 或 se3 其中的一台，或者把 se4 和 se5 重启，se4 和 se5 会自动加进去

ES 集群验证：返回字段解析：

”status”：”green“ 集群状态：绿色为正常、黄色表示有问题但不是很严重、红色表

示严重故障

”number_of_nodes”： 5, 表示集群中节点的数量

图-2

3 案例 3：练习 curl 命令

3.1 问题

本案例要求：

- 练习使用 curl 命令
- 理解 GET POST
- 使用 curl 命令访问 ES 集群

3.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：curl 命令的使用

http 的请求方法：

常用方法 GET，POST，HEAD

其他方法 OPTIONS，PUT，DELETE，TRACE 和 CONNECT

ES 常用：

PUT --增

DELETE --删

POST --改

GET --查

系统命令 curl：

是一个利用 URL 规则在命令行下工作的文件传输工具,可以说是一款很强大的 http 命令行工具。它支持多种请求模式,自定义请求头等强大功能,是一款综合工具

curl 常用参数介绍：

-A 修改请求 agent

-X 设置请求方法

-i 显示返回头信息

1）索引的分片信息，如图-1 所示：

```
[root@room9pc01 ~]# curl -X GET http://192.168.1.61:9200/_cat
```

图-1

2）显示 health 的详细信息，如图-2 所示：

```
[root@room9pc01 ~]# curl -X GET http://192.168.1.62:9200/_cat/health?v
```

图-2

3）查看 nodes 的帮助，如图-3 所示：

```
[root@room9pc01 ~]# curl -X GET http://192.168.1.61:9200/_cat/nodes?help
```

图-3

4 案例 4：练习插件

4.1 问题

本案例要求：

在其中一台机器上部署插件
使用 bigdesk 查看集群状态
使用 head 创建 index
使用 kopf 查看数据

4.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：部署插件

插件装在哪一台机器上，只能在哪台机器上使用（这里安装在 se5 机器上面）

1）使用远程 uri 路径可以直接安装

```
[root@se5 ~]# cd /usr/share/elasticsearch/bin
[root@se5 bin]# ./plugin   install   \
ftp://192.168.1.254/elk/elasticsearch-head-master.zip              //安装 head 插件
[root@se5 bin]# ./plugin   install   \
ftp://192.168.1.254/elk/elasticsearch-kopf-master.zip              //安装 kopf 插件
[root@se5 bin]# [root@se5 bin]# ./plugin install   \
 ftp://192.168.1.254/elk/bigdesk-master.zip
//安装 bigdesk 插件
[root@se5 bin]# ./plugin   list              //查看安装的插件
Installed plugins in /usr/share/elasticsearch/plugins:
    - head
    - kopf
    - bigdesk
```

2）访问 head 插件，如图-4 所示：

```
[root@room9pc01 ~]#   firefox http://192.168.1.65:9200/_plugin/head
```

图-4

3）访问 kopf 插件，如图-5 所示：

```
[root@room9pc01 ~]#   http://192.168.1.65:9200/_plugin/kopf
```

图-5

4）访问 bigdesk 插件，如图-6 所示：

```
[root@room9pc01 ~]#   http://192.168.1.65:9200/_plugin/bigdesk
```

图-6

步骤二：使用 head 创建 index

```
[root@se5 bin]# curl -X PUT "http://192.168.1.65:9200/index" -d '
> {
>       "settings":{
>       "index":{
>       "number_of_shards":5,            //分片数
>       "number_of_replicas":1            //副本数
>      }
>    }
> }'
```

```
{"acknowledged":true}
```

步骤三：使用 kopf 查看数据，如图-7 所示：

图-7

5 案例 5：插入，增加，删除查询数据

5.1 问题

本案例要求：

- 使用 curl 命令连接使用 ES 数据库
- 使用 PUT 方法增加数据
- 使用 POST 修改数据
- 使用 GET 查询数据
- 使用 DELETE 删除数据

5.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：增加数据

```
[root@se5 ~]# locale
[root@se5 ~]# LANG=en_US.UTF-8   //设置编码
[root@se5 ~]# curl -X PUT "http://192.168.1.65:9200/taindex/teacher/1" -d
'{
"职业":"诗人",
"名字":"李白",
"称号":"诗仙",
"年代":"唐"
}'
```

```
{"_index":"taindex","_type":"teacher","_id":"1","_version":2,"_shards":{"total":
2,"successful":2,"failed":0},"created":false}
```

步骤二：修改数据

```
[root@se5 ~]# curl -X PUT "http://192.168.1.65:9200/taindex/teacher/1" -d
'{
 "doc":{
 "年代": "唐代"
 }
 }'
```

```
{"_index":"taindex","_type":"teacher","_id":"1","_version":3,"_shards":{"total":2,"successful":2,"failed":0},"created":false}
```

步骤三：查询数据

```
[root@se5 ~]# curl -X GET "http://192.168.1.65:9200/taindex/teacher/3?pretty"
{
  "_index" : "taindex",
  "_type" : "teacher",
  "_id" : "3",
  "found" : false
}
```

步骤四：删除数据

```
[root@se5 ~]# curl -X DELETE "http://192.168.1.65:9200/taindex/teacher/3?pretty"
{
  "found" : false,
  "_index" : "taindex",
  "_type" : "teacher",
  "_id" : "3",
  "_version" : 1,
  "_shards" : {
    "total" : 2,
    "successful" : 2,
    "failed" : 0
  }
}
```

步骤五：删除索引

```
[root@se5 bin]# curl -X DELETE http://192.168.1.65:9200/taindex/
//删除索引
{"acknowledged":true}
[root@se5 bin]# curl -X DELETE http://192.168.1.65:9200/*          //删除所有索引
{"acknowledged":true}
```

6 案例 6：安装 Kibana

6.1 问题

本案例要求：

安装 Kibana
配置启动服务查看 5601 端口是否正常
通过 web 页面访问 Kibana

6.2 步骤

实现此案例需要按照如下步骤进行

步骤一：安装 kibana

1）在另一台主机，配置 ip 为 192.168.1.66，配置 yum 源，更改主机名

2）安装 kibana

```
[root@kibana ~]# yum -y install kibana
[root@kibana ~]# rpm -qc kibana
/opt/kibana/config/kibana.yml
[root@kibana ~]# vim /opt/kibana/config/kibana.yml
  2 server.port: 5601
//若把端口改为 80，可以成功启动 kibana，但 ss 时没有端口，没有监听 80 端口，服务里面写死了，不能用 80 端口，只能是 5601 这个端口
  5 server.host: "0.0.0.0"          //服务器监听地址
15 elasticsearch.url: http://192.168.1.61:9200
//声明地址，从哪里查，集群里面随便选一个
23 kibana.index: ".kibana"     //kibana 自己创建的索引
26 kibana.defaultAppId: "discover"     //打开 kibana 页面时，默认打开的页面 discover
53 elasticsearch.pingTimeout: 1500     //ping 检测超时时间
57 elasticsearch.requestTimeout: 30000     //请求超时
64 elasticsearch.startupTimeout: 5000     //启动超时
[root@kibana ~]# systemctl restart kibana
[root@kibana ~]# systemctl enable  kibana
Created symlink from /etc/systemd/system/multi-user.target.wants/kibana.service to /usr/lib/systemd/system/kibana.service.
[root@kibana ~]# ss -antup | grep 5601  //查看监听端口
```

3）浏览器访问 kibana，如图-8 所示：

```
[root@kibana ~]# firefox 192.168.1.66:5601
```

图-8

4）点击 Status，查看是否安装成功，全部是绿色的对钩,说明安装成功，如图-9 所示：

图-9

5）用 head 插件访问会有.kibana 的索引信息，如图-10 所示：

```
[root@se5 ~]# firefox http://192.168.1.65:9200/_plugin/head/
```

图-10

Top

NSD ARCHITECTURE DAY04

- 案例 1：导入数据
- 案例 2：综合练习

1 案例 1：导入数据

1.1 问题

本案例要求批量导入数据：

- 批量导入数据并查看

1.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：导入数据

使用 POST 方式批量导入数据，数据格式为 json，url 编码使用 data-binary 导入含有 index 配置的 json 文件

```
    [root@room9pc01 ~]# scp /var/ftp/elk/*.gz 192.168.1.66:/root/
    [root@kibana ~]# gzip   -d logs.jsonl.gz
    [root@kibana ~]#   gzip   -d accounts.json.gz
    [root@kibana ~]# gzip   -d shakespeare.json.gz
    [root@kibana ~]# curl -X POST "http://192.168.1.61:9200/_bulk" \
    --data-binary @shakespeare.json
    [root@kibana ~]# curl -X POST "http://192.168.1.61:9200/xixi/haha/_bulk"
\
     --data-binary @accounts.json
    //索引是 xixi，类型是 haha，必须导入索引和类型，没有索引，要加上
    [root@kibana ~]# curl -X POST "http://192.168.1.61:9200/_bulk"   \
--data-binary @logs.jsonl
```

2）使用 GET 查询结果

```
[root@kibana ~]# curl -XGET 'http://192.168.1.61:9200/_mget?pretty' -d '{
 "docs":[
      {
         "_index":"shakespeare",
         "_type:":"act",
         "_id":0
},
{
         "_index":"shakespeare",
         "_type:":"line",
         "_id":0
},
{
         "_index":"xixi",
         "_type:":"haha",
         "_id":25
}
]
}'
{           //查询的结果
  "docs" : [ {
    "_index" : "shakespeare",
    "_type" : "act",
    "_id" : "0",
    "_version" : 1,
    "found" : true,
    "_source" : {
      "line_id" : 1,
      "play_name" : "Henry IV",
      "speech_number" : "",
      "line_number" : "",
      "speaker" : "",
      "text_entry" : "ACT I"
    }
  }, {
    "_index" : "shakespeare",
    "_type" : "act",
    "_id" : "0",
    "_version" : 1,
    "found" : true,
        "_source" : {
          "line_id" : 1,
          "play_name" : "Henry IV",
          "speech_number" : "",
          "line_number" : "",
          "speaker" : "",
          "text_entry" : "ACT I"
        }
      }, {
        "_index" : "xixi",
        "_type" : "haha",
        "_id" : "25",
        "_version" : 1,
        "found" : true,
        "_source" : {
          "account_number" : 25,
          "balance" : 40540,
          "firstname" : "Virginia",
          "lastname" : "Ayala",
          "age" : 39,
          "gender" : "F",
          "address" : "171 Putnam Avenue",
          "employer" : "Filodyne",
          "email" : "virginiaayala@filodyne.com",
          "city" : "Nicholson",
          "state" : "PA"
        }
      } ]
    }
```

