云计算技术与应用赛项新题型解读

2019 年 5 月

目录

一.OpenStack 超融合部署

（一）构建 Ceph 的分布式存储系统

任务描述

1. 了解 Ceph 文件系统的构架和使用环境。
2. 使用 3 台服务器搭建 1 个 Ceph 文件系统。
3. 实现基于 Ceph 文件系统的集群存储。

任务实施

1.基础环境配置

创建 3 个 CentOS 7 系统虚拟机，并修改 hostname 叫 ceph-node1、ceph-node2 和 ceph-node3。

```
[root@ceph-node1 ~]# hostnamectl set-hostname ceph-node1
[root@ceph-node1 ~]# hostnamectl
   Static hostname: ceph-node1
         Icon name: computer-vm
           Chassis: vm
        Machine ID: dae72fe0cc064eb0b7797f25bfaf69df
           