步骤二：使用 kibana 查看数据是否导入成功

1）数据导入以后查看 logs 是否导入成功，如图-1 所示：

```
    [root@se5 ~]# firefox http://192.168.1.65:9200/_plugin/head/
```

图-1

2）kibana 导入数据，如图-2 所示：

```
    [root@kibana ~]# firefox   http://192.168.1.66:5601
```

图-2

3）成功创建会有 logstash-*，如图-3 所示：

/

图-3

4）导入成功之后选择 Discover，如图-4 所示：

图-4

注意： 这里没有数据的原因是导入日志的时间段不对，默认配置是最近 15 分钟，在这可以修改一下时间来显示

5）kibana 修改时间，选择 Lsat 15 miuntes，如图-5 所示：

图-5

6）选择 Absolute，如图-6 所示：

图-6

7）选择时间 2015-5-15 到 2015-5-22，如图-7 所示：

图-7

8）查看结果，如图-8 所示：

图-8

9）除了柱状图，Kibana 还支持很多种展示方式 ，如图-9 所示：

图-9

10）做一个饼图，选择 Pie chart，如图-10 所示：

图-10

11）选择 from a new serach，如图-11 所示：

图-11

12）选择 Spilt Slices，如图-12 所示：

图-12

13）选择 Trems,Memary(也可以选择其他的，这个不固定)，如图-13 所示：

图-13

14）结果，如图-14 所示：

图-14

15）保存后可以在 Dashboard 查看，如图-15 所示：

图-15

2 案例 2：综合练习

2.1 问题

本案例要求：

- 练习插件
- 安装一台 Apache 服务并配置
- 使用 filebeat 收集 Apache 服务器的日志
- 使用 grok 处理 filebeat 发送过来的日志
- 存入 elasticsearch

2.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：安装 logstash

1）配置主机名，ip 和 yum 源，配置/etc/hosts（请把 se1-se5 和 kibana 主机配置和 logstash 一样的/etc/hosts）

```
[root@logstash ~]# vim /etc/hosts
192.168.1.61 se1
192.168.1.62 se2
192.168.1.63 se3
192.168.1.64 se4
192.168.1.65 se5
192.168.1.66 kibana
192.168.1.67 logstash
```

2）安装 java-1.8.0-openjdk 和 logstash

```
[root@logstash ~]#   yum -y install java-1.8.0-openjdk
```

```
[root@logstash ~]# yum -y install logstash
[root@logstash ~]#   java -version
openjdk version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-b12)
OpenJDK 64-Bit Server VM (build 25.131-b12, mixed mode)
[root@logstash ~]# touch /etc/logstash/logstash.conf
[root@logstash ~]#   /opt/logstash/bin/logstash   --version
logstash 2.3.4
[root@logstash ~]# /opt/logstash/bin/logstash-plugin   list     //查看插件
...
logstash-input-stdin      //标准输入插件
logstash-output-stdout      //标准输出插件
...
[root@logstash ~]# vim /etc/logstash/logstash.conf
input{
    stdin{
   }
}
filter{
}
output{
    stdout{
   }
}
[root@logstash ~]# /opt/logstash/bin/logstash -f /etc/logstash/logstash.conf
//启动并测试
Settings: Default pipeline workers: 2
Pipeline main started
aa          //logstash 配置从标准输入读取输入源,然后从标准输出输出到屏幕
2018-09-15T06:19:28.724Z logstash aa
```

备注：若不会写配置文件可以找帮助，插件文档的位置：

https://github.com/logstash-plugins

3）codec 类插件

```
[root@logstash ~]# vim /etc/logstash/logstash.conf
input{
    stdin{
    codec => "json"          //输入设置为编码 json
   }
}
```

```
    filter{
    }
    output{
        stdout{
        codec => "rubydebug"            //输出设置为 rubydebug
       }
    }
    [root@logstash ~]# /opt/logstash/bin/logstash -f /etc/logstash/logstash.conf
    Settings: Default pipeline workers: 2
    Pipeline main started
    {"a":1}
    {
                 "a" => 1,
            "@version" => "1",
        "@timestamp" => "2018-09-15T06:34:14.538Z",
              "host" => "logstash"
    }
```

4）file 模块插件

```
    [root@logstash ~]# vim /etc/logstash/logstash.conf
    input{
      file {
        path           => [ "/tmp/a.log", "/var/tmp/b.log" ]
       sincedb_path    => "/var/lib/logstash/sincedb"     //记录读取文件的位置
       start_position => "beginning"                  //配置第一次读取文件从什么地方开始
       type           => "testlog"                    //类型名称
      }
    }
    filter{
    }
    output{
        stdout{
        codec => "rubydebug"
    }
    }
    [root@logstash ~]# touch /tmp/a.log
    [root@logstash ~]# touch /var/tmp/b.log
    [root@logstash ~]# /opt/logstash/bin/logstash -f /etc/logstash/logstash.conf
```

另开一个终端：写入数据

```
[root@logstash ~]#   echo a1 > /tmp/a.log
[root@logstash ~]#   echo b1 > /var/tmp/b.log
```

之前终端查看：

```
[root@logstash ~]#   /opt/logstash/bin/logstash -f /etc/logstash/logstash.conf
Settings: Default pipeline workers: 2
Pipeline main started
{
       "message" => "a1",
      "@version" => "1",
    "@timestamp" => "2018-09-15T06:44:30.671Z",
          "path" => "/tmp/a.log",
          "host" => "logstash",
          "type" => "testlog"
}
{
       "message" => "b1",
      "@version" => "1",
    "@timestamp" => "2018-09-15T06:45:04.725Z",
          "path" => "/var/tmp/b.log",
          "host" => "logstash",
          "type" => "testlog"
}
```

5）tcp、udp 模块插件

```
[root@logstash ~]#   vim /etc/logstash/logstash.conf
input{
  file {
    path           => [ "/tmp/a.log", "/var/tmp/b.log" ]
   sincedb_path    => "/var/lib/logstash/sincedb"
   start_position => "beginning"
   type            => "testlog"
  }
  tcp {
     host => "0.0.0.0"
     port => "8888"
     type => "tcplog"
}
   udp {
          host => "0.0.0.0"
          port => "9999"
          type => "udplog"
    }
    }
    filter{
    }
    output{
        stdout{
        codec => "rubydebug"
    }
    }
    [root@logstash          ~]#                    /opt/logstash/bin/logstash          -f
/etc/logstash/logstash.conf
    //启动
```

另开一个终端查看，可以看到端口

```
    [root@logstash tmp]#   netstat -antup | grep 8888
    tcp6           0          0 :::8888                        :::*                          LISTEN
22191/java
    [root@logstash tmp]# netstat -antup | grep 9999
    udp6                   0                 0  :::9999                                          :::*
22191/java
```

在另一台主机上写一个脚本，发送数据，使启动的 logstash 可以接收到数据

```
    [root@se5 ~]# vim tcp.sh
    function sendmsg(){
      if [[ "$1" == "tcp" ]];then
              exec 9<>/dev/tcp/192.168.1.67/8888
       else
              exec 9<>/dev/udp/192.168.1.67/9999
       fi
          echo "$2" >&9
          exec 9<&-
    }
    [root@se5 ~]# . tcp.sh            //重新载入一下
    [root@se5 ~]# sendmsg udp "is tcp test"
    [root@se5 ~]# sendmsg udp "is tcp ss"
```

logstash 主机查看结果

```
    [root@logstash          ~]#                    /opt/logstash/bin/logstash          -f
```

```
/etc/logstash/logstash.conf
    Settings: Default pipeline workers: 2
    Pipeline main started
    {
           "message" => "is tcp test\n",
          "@version" => "1",
        "@timestamp" => "2018-09-15T07:45:00.638Z",
              "type" => "udplog",
              "host" => "192.168.1.65"
    }
    {
           "message" => "is tcp ss\n",
          "@version" => "1",
        "@timestamp" => "2018-09-15T07:45:08.897Z",
              "type" => "udplog",
              "host" => "192.168.1.65"
    }
```

6）syslog 插件练习

```
    [root@logstash ~]#   systemctl   list-unit-files | grep syslog
    rsyslog.service                                  enabled
    syslog.socket                                    static
    [root@logstash ~]#   vim /etc/logstash/logstash.conf
       start_position => "beginning"
       type           => "testlog"
      }
      tcp {
         host => "0.0.0.0"
         port => "8888"
         type => "tcplog"
    }
       udp {
         host => "0.0.0.0"
         port => "9999"
         type => "udplog"
    }
      syslog {
         port => "514"
         type => "syslog"
      }
    }
    filter{
    }
    output{
        stdout{
        codec => "rubydebug"
    }
    }
```

另一个终端查看是否检测到 514

```
    [root@logstash ~]#   netstat -antup | grep 514
    tcp6           0          0 :::514                          :::*                            LISTEN
22728/java
    udp6                   0                0   :::514                                                    :::*
22728/java
```

另一台主机上面操作,本地写的日志本地可以查看

```
    [root@se5 ~]# vim /etc/rsyslog.conf
    local0.info                                                  /var/log/mylog    //自己添加这
一行
    [root@se5 ~]# systemctl restart rsyslog       //重启 rsyslog
    [root@se5 ~]#   ll /var/log/mylog             //提示没有那个文件或目录
    ls: cannot access /var/log/mylog: No such file or directory
    [root@se5 ~]# logger -p local0.info -t nsd "elk"             //写日志
    [root@se5 ~]#   ll /var/log/mylog             //再次查看，有文件
    -rw------- 1 root root 29 Sep 15 16:23 /var/log/mylog
    [root@se5 ~]# tail   /var/log/mylog     //可以查看到写的日志
    Sep 15 16:23:25 se5 nsd: elk
    [root@se5 ~]# tail   /var/log/messages
    //可以查看到写的日志，因为配置文件里有写以.info 结尾的可以收到
    ...
    Sep 15 16:23:25 se5 nsd: elk
```

把本地的日志发送给远程 1.67

```
    [root@se5 ~]# vim /etc/rsyslog.conf
    local0.info                              @192.168.1.67:514
    //写一个@或两个@@都可以，一个@代表 udp，两个@@代表 tcp
    [root@se5 ~]# systemctl restart rsyslog
    [root@se5 ~]# logger   -p local0.info -t nds "001 elk"
    [root@logstash              bin]#              /opt/logstash/bin/logstash              -f
/etc/logstash/logstash.conf
    //检测到写的日志
    {
                   "message" => "001 elk",
              "@version" => "1",
            "@timestamp" => "2018-09-05T09:15:47.000Z",
                  "type" => "syslog",
                  "host" => "192.168.1.65",
              "priority" => 134,
             "timestamp" => "Jun  5 17:15:47",
             "logsource" => "kibana",
               "program" => "nds1801",
              "severity" => 6,
              "facility" => 16,
        "facility_label" => "local0",
        "severity_label" => "Informational"
    }
```

rsyslog.conf 配置向远程发送数据，远程登陆 1.65 的时侯，把登陆日志的信息（/var/log/secure）转发给 logstash 即 1.67 这台机器

```
    [root@se5 ~]#  vim /etc/rsyslog.conf
    57                                                              authpriv.*
@@192.168.1.67:514
    //57 行的/var/log/secure 改为@@192.168.1.67:514
    [root@se5 ~]# systemctl restart rsyslog
    [root@logstash          ~]#          /opt/logstash/bin/logstash          -f
/etc/logstash/logstash.conf
    //找一台主机登录 1.65，logstash 主机会有数据
    Settings: Default pipeline workers: 2
    Pipeline main started
    {
               "message" => "Accepted password for root from 192.168.1.254
port 33780 ssh2\n",
              "@version" => "1",
            "@timestamp" => "2018-09-15T08:40:57.000Z",
                  "type" => "syslog",
                  "host" => "192.168.1.65",
              "priority" => 86,
             "timestamp" => "Sep 15 16:40:57",
             "logsource" => "se5",
               "program" => "sshd",
                   "pid" => "26133",
              "severity" => 6,
              "facility" => 10,
        "facility_label" => "security/authorization",
        "severity_label" => "Informational"
    }
    {
                  "message"  =>  "pam_unix(sshd:session):  session  opened  for
user root by (uid=0)\n",
                 "@version" => "1",
               "@timestamp" => "2018-09-15T08:40:57.000Z",
                     "type" => "syslog",
                     "host" => "192.168.1.65",
                 "priority" => 86,
                "timestamp" => "Sep 15 16:40:57",
                "logsource" => "se5",
                  "program" => "sshd",
                      "pid" => "26133",
                 "severity" => 6,
                 "facility" => 10,
           "facility_label" => "security/authorization",
           "severity_label" => "Informational"
```

7）filter grok 插件

grok 插件：

解析各种非结构化的日志数据插件

grok 使用正则表达式把飞结构化的数据结构化

在分组匹配，正则表达式需要根据具体数据结构编写

虽然编写困难，但适用性极广

```
    [root@logstash ~]#   vim /etc/logstash/logstash.conf
    input{
            stdin{ codec => "json" }
      file {
        path              => [ "/tmp/a.log", "/var/tmp/b.log" ]
       sincedb_path     => "/var/lib/logstash/sincedb"
       start_position => "beginning"
       type               => "testlog"
      }
      tcp {
         host => "0.0.0.0"
         port => "8888"
         type => "tcplog"
    }
       udp {
          host => "0.0.0.0"
          port => "9999"
          type => "udplog"
    }
       syslog {
          port => "514"
          type => "syslog"
       }
    }
    filter{
      grok{
             match => ["message", "(?<key>reg)"]
       }
    }
    output{
        stdout{
        codec => "rubydebug"
    }
    }
    [root@se5 ~]# yum -y install httpd
    [root@se5 ~]# systemctl restart httpd
    [root@se5 ~]# vim /var/log/httpd/access_log
    192.168.1.254 - - [15/Sep/2018:18:25:46 +0800] "GET / HTTP/1.1" 403
4897 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:52.0) Gecko/20100101
Firefox/52.0"
```

复制/var/log/httpd/access_log 的日志到 logstash 下的/tmp/a.log

```
    [root@logstash ~]# vim /tmp/a.log
    192.168.1.254 - - [15/Sep/2018:18:25:46 +0800] "GET / HTTP/1.1" 403
4897 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:52.0) Gecko/20100101
Firefox/52.0"
    [root@logstash ~]# /opt/logstash/bin/logstash -f
/etc/logstash/logstash.conf
    //出现 message 的日志，但是没有解析是什么意思
    Settings: Default pipeline workers: 2
    Pipeline main started
    {
           "message" => ".168.1.254 - - [15/Sep/2018:18:25:46 +0800] \"GET
/ HTTP/1.1\" 403 4897 \"-\" \"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:52.0)
Gecko/20100101 Firefox/52.0\"",
          "@version" => "1",
        "@timestamp" => "2018-09-15T10:26:51.335Z",
              "path" => "/tmp/a.log",
              "host" => "logstash",
              "type" => "testlog",
              "tags" => [
            [0] "_grokparsefailure"
        ]
    }
```

若要解决没有解析的问题，同样的方法把日志复制到/tmp/a.log，logstash.conf 配置文件里面修改 grok

查找正则宏路径

```
    [root@logstash ~]# cd   /opt/logstash/vendor/bundle/ \
    jruby/1.9/gems/logstash-patterns-core-2.0.5/patterns/
    [root@logstash ~]# vim grok-patterns   //查找 COMBINEDAPACHELOG

COMBINEDAPACHELOG %{COMMONAPACHELOG} %{QS:referrer} %{QS:agent}
    [root@logstash ~]#   vim /etc/logstash/logstash.conf
    ...
    filter{
       grok{
            match => ["message", "%{COMBINEDAPACHELOG}"]
      }
    }
    ...
```

解析出的结果

```
     [root@logstash ~]# /opt/logstash/bin/logstash -f /etc/logstash/logstash.conf
    Settings: Default pipeline workers: 2
    Pipeline main started
    {
            "message" => "192.168.1.254 - - [15/Sep/2018:18:25:46 +0800] \"GET /noindex/css/open-sans.css HTTP/1.1\" 200 5081 \"http://192.168.1.65/\" \"Mozilla/5.0 (Windows NT 6.1; WOW64; rv:52.0) Gecko/20100101 Firefox/52.0\"",
           "@version" => "1",
         "@timestamp" => "2018-09-15T10:55:57.743Z",
               "path" => "/tmp/a.log",
               "host" => "logstash",
               "type" => "testlog",
           "clientip" => "192.168.1.254",
              "ident" => "-",
               "auth" => "-",
          "timestamp" => "15/Sep/2018:18:25:46 +0800",
               "verb" => "GET",
            "request" => "/noindex/css/open-sans.css",
        "httpversion" => "1.1",
           "response" => "200",
              "bytes" => "5081",
           "referrer" => "\"http://192.168.1.65/\"",
              "agent"  =>  "\"Mozilla/5.0  (Windows  NT  6.1;  WOW64;  rv:52.0)
Gecko/20100101 Firefox/52.0\""
    }
```

步骤二：安装 Apache 服务，用 filebeat 收集 Apache 服务器的日志，存入 elasticsearch

1）在之前安装了 Apache 的主机上面安装 filebeat

```
[root@se5 ~]#   yum -y install filebeat
[root@se5 ~]#   vim/etc/filebeat/filebeat.yml
paths:
    - /var/log/httpd/access_log     //日志的路径，短横线加空格代表 yml 格式
document_type: apachelog      //文档类型
elasticsearch:            //加上注释
hosts: ["localhost:9200"]                       //加上注释
logstash:                             //去掉注释
hosts: ["192.168.1.67:5044"]        //去掉注释,logstash 那台主机的 ip
[root@se5 ~]# systemctl start filebeat
[root@logstash ~]#   vim /etc/logstash/logstash.conf
input{
        stdin{ codec => "json" }
        beats{
            port => 5044
}
  file {
    path              => [ "/tmp/a.log", "/var/tmp/b.log" ]
   sincedb_path     => "/dev/null"
   start_position => "beginning"
   type               => "testlog"
  }
  tcp {
     host => "0.0.0.0"
     port => "8888"
     type => "tcplog"
}
```

```
   udp {
     host => "0.0.0.0"
     port => "9999"
     type => "udplog"
}
  syslog {
     port => "514"
     type => "syslog"
  }
}
filter{
if [type] == "apachelog"{
   grok{
        match => ["message", "%{COMBINEDAPACHELOG}"]
  }}
}
output{
      stdout{ codec => "rubydebug" }
      if [type] == "filelog"{
      elasticsearch {
          hosts => ["192.168.1.61:9200", "192.168.1.62:9200"]
          index => "filelog"
          flush_size => 2000
          idle_flush_time => 10
      }}
}
 [root@logstash logstash]#   /opt/logstash/bin/logstash   \
-f   /etc/logstash/logstash.conf
```

打开另一终端查看 5044 是否成功启动

```
[root@logstash ~]#   netstat -antup | grep 5044
tcp6          0         0 :::5044                        :::*                          LISTEN
23776/java
[root@se5 ~]#   firefox 192.168.1.65    //ip 为安装 filebeat 的那台机器
```

回到原来的终端，有数据

2）修改 logstash.conf 文件

```
[root@logstash logstash]# vim logstash.conf
...
output{
      stdout{ codec => "rubydebug" }
```

```
        if [type] == "apachelog"{
        elasticsearch {
            hosts => ["192.168.1.61:9200", "192.168.1.62:9200"]
            index => "apachelog"
            flush_size => 2000
            idle_flush_time => 10
        }}
    }
```

浏览器访问 Elasticsearch，有 apachelog，如图-16 所示：

图-16

Top

NSD ARCHITECTURE DAY05

- 案例 1：安装 Hadoop
- 案例 2：安装配置 Hadoop

1 案例 1：安装 Hadoop

1.1 问题

本案例要求安装单机模式 Hadoop：

- 单机模式安装 Hadoop
- 安装 JAVA 环境
- 设置环境变量，启动运行

1.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：环境准备

1）配置主机名为 nn01，ip 为 192.168.1.21，配置 yum 源（系统源）

备注：由于在之前的案例中这些都已经做过，这里不再重复，不会的学员可以参考之前的案例

2）安装 java 环境

```
[root@nn01 ~]# yum -y install java-1.8.0-openjdk-devel
[root@nn01 ~]# java -version
```

```
openjdk version "1.8.0_131"
OpenJDK Runtime Environment (build 1.8.0_131-b12)
OpenJDK 64-Bit Server VM (build 25.131-b12, mixed mode)
[root@nn01 ~]# jps
1235 Jps
```

3）安装 hadoop

```
[root@nn01 ~]# tar -xf hadoop-2.7.6.tar.gz
[root@nn01 ~]#   mv hadoop-2.7.6 /usr/local/hadoop
[root@nn01 ~]# cd /usr/local/hadoop/
[root@nn01 hadoop]# ls
bin   include   libexec         NOTICE.txt   sbin
etc   lib         LICENSE.txt   README.txt   share
[root@nn01 hadoop]# ./bin/hadoop     //报错，JAVA_HOME 没有找到
Error: JAVA_HOME is not set and could not be found.
[root@nn01 hadoop]#
```

4）解决报错问题

```
[root@nn01 hadoop]# rpm -ql   java-1.8.0-openjdk
[root@nn01 hadoop]# cd ./etc/hadoop/
[root@nn01 hadoop]# vim hadoop-env.sh
25 export \

JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.131-11.b12.el7.x86_64/jre"
33 export HADOOP_CONF_DIR="/usr/local/hadoop/etc/hadoop"
[root@nn01 ~]# cd /usr/local/hadoop/
[root@nn01 hadoop]# ./bin/hadoop
Usage: hadoop [--config confdir] [COMMAND | CLASSNAME]
  CLASSNAME                  run the class named CLASSNAME
 or
  where COMMAND is one of:
  fs                          run a generic filesystem user client
  version                     print the version
  jar <jar>                   run a jar file
                              note: please use "yarn jar" to launch
                                    YARN applications, not this command.
  checknative  [-a|-h]    check  native  hadoop  and  compression  libraries
availability
  distcp <srcurl> <desturl> copy file or directories recursively
  archive -archiveName NAME -p <parent path> <src>* <dest> create a
hadoop archive
  classpath                   prints the class path needed to get the
```

```
    credential              interact with credential providers
                            Hadoop jar and the required libraries
    daemonlog                 get/set the log level for each daemon
    trace                   view and modify Hadoop tracing settings
Most commands print help when invoked w/o parameters.
[root@nn01 hadoop]# mkdir /usr/local/hadoop/aa
[root@nn01 hadoop]# ls
bin   etc   include   lib   libexec   LICENSE.txt   NOTICE.txt   aa
README.txt  sbin  share
[root@nn01 hadoop]# cp *.txt /usr/local/hadoop/aa
[root@nn01 hadoop]# ./bin/hadoop jar  \
 share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.6.jar
wordcount aa bb          //wordcount 为参数 统计 aa 这个文件夹，存到 bb 这个文件里面（这个文件不能存在，要是存在会报错，是为了防止数据覆盖）
[root@nn01 hadoop]#  cat   bb/part-r-00000    //查看
```

2 案例 2：安装配置 Hadoop

2.1 问题

本案例要求：

- 另备三台虚拟机，安装 Hadoop
- 使所有节点能够 ping 通，配置 SSH 信任关系
- 节点验证

2.2 方案

准备四台虚拟机，由于之前已经准备过一台，所以只需再准备三台新的虚拟机即可，安装 hadoop，使所有节点可以 ping 通，配置 SSH 信任关系，如图-1 所示：

图-1

2.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：环境准备

1）三台机器配置主机名为 node1、node2、node3，配置 ip 地址（ip 如图-1 所示），yum 源（系统源）

2）编辑/etc/hosts（四台主机同样操作，以 nn01 为例）

```
[root@nn01 ~]# vim /etc/hosts
192.168.1.21  nn01
```

```
192.168.1.22   node1
192.168.1.23   node2
192.168.1.24   node3
```

3）安装 java 环境，在 node1，node2，node3 上面操作（以 node1 为例）

```
[root@node1 ~]# yum -y install java-1.8.0-openjdk-devel
```

4）布置 SSH 信任关系

```
[root@nn01 ~]# vim /etc/ssh/ssh_config      //第一次登陆不需要输入 yes
Host *
        GSSAPIAuthentication yes
        StrictHostKeyChecking no
[root@nn01 .ssh]# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:UcI8OCezw92aArY5+zPtOrJ9oI1ojRE3EAZ1mgndYQM root@nn01
The key's randomart image is:
+---[RSA 2048]----+
|        o*E*=.   |
|          +XB+.  |
|        ..=Oo.   |
|        o.+o...  |
|       .S+.. o   |
|        + .=o    |
|         o+oo    |
|        o+=.o    |
|        o==O.    |
+----[SHA256]-----+
[root@nn01 .ssh]# for i in 21 22 23 24 ; do  ssh-copy-id  192.168.1.$i; done
//部署公钥给 nn01，node1，node2，node3
```

5）测试信任关系

```
[root@nn01 .ssh]# ssh node1
Last login: Fri Sep  7 16:52:00 2018 from 192.168.1.21
[root@node1 ~]# exit
```

```
logout
Connection to node1 closed.
[root@nn01 .ssh]# ssh node2
Last login: Fri Sep   7 16:52:05 2018 from 192.168.1.21
[root@node2 ~]# exit
logout
Connection to node2 closed.
[root@nn01 .ssh]# ssh node3
```

步骤二：配置 hadoop

1）修改 slaves 文件

```
[root@nn01 ~]# cd   /usr/local/hadoop/etc/hadoop
[root@nn01 hadoop]# vim slaves
node1
node2
node3
```

2）hadoop 的核心配置文件 core-site

```
[root@nn01 hadoop]# vim core-site.xml
<configuration>
<property>
        <name>fs.defaultFS</name>
        <value>hdfs://nn01:9000</value>
    </property>
    <property>
        <name>hadoop.tmp.dir</name>
        <value>/var/hadoop</value>
    </property>
</configuration>
[root@nn01 hadoop]# mkdir /var/hadoop           //hadoop 的数据根目录
[root@nn01 hadoop]# ssh node1 mkdir /var/hadoop
[root@nn01 hadoop]# ssh node2 mkdir /var/hadoop
[root@nn01 hadoop]# ssh node3 mkdir /var/hadoop
```

3）配置 hdfs-site 文件

```
[root@nn01 hadoop]# vim hdfs-site.xml
<configuration>
 <property>
        <name>dfs.namenode.http-address</name>
        <value>nn01:50070</value>
    </property>
    <property>
        <name>dfs.namenode.secondary.http-address</name>
        <value>nn01:50090</value>
    </property>
    <property>
        <name>dfs.replication</name>
        <value>2</value>
    </property>
</configuration>
```

4）同步配置到 node1，node2，node3

```
    [root@nn01 hadoop]# yum –y install rsync   //同步的主机都要安装 rsync
    [root@nn01 hadoop]# for i in 22 23 24 ; do rsync -aSH --delete
/usr/local/hadoop/
    \    192.168.1.$i:/usr/local/hadoop/   -e 'ssh' & done
    [1] 23260
    [2] 23261
    [3] 23262
```

5）查看是否同步成功

```
    [root@nn01 hadoop]# ssh node1 ls /usr/local/hadoop/
    bin
    etc
    include
    lib
    libexec
    LICENSE.txt
    NOTICE.txt
    bb
    README.txt
    sbin
    share
    aa
    [root@nn01 hadoop]# ssh node2 ls /usr/local/hadoop/
    bin
    etc
    include
    lib
    libexec
    LICENSE.txt
    NOTICE.txt
```

```
bb
README.txt
sbin
share
aa
[root@nn01 hadoop]# ssh node3 ls /usr/local/hadoop/
bin
etc
include
lib
libexec
LICENSE.txt
NOTICE.txt
bb
README.txt
sbin
share
aa
```

步骤三：格式化

```
[root@nn01 hadoop]# cd /usr/local/hadoop/
[root@nn01 hadoop]# ./bin/hdfs namenode -format              //格式化
namenode
[root@nn01 hadoop]# ./sbin/start-dfs.sh          //启动
[root@nn01 hadoop]# jps          //验证角色
23408 NameNode
23700 Jps
23591 SecondaryNameNode
[root@nn01 hadoop]# ./bin/hdfs dfsadmin -report          //查看集群是否组建
成功
Live datanodes (3):          //有三个角色成功
```

Top

NSD ARCHITECTURE DAY06

案例 1：安装与部署
案例 2：Hadoop 词频统计
案例 3：节点管理
案例 4：NFS 配置

1 案例 1：安装与部署

1.1 问题

本案例要求：

对 mapred 和 yarn 文件进行配置
验证访问 Hadoop

1.2 方案

在 day05 准备好的环境下给 master （nn01）主机添加 ResourceManager 的角色，在 node1，node2，node3 上面添加 NodeManager 的角色，如表-1 所示：

表-1

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：安装与部署 hadoop

1）配置 mapred-site（nn01 上面操作）

```
[root@nn01 ~]# cd /usr/local/hadoop/etc/hadoop/
[root@nn01 hadoop]# mv mapred-site.xml.template mapred-site.xml
[root@nn01 hadoop]# vim mapred-site.xml
<configuration>
<property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

2）配置 yarn-site（nn01 上面操作）

```
[root@nn01 hadoop]# vim yarn-site.xml
<configuration>
<!-- Site specific YARN configuration properties -->
<property>
        <name>yarn.resourcemanager.hostname</name>
        <value>nn01</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
```

```
</configuration>
```

3）同步配置（nn01 上面操作）

```
[root@nn01 hadoop]# for i in {22..24}; do rsync -aSH --delete /usr/local/hadoop/ 192.168.1.$i:/usr/local/hadoop/  -e 'ssh' & done
[1] 712
[2] 713
[3] 714
```

4）验证配置（nn01 上面操作）

```
[root@nn01 hadoop]# cd /usr/local/hadoop
[root@nn01 hadoop]# ./sbin/start-dfs.sh
Starting namenodes on [nn01]
nn01: namenode running as process 23408. Stop it first.
node1: datanode running as process 22409. Stop it first.
node2: datanode running as process 22367. Stop it first.
node3: datanode running as process 22356. Stop it first.
Starting secondary namenodes [nn01]
nn01: secondarynamenode running as process 23591. Stop it first.
[root@nn01 hadoop]# ./sbin/start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-root-resourcemanager-nn01.out
node2: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-node2.out
node3: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-node3.out
node1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-node1.out
[root@nn01 hadoop]# jps     //nn01 查看有 ResourceManager
23408 NameNode
1043 ResourceManager
1302 Jps
23591 SecondaryNameNode
[root@nn01 hadoop]# ssh node1 jps        //node1 查看有 NodeManager
25777 Jps
22409 DataNode
25673 NodeManager
[root@nn01 hadoop]# ssh node2 jps        //node1 查看有 NodeManager
25729 Jps
25625 NodeManager
22367 DataNode
```

```
[root@nn01 hadoop]# ssh node3 jps            //node1 查看有 NodeManager
22356 DataNode
25620 NodeManager
25724 Jps
```

5）web 访问 hadoop

```
http://192.168.1.21:50070/              //--namenode web 页面（nn01）
http://192.168.1.21:50090/          //--secondory namenode web 页面（nn01）
http://192.168.1.22:50075/                  //--datanode  web  页 面
（node1,node2,node3）
http://192.168.1.21:8088/          //--resourcemanager web 页面（nn01）
http://192.168.1.22:8042/                  //--nodemanager  web  页 面
（node1,node2,node3）
```

2 案例 2：Hadoop 词频统计
2.1 问题

本案例要求：

在集群文件系统里创建文件夹
上传要分析的文件到目录中
分析上传文件
展示结果

2.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：词频统计

```
[root@nn01 hadoop]# ./bin/hadoop fs -ls /            //查看集群文件系统的根，没
有内容
[root@nn01 hadoop]# ./bin/hadoop fs -mkdir  /aaa
//在集群文件系统下创建 aaa 目录
[root@nn01 hadoop]# ./bin/hadoop fs -ls /            //再次查看，有刚创建的 aaa
目录
Found 1 items
drwxr-xr-x   - root supergroup          0 2018-09-10 09:56 /aaa
[root@nn01 hadoop]#  ./bin/hadoop fs -touchz  /fa     //在集群文件系统下创
建 fa 文件
[root@nn01 hadoop]# ./bin/hadoop fs -put *.txt /aaa
//上传*.txt 到集群文件系统下的 aaa 目录
[root@nn01 hadoop]#  ./bin/hadoop fs -ls /aaa     //查看
```

```
    Found 3 items
    -rw-r--r--    2  root  supergroup         86424  2018-09-10  09:58
/aaa/LICENSE.txt
    -rw-r--r--    2  root  supergroup         14978  2018-09-10  09:58
/aaa/NOTICE.txt
    -rw-r--r--    2  root  supergroup          1366  2018-09-10  09:58
/aaa/README.txt
    [root@nn01 hadoop]# ./bin/hadoop fs -get   /aaa   //下载集群文件系统的 aaa
目录
    [root@nn01 hadoop]# ./bin/hadoop jar   \
     share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.6.jar
wordcount /aaa /bbb      //hadoop 集群分析大数据，hadoop 集群/aaa 里的数据存到
hadoop 集群/bbb 下
    [root@nn01 hadoop]# ./bin/hadoop fs -cat /bbb/*          //查看集群里的数据
```

3 案例 3：节点管理

3.1 问题

本案例要求：

- 增加一个新的节点
- 查看状态
- 删除节点

3.2 方案：

另外准备两台主机，node4 和 nfsgw，作为新添加的节点和网关，具体要求如表-2 所示：

表-2

3.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：增加节点

1）增加一个新的节点 node4

```
    [root@hadoop5 ~]# echo node4 > /etc/hostname       //更改主机名为 node4
    [root@hadoop5 ~]# hostname node4
    [root@node4 ~]# yum -y install rsync
    [root@node4 ~]# yum -y install java-1.8.0-openjdk-devel
    [root@node4 ~]# mkdir /var/hadoop
    [root@nn01 .ssh]# ssh-copy-id 192.168.1.25
    [root@nn01 .ssh]# vim /etc/hosts
    192.168.1.21   nn01
    192.168.1.22   node1
    192.168.1.23   node2
    192.168.1.24   node3
    192.168.1.25   node4
    [root@nn01 .ssh]# scp /etc/hosts 192.168.1.25:/etc/
    [root@nn01 ~]# cd /usr/local/hadoop/
    [root@nn01 hadoop]# vim ./etc/hadoop/slaves
    node1
    node2
    node3
    node4
    [root@nn01 hadoop]# for i in {22..25}; do rsync -aSH --delete /usr/local/hadoop/
    \ 192.168.1.$i:/usr/local/hadoop/   -e 'ssh' & done            //同步配置
    [1] 1841
    [2] 1842
    [3] 1843
    [4] 1844
    [root@node4 hadoop]# ./sbin/hadoop-daemon.sh start datanode   //启动
```

2）查看状态

```
    [root@node4 hadoop]# jps
    24439 Jps
    24351 DataNode
```

3）设置同步带宽

```
    [root@node4 hadoop]# ./bin/hdfs dfsadmin -setBalancerBandwidth 60000000
    Balancer bandwidth is set to 60000000
    [root@node4 hadoop]# ./sbin/start-balancer.sh
```

4）删除节点

```
    [root@nn01 hadoop]# vim /usr/local/hadoop/etc/hadoop/slaves
    //去掉之前添加的 node4
    node1
    node2
    node3
    [root@nn01 hadoop]# vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml
    //在此配置文件里面加入下面四行
    <property>
```

```
        <name>dfs.hosts.exclude</name>
        <value>/usr/local/hadoop/etc/hadoop/exclude</value>
    </property>
    [root@nn01 hadoop]# vim /usr/local/hadoop/etc/hadoop/exclude
    node4
```

5）导出数据

```
    [root@nn01 hadoop]# ./bin/hdfs dfsadmin -refreshNodes
    Refresh nodes successful
    [root@nn01 hadoop]# ./bin/hdfs dfsadmin -report    //查看 node4 显示
Decommissioned
    Dead datanodes (1):
    Name: 192.168.1.25:50010 (node4)
    Hostname: node4
    Decommission Status : Decommissioned
    Configured Capacity: 17168314368 (15.99 GB)
    DFS Used: 12288 (12 KB)
    Non DFS Used: 1656664064 (1.54 GB)
    DFS Remaining: 15511638016 (14.45 GB)
    DFS Used%: 0.00%
    DFS Remaining%: 90.35%
    Configured Cache Capacity: 0 (0 B)
    Cache Used: 0 (0 B)
    Cache Remaining: 0 (0 B)
    Cache Used%: 100.00%
    Cache Remaining%: 0.00%
    Xceivers: 1
    Last contact: Mon Sep 10 10:59:58 CST 2018
    [root@node4 hadoop]# ./sbin/hadoop-daemon.sh stop datanode    //停止
datanode
    stopping datanode
    [root@node4 hadoop]# ./sbin/yarn-daemon.sh start nodemanager
    //yarn 增加 nodemanager
    [root@node4 hadoop]# ./sbin/yarn-daemon.sh stop  nodemanager  //停止
nodemanager
    stopping nodemanager
    [root@node4 hadoop]# ./bin/yarn node -list
    //yarn 查看节点状态，还是有 node4 节点，要过一段时间才会消失
    18/09/10 11:04:50 INFO client.RMProxy: Connecting to ResourceManager at
nn01/192.168.1.21:8032
    Total Nodes:4
             Node-Id              Node-State        Node-Http-Address
Number-of-Running-Containers
```

```
         node3:34628                    RUNNING                   node3:8042
0
         node2:36300                    RUNNING                   node2:8042
0
         node4:42459                    RUNNING                   node4:8042
0
         node1:39196                    RUNNING                   node1:8042
0
```

4 案例 4：NFS 配置

4.1 问题

本案例要求：

- 创建代理用户
- 启动一个新系统，禁用 Selinux 和 firewalld
- 配置 NFSWG
- 启动服务
- 挂载 NFS 并实现开机自启

4.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：基础准备

1）更改主机名，配置/etc/hosts（/etc/hosts 在 nn01 和 nfsgw 上面配置）

```
[root@localhost ~]# echo nfsgw > /etc/hostname
[root@localhost ~]# hostname nfsgw
[root@nn01 hadoop]# vim /etc/hosts
192.168.1.21    nn01
192.168.1.22    node1
192.168.1.23    node2
192.168.1.24    node3
192.168.1.25    node4
192.168.1.26    nfsgw
```

2）创建代理用户（nn01 和 nfsgw 上面操作），以 nn01 为例子

```
[root@nn01 hadoop]# groupadd -g 200 nfs
[root@nn01 hadoop]# useradd -u 200 -g nfs nfs
```

3）配置 core-site.xml

```
[root@nn01 hadoop]# ./sbin/stop-all.sh     //停止所有服务
This script is Deprecated. Instead use stop-dfs.sh and stop-yarn.sh
Stopping namenodes on [nn01]
nn01: stopping namenode
node2: stopping datanode
node4: no datanode to stop
node3: stopping datanode
node1: stopping datanode
Stopping secondary namenodes [nn01]
nn01: stopping secondarynamenode
stopping yarn daemons
stopping resourcemanager
node2: stopping nodemanager
node3: stopping nodemanager
node4: no nodemanager to stop
node1: stopping nodemanager
...
[root@nn01 hadoop]# cd etc/hadoop
[root@nn01 hadoop]# >exclude
[root@nn01 hadoop]# vim core-site.xml
    <property>
        <name>hadoop.proxyuser.nfs.groups</name>
        <value>*</value>
    </property>
    <property>
        <name>hadoop.proxyuser.nfs.hosts</name>
        <value>*</value>
    </property>
```

4）同步配置到 node1，node2，node3

```
[root@nn01 hadoop]# for i in {22..24}; do rsync -aSH --delete /usr/local/hadoop/ 192.168.1.$i:/usr/local/hadoop/  -e 'ssh' & done
[4] 2722
[5] 2723
[6] 2724
```

5）启动集群

```
[root@nn01 hadoop]# /usr/local/hadoop/sbin/start-dfs.sh
```

6）查看状态

```
[root@nn01 hadoop]# /usr/local/hadoop/bin/hdfs   dfsadmin -report
```

步骤二：NFSGW 配置

1）安装 java-1.8.0-openjdk-devel 和 rsync

```
[root@nfsgw ~]# yum -y install java-1.8.0-openjdk-devel
[root@nfsgw ~]# yum -y install rsync
[root@nn01 hadoop]# rsync -avSH --delete \
/usr/local/hadoop/ 192.168.1.26:/usr/local/hadoop/   -e 'ssh'
```

2）创建数据根目录 /var/hadoop（在 NFSGW 主机上面操作）

```
[root@nfsgw ~]# mkdir /var/hadoop
```

3）创建转储目录，并给用户 nfs 赋权

```
[root@nfsgw ~]# mkdir /var/nfstmp
[root@nfsgw ~]# chown nfs:nfs /var/nfstmp
```

4）给/usr/local/hadoop/logs 赋权（在 NFSGW 主机上面操作）

```
[root@nfsgw ~]# setfacl -m u:nfs:rwx /usr/local/hadoop/logs
[root@nfsgw ~]# vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml
    <property>
        <name>nfs.exports.allowed.hosts</name>
        <value>* rw</value>
    </property>
    <property>
        <name>nfs.dump.dir</name>
        <value>/var/nfstmp</value>
    </property>
```

5）可以创建和删除即可

```
[root@nfsgw ~]# su - nfs
[nfs@nfsgw ~]$ cd /var/nfstmp/
[nfs@nfsgw nfstmp]$ touch 1
[nfs@nfsgw nfstmp]$ ls
1
[nfs@nfsgw nfstmp]$ rm -rf 1
[nfs@nfsgw nfstmp]$ ls
[nfs@nfsgw nfstmp]$ cd /usr/local/hadoop/logs/
[nfs@nfsgw logs]$ touch 1
```

```
    [nfs@nfsgw logs]$ ls
    1                                   hadoop-root-secondarynamenode-nn01.log
yarn-root-resourcemanager-nn01.log
    hadoop-root-namenode-nn01.log
hadoop-root-secondarynamenode-nn01.out
yarn-root-resourcemanager-nn01.out
    hadoop-root-namenode-nn01.out
hadoop-root-secondarynamenode-nn01.out.1
    hadoop-root-namenode-nn01.out.1   SecurityAuth-root.audit
    [nfs@nfsgw logs]$ rm -rf 1
    [nfs@nfsgw logs]$ ls
```

6）启动服务

```
    [root@nfsgw          ~]#          /usr/local/hadoop/sbin/hadoop-daemon.sh
--script ./bin/hdfs start portmap          //portmap 服务只能用 root 用户启动
    starting                portmap,                logging                to
/usr/local/hadoop/logs/hadoop-root-portmap-nfsgw.out
    [root@nfsgw ~]# jps
    23714 Jps
    23670 Portmap
    [root@nfsgw ~]# su - nfs
    Last login: Mon Sep 10 12:31:58 CST 2018 on pts/0
    [nfs@nfsgw ~]$ cd /usr/local/hadoop/
    [nfs@nfsgw  hadoop]$  ./sbin/hadoop-daemon.sh    --script  ./bin/hdfs  start
nfs3
    //nfs3 只能用代理用户启动
    starting nfs3, logging to /usr/local/hadoop/logs/hadoop-nfs-nfs3-nfsgw.out
    [nfs@nfsgw hadoop]$ jps
    1362 Jps
    1309 Nfs3
    [root@nfsgw hadoop]# jps                //root 用户执行可以看到 portmap 和
nfs3
    1216 Portmap
    1309 Nfs3
    1374 Jps
```

7）实现客户端挂载（客户端可以用 node4 这台主机）

```
    [root@node4 ~]# rm -rf /usr/local/hadoop
    [root@node4 ~]# yum -y install nfs-utils
    [root@node4 ~]# mount -t nfs -o \
    vers=3,proto=tcp,nolock,noatime,sync,noacl  192.168.1.26:/    /mnt/    //挂
载
```

```
    [root@node4 ~]# cd /mnt/
    [root@node4 mnt]# ls
    aaa   bbb   fa   system   tmp
    [root@node4 mnt]# touch a
    [root@node4 mnt]# ls
    a   aaa   bbb   fa   system   tmp
    [root@node4 mnt]# rm -rf a
    [root@node4 mnt]# ls
    aaa   bbb   fa   system   tmp
```

8）实现开机自动挂载

```
    [root@node4 ~]# vim /etc/fstab
    192.168.1.26:/                                    /mnt/                      nfs
vers=3,proto=tcp,nolock,noatime,sync,noacl,_netdev 0 0
    [root@node4 ~]# mount -a
    [root@node4 ~]# df -h
    192.168.1.26:/     64G   6.2G     58G   10% /mnt
    [root@node4 ~]# rpcinfo -p 192.168.1.26
       program vers proto     port   service
        100005      3     udp     4242   mountd
        100005      1     tcp     4242   mountd
        100000      2     udp      111   portmapper
        100000      2     tcp      111   portmapper
        100005      3     tcp     4242   mountd
        100005      2     tcp     4242   mountd
        100003      3     tcp     2049   nfs
        100005      2     udp     4242   mountd
        100005      1     udp     4242   mountd
```

Top

NSD ARCHITECTURE DAY07

1 案例 1：Zookeeper 安装

1.1 问题

本案例要求：

搭建 Zookeeper 集群并查看各服务器的角色
停止 Leader 并查看各服务器的角色

1.2 步骤

实现此案例需要按照如下步骤进行。

2 步骤一：安装 Zookeeper

1）编辑/etc/hosts ,所有集群主机可以相互 ping 通（在 nn01 上面配置，同步到 node1，node2，node3）

```
[root@nn01 hadoop]# vim /etc/hosts
192.168.1.21   nn01
192.168.1.22   node1
192.168.1.23   node2
192.168.1.24   node3
192.168.1.25   node4
[root@nn01 hadoop]# for i in {22..24}   \
do      \
scp /etc/hosts 192.168.1.$i:/etc/      \
done            //同步配置
hosts          100%   253    639.2KB/s     00:00
hosts          100%   253    497.7KB/s     00:00
hosts          100%   253    662.2KB/s     00:00
```

2）安装 java-1.8.0-openjdk-devel,由于之前的 hadoop 上面已经安装过，这里不再安装，若是新机器要安装

3）zookeeper 解压拷贝到 /usr/local/zookeeper

```
[root@nn01 ~]# tar -xf zookeeper-3.4.10.tar.gz
[root@nn01 ~]# mv zookeeper-3.4.10 /usr/local/zookeeper
```

4）配置文件改名，并在最后添加配置

```
[root@nn01 ~]# cd /usr/local/zookeeper/conf/
[root@nn01 conf]# ls
configuration.xsl   log4j.properties   zoo_sample.cfg
[root@nn01 conf]# mv zoo_sample.cfg   zoo.cfg
[root@nn01 conf]# chown root.root zoo.cfg
[root@nn01 conf]# vim zoo.cfg
server.1=node1:2888:3888
```

```
    server.2=node2:2888:3888
    server.3=node3:2888:3888
    server.4=nn01:2888:3888:observer
```

5）拷贝 /usr/local/zookeeper 到其他集群主机

```
    [root@nn01 conf]# for i in {22..24}; do rsync -aSH --delete
/usr/local/zookeeper/ 192.168.1.$i:/usr/local/zookeeper   -e 'ssh' & done
    [4] 4956
    [5] 4957
    [6] 4958
```

6）创建 mkdir /tmp/zookeeper，每一台都要

```
    [root@nn01 conf]# mkdir /tmp/zookeeper
    [root@nn01 conf]# ssh node1 mkdir /tmp/zookeeper
    [root@nn01 conf]# ssh node2 mkdir /tmp/zookeeper
    [root@nn01 conf]# ssh node3 mkdir /tmp/zookeeper
```

7）创建 myid 文件，id 必须与配置文件里主机名对应的 server.(id) 一致

```
    [root@nn01 conf]# echo 4 >/tmp/zookeeper/myid
    [root@nn01 conf]# ssh node1 'echo 1 >/tmp/zookeeper/myid'
    [root@nn01 conf]# ssh node2 'echo 2 >/tmp/zookeeper/myid'
    [root@nn01 conf]# ssh node3 'echo 3 >/tmp/zookeeper/myid'
```

8）启动服务，单启动一台无法查看状态，需要启动全部集群以后才能查看状态，每一台上面都要手工启动（以 nn01 为例子）

```
    [root@nn01 conf]# /usr/local/zookeeper/bin/zkServer.sh start
    ZooKeeper JMX enabled by default
    Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
    Starting zookeeper ... STARTED
```

注意：刚启动 zookeeper 查看状态的时候报错，启动的数量要保证半数以上，这时再去看就成功了

9）查看状态

```
    [root@nn01 conf]# /usr/local/zookeeper/bin/zkServer.sh status
    ZooKeeper JMX enabled by default
    Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
    Mode: observe
    [root@nn01 conf]# /usr/local/zookeeper/bin/zkServer.sh stop
```

```
//关闭之后查看状态其他服务器的角色
ZooKeeper JMX enabled by default
Using config: /usr/local/zookeeper/bin/../conf/zoo.cfg
Stopping zookeeper ... STOPPED
[root@nn01 conf]# yum -y install telnet
[root@nn01 conf]# telnet node3 2181
Trying 192.168.1.24...
Connected to node3.
Escape character is '^]'.
ruok            //发送
imokConnection closed by foreign host.            //imok 回应的结果
```

10）利用 api 查看状态（nn01 上面操作）

```
[root@nn01 conf]# /usr/local/zookeeper/bin/zkServer.sh start
[root@nn01 conf]# vim api.sh
#!/bin/bash
function getstatus(){
    exec 9<>/dev/tcp/$1/2181 2>/dev/null
    echo stat >&9
    MODE=$(cat <&9 |grep -Po "(?<=Mode:).*")
    exec 9<&-
    echo ${MODE:-NULL}
}
for i in node{1..3} nn01;do
    echo -ne "${i}\t"
    getstatus ${i}
done
[root@nn01 conf]# chmod 755 api.sh
[root@nn01 conf]# ./api.sh
node1     follower
node2     leader
node3     follower
nn01     observer
```

3 案例 2：Kafka 集群实验

3.1 问题

本案例要求：

- 利用 Zookeeper 搭建一个 Kafka 集群
- 创建一个 topic
- 模拟生产者发布消息
- 模拟消费者接收消息

3.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：搭建 Kafka 集群

1）解压 kafka 压缩包

Kafka 在 node1，node2，node3 上面操作即可

```
[root@node1 ~]# tar -xf kafka_2.10-0.10.2.1.tgz
```

2）把 kafka 拷贝到 /usr/local/kafka 下面

```
[root@node1 ~]# mv kafka_2.10-0.10.2.1 /usr/local/kafka
```

3）修改配置文件 /usr/local/kafka/config/server.properties

```
[root@node1 ~]# cd /usr/local/kafka/config
[root@node1 config]# vim server.properties
broker.id=22
zookeeper.connect=node1:2181,node2:2181,node3:2181
```

4）拷贝 kafka 到其他主机，并修改 broker.id ,不能重复

```
[root@node1 config]# for i in 23 24; do rsync -aSH --delete /usr/local/kafka 192.168.1.$i:/usr/local/; done
[1] 27072
[2] 27073
[root@node2 ~]# vim /usr/local/kafka/config/server.properties
//node2 主机修改
broker.id=23
[root@node3 ~]# vim /usr/local/kafka/config/server.properties
//node3 主机修改
broker.id=24
```

5）启动 kafka 集群（node1，node2，node3 启动）

```
[root@node1 local]# /usr/local/kafka/bin/kafka-server-start.sh -daemon /usr/local/kafka/config/server.properties
[root@node1 local]# jps          //出现 kafka
26483 DataNode
27859 Jps
```

```
    27833 Kafka
    26895 QuorumPeerMain
```

6）验证配置，创建一个 topic

```
    [root@node1 local]# /usr/local/kafka/bin/kafka-topics.sh --create
--partitions 1 --replication-factor 1 --zookeeper node3:2181 --topic aa
    Created topic "aa".
```

7) 模拟生产者，发布消息

```
    [root@node2 ~]# /usr/local/kafka/bin/kafka-console-producer.sh \
    --broker-list node2:9092 --topic aa          //写一个数据
    ccc
    ddd
```

9）模拟消费者，接收消息

```
    [root@node3 ~]# /usr/local/kafka/bin/kafka-console-consumer.sh \
    --bootstrap-server node1:9092 --topic aa          //这边会直接同步
    ccc
    ddd
```

注意：kafka 比较吃内存，做完这个 kafka 的实验可以把它停了

4 案例 3：Hadoop 高可用

4.1 问题

本案例要求：

- 配置 Hadoop 的高可用
- 修改配置文件

4.2 方案

配置 Hadoop 的高可用，解决 NameNode 单点故障问题，使用之前搭建好的 hadoop 集群，新添加一台 nn02，ip 为 192.168.1.25，之前有一台 node4 主机，可以用这台主机，具体要求如图-1 所示：

图-1

4.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：hadoop 的高可用

1）停止所有服务（由于 kafka 的实验做完之后就已经停止，这里不在重复）

```
[root@nn01 ~]# cd /usr/local/hadoop/
[root@nn01 hadoop]# ./sbin/stop-all.sh    //停止所有服务
```

2）启动 zookeeper（需要一台一台的启动）这里以 nn01 为例子

```
[root@nn01 hadoop]# /usr/local/zookeeper/bin/zkServer.sh start
[root@nn01 hadoop]# sh /usr/local/zookeeper/conf/api.sh //利用之前写好的
脚本查看
node1       follower
node2       leader
node3       follower
nn01       observer
```

3）新加一台机器 nn02，这里之前有一台 node4，可以用这个作为 nn02

```
[root@node4 ~]# echo nn02 > /etc/hostname
[root@node4 ~]# hostname nn02
```

4）修改 vim /etc/hosts

```
[root@nn01 hadoop]# vim /etc/hosts
192.168.1.21    nn01
192.168.1.25    nn02
192.168.1.22    node1
192.168.1.23    node2
192.168.1.24    node3
```

5）同步到 nn02，node1，node2，node3

```
[root@nn01 hadoop]# for i in {22..25}; do rsync -aSH --delete /etc/hosts
192.168.1.$i:/etc/hosts    -e 'ssh' & done
[1] 14355
[2] 14356
[3] 14357
[4] 14358
```

6）配置 SSH 信任关系

注意：nn01 和 nn02 互相连接不需要密码，nn02 连接自己和 node1，node2，node3 同样不需要密码

```
[root@nn02 ~]# vim /etc/ssh/ssh_config
Host *
        GSSAPIAuthentication yes
        StrictHostKeyChecking no
[root@nn01 hadoop]# cd /root/.ssh/
[root@nn01 .ssh]# scp id_rsa id_rsa.pub   nn02:/root/.ssh/
//把 nn01 的公钥私钥考给 nn02
```

7）所有的主机删除/var/hadoop/*

```
[root@nn01 .ssh]# rm -rf /var/hadoop/*
[root@nn01 .ssh]# ssh nn02 rm -rf /var/hadoop/*
[root@nn01 .ssh]# ssh node1 rm -rf /var/hadoop/*
[root@nn01 .ssh]# ssh node2 rm -rf /var/hadoop/*
[root@nn01 .ssh]# ssh node3 rm -rf /var/hadoop/*
```

8）配置 core-site

```
    [root@nn01 .ssh]# vim /usr/local/hadoop/etc/hadoop/core-site.xml
    <configuration>
    <property>
            <name>fs.defaultFS</name>
            <value>hdfs://nsdcluster</value>
    //nsdcluster 是随便起的名。相当于一个组，访问的时候访问这个组
        </property>
        <property>
            <name>hadoop.tmp.dir</name>
            <value>/var/hadoop</value>
        </property>
        <property>
            <name>ha.zookeeper.quorum</name>
            <value>node1:2181,node2:2181,node3:2181</value>
//zookeepe 的地址
        </property>
        <property>
            <name>hadoop.proxyuser.nfs.groups</name>
            <value>*</value>
        </property>
        <property>
            <name>hadoop.proxyuser.nfs.hosts</name>
            <value>*</value>
        </property>
    </configuration>
```

9）配置 hdfs-site

```
    [root@nn01 ~]# vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml
    <configuration>
        <property>
            <name>dfs.replication</name>
            <value>2</value>
        </property>
        <property>
            <name>dfs.nameservices</name>
            <value>nsdcluster</value>
        </property>
        <property>
            <name>dfs.ha.namenodes.nsdcluster</name>
    //nn1,nn2 名称固定，是内置的变量，nsdcluster 里面有 nn1，nn2
            <value>nn1,nn2</value>
        </property>
        <property>
            <name>dfs.namenode.rpc-address.nsdcluster.nn1</name>
    //声明 nn1 8020 为通讯端口，是 nn01 的 rpc 通讯端口
            <value>nn01:8020</value>
        </property>
        <property>
            <name>dfs.namenode.rpc-address.nsdcluster.nn2</name>
    //声明 nn2 是谁，nn02 的 rpc 通讯端口
            <value>nn02:8020</value>
        </property>
        <property>
            <name>dfs.namenode.http-address.nsdcluster.nn1</name>
    //nn01 的 http 通讯端口
            <value>nn01:50070</value>
        </property>
        <property>
            <name>dfs.namenode.http-address.nsdcluster.nn2</name>
    //nn01 和 nn02 的 http 通讯端口
            <value>nn02:50070</value>
        </property>
        <property>
            <name>dfs.namenode.shared.edits.dir</name>
    //指定 namenode 元数据存储在 journalnode 中的路径

<value>qjournal://node1:8485;node2:8485;node3:8485/nsdcluster</value>
        </property>
        <property>
```

```
            <name>dfs.journalnode.edits.dir</name>
    //指定 journalnode 日志文件存储的路径
            <value>/var/hadoop/journal</value>
        </property>
        <property>
            <name>dfs.client.failover.proxy.provider.nsdcluster</name>
    //指定 HDFS 客户端连接 active namenode 的 java 类

<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProx
yProvider</value>
        </property>
        <property>
            <name>dfs.ha.fencing.methods</name>                    //配
置隔离机制为 ssh
            <value>sshfence</value>
        </property>
        <property>
            <name>dfs.ha.fencing.ssh.private-key-files</name>      //指定密钥
的位置
            <value>/root/.ssh/id_rsa</value>
        </property>
        <property>
            <name>dfs.ha.automatic-failover.enabled</name>          //开启
自动故障转移
            <value>true</value>
        </property>
    </configuration>
```

10）配置 yarn-site

```
    [root@nn01 ~]# vim /usr/local/hadoop/etc/hadoop/yarn-site.xml
    <configuration>
    <!-- Site specific YARN configuration properties -->
        <property>
            <name>yarn.nodemanager.aux-services</name>
            <value>mapreduce_shuffle</value>
        </property>
        <property>
            <name>yarn.resourcemanager.ha.enabled</name>
            <value>true</value>
        </property>
        <property>
            <name>yarn.resourcemanager.ha.rm-ids</name>
//rm1,rm2 代表 nn01 和 nn02
```

```
            <value>rm1,rm2</value>
        </property>
        <property>
            <name>yarn.resourcemanager.recovery.enabled</name>
            <value>true</value>
        </property>
        <property>
            <name>yarn.resourcemanager.store.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMState
Store</value>
        </property>
        <property>
            <name>yarn.resourcemanager.zk-address</name>
            <value>node1:2181,node2:2181,node3:2181</value>
        </property>
        <property>
            <name>yarn.resourcemanager.cluster-id</name>
            <value>yarn-ha</value>
        </property>
        <property>
            <name>yarn.resourcemanager.hostname.rm1</name>
            <value>nn01</value>
        </property>
        <property>
            <name>yarn.resourcemanager.hostname.rm2</name>
            <value>nn02</value>
        </property>
    </configuration>
```

11）同步到 nn02，node1，node2，node3

```
    [root@nn01 ~]# for i in {22..25}; do rsync -aSH --delete /usr/local/hadoop/
192.168.1.$i:/usr/local/hadoop   -e 'ssh' & done
    [1] 25411
    [2] 25412
    [3] 25413
    [4] 25414
```

12）删除所有机器上面的/user/local/hadoop/logs，方便排错

```
    [root@nn01 ~]# for i in {21..25}; do ssh 192.168.1.$i rm -rf
/usr/local/hadoop/logs ; done
```

13）同步配置

```
[root@nn01 ~]# for i in {22..25}; do rsync -aSH --delete /usr/local/hadoop 192.168.1.$i:/usr/local/hadoop -e 'ssh' & done
[1] 28235
[2] 28236
[3] 28237
[4] 28238
```

5 案例 4：高可用验证

5.1 问题

本案例要求：

- 初始化集群
- 验证集群

5.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：验证 hadoop 的高可用

1）初始化 ZK 集群

```
[root@nn01 ~]# /usr/local/hadoop/bin/hdfs zkfc -formatZK
...
18/09/11 15:43:35 INFO ha.ActiveStandbyElector: Successfully created /hadoop-ha/nsdcluster in ZK      //出现 Successfully 即为成功
...
```

2）在 node1，node2，node3 上面启动 journalnode 服务（以 node1 为例子）

```
[root@node1 ~]# /usr/local/hadoop/sbin/hadoop-daemon.sh start journalnode
starting journalnode, logging to /usr/local/hadoop/logs/hadoop-root-journalnode-node1.out
[root@node1 ~]# jps
29262 JournalNode
26895 QuorumPeerMain
29311 Jps
```

3）格式化，先在 node1，node2，node3 上面启动 journalnode 才能格式化

```
[root@nn01 ~]# /usr/local/hadoop//bin/hdfs   namenode   -format
//出现 Successfully 即为成功
[root@nn01 hadoop]# ls /var/hadoop/
dfs
```

4）nn02 数据同步到本地 /var/hadoop/dfs

```
[root@nn02 ~]# cd /var/hadoop/
[root@nn02 hadoop]# ls
[root@nn02 hadoop]# rsync -aSH   nn01:/var/hadoop/   /var/hadoop/
[root@nn02 hadoop]# ls
dfs
```

5）初始化 JNS

```
[root@nn01 hadoop]# /usr/local/hadoop/bin/hdfs namenode -initializeSharedEdits
18/09/11 16:26:15 INFO client.QuorumJournalManager: Successfully started new epoch 1          //出现 Successfully，成功开启一个节点
```

6）停止 journalnode 服务（node1，node2，node3）

```
[root@node1 hadoop]# /usr/local/hadoop/sbin/hadoop-daemon.sh stop journalnode
stopping journalnode
[root@node1 hadoop]# jps
29346 Jps
26895 QuorumPeerMain
```

步骤二：启动集群

1）nn01 上面操作

```
[root@nn01 hadoop]# /usr/local/hadoop/sbin/start-all.sh   //启动所有集群
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [nn01 nn02]
nn01: starting namenode, logging to /usr/local/hadoop/logs/hadoop-root-namenode-nn01.out
nn02: starting namenode, logging to /usr/local/hadoop/logs/hadoop-root-namenode-nn02.out
node2: starting datanode, logging to /usr/local/hadoop/logs/hadoop-root-datanode-node2.out
node3: starting datanode, logging to /usr/local/hadoop/logs/hadoop-root-datanode-node3.out
```

```
node1: starting datanode, logging to /usr/local/hadoop/logs/hadoop-root-datanode-node1.out
Starting journal nodes [node1 node2 node3]
node1: starting journalnode, logging to /usr/local/hadoop/logs/hadoop-root-journalnode-node1.out
node3: starting journalnode, logging to /usr/local/hadoop/logs/hadoop-root-journalnode-node3.out
node2: starting journalnode, logging to /usr/local/hadoop/logs/hadoop-root-journalnode-node2.out
Starting ZK Failover Controllers on NN hosts [nn01 nn02]
nn01: starting zkfc, logging to /usr/local/hadoop/logs/hadoop-root-zkfc-nn01.out
nn02: starting zkfc, logging to /usr/local/hadoop/logs/hadoop-root-zkfc-nn02.out
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-root-resourcemanager-nn01.out
node2: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-node2.out
node1: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-node1.out
node3: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-root-nodemanager-node3.out
```

2）nn02 上面操作

```
[root@nn02 hadoop]# /usr/local/hadoop/sbin/yarn-daemon.sh start resourcemanager
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-root-resourcemanager-nn02.out
```

3）查看集群状态

```
[root@nn01 hadoop]# /usr/local/hadoop/bin/hdfs haadmin -getServiceState nn1
active
[root@nn01 hadoop]# /usr/local/hadoop/bin/hdfs haadmin -getServiceState nn2
standby
[root@nn01 hadoop]# /usr/local/hadoop/bin/yarn rmadmin -getServiceState rm1
active
[root@nn01 hadoop]# /usr/local/hadoop/bin/yarn rmadmin -getServiceState rm2
```

```
    standby
```

4）查看节点是否加入

```
    [root@nn01 hadoop]# /usr/local/hadoop/bin/hdfs dfsadmin -report
    ...
    Live datanodes (3):      //会有三个节点
    ...
    [root@nn01 hadoop]# /usr/local/hadoop/bin/yarn   node   -list
    Total Nodes:3
                 Node-Id                        Node-State           Node-Http-Address
Number-of-Running-Containers
          node2:43307                          RUNNING                       node2:8042
0
          node1:34606                          RUNNING                       node1:8042
0
          node3:36749                          RUNNING                       node3:8042
0
```

步骤三：访问集群

1）查看并创建

```
    [root@nn01 hadoop]# /usr/local/hadoop/bin/hadoop   fs -ls   /
    [root@nn01 hadoop]# /usr/local/hadoop/bin/hadoop   fs -mkdir /aa //创建 aa
    [root@nn01 hadoop]# /usr/local/hadoop/bin/hadoop   fs -ls   /           //再次
查看
    Found 1 items
    drwxr-xr-x   - root supergroup            0 2018-09-11 16:54 /aa
    [root@nn01 hadoop]# /usr/local/hadoop/bin/hadoop   fs -put *.txt /aa
    [root@nn01    hadoop]#    /usr/local/hadoop/bin/hadoop         fs     -ls
hdfs://nsdcluster/aa
    //也可以这样查看
    Found 3 items
    -rw-r--r--       2    root    supergroup    86424    2018-09-11    17:00
hdfs://nsdcluster/aa/LICENSE.txt
    -rw-r--r--       2    root    supergroup    14978    2018-09-11    17:00
hdfs://nsdcluster/aa/NOTICE.txt
    -rw-r--r--       2    root    supergroup    1366    2018-09-11    17:00
hdfs://nsdcluster/aa/README.txt
```

2）验证高可用，关闭 active namenode

```
    [root@nn01       hadoop]#       /usr/local/hadoop/bin/hdfs       haadmin
```

```
-getServiceState nn1
active
[root@nn01 hadoop]# /usr/local/hadoop/sbin/hadoop-daemon.sh stop namenode
stopping namenode
[root@nn01 hadoop]# /usr/local/hadoop/bin/hdfs haadmin -getServiceState nn1
//再次查看会报错
[root@nn01 hadoop]# /usr/local/hadoop/bin/hdfs haadmin -getServiceState nn2
//nn02 由之前的 standby 变为 active
active
[root@nn01 hadoop]# /usr/local/hadoop/bin/yarn rmadmin -getServiceState rm1
active
[root@nn01 hadoop]# /usr/local/hadoop/sbin/yarn-daemon.sh stop resourcemanager
//停止 resourcemanager
[root@nn01 hadoop]# /usr/local/hadoop/bin/yarn rmadmin -getServiceState rm2
active
```

3） 恢复节点

```
[root@nn01 hadoop]# /usr/local/hadoop/sbin/hadoop-daemon.sh start namenode
//启动 namenode
[root@nn01 hadoop]# /usr/local/hadoop/sbin/yarn-daemon.sh start resourcemanager
//启动 resourcemanager
[root@nn01 hadoop]# /usr/local/hadoop/bin/hdfs haadmin -getServiceState nn1
//查看
[root@nn01 hadoop]# /usr/local/hadoop/bin/yarn rmadmin -getServiceState rm1
//查看
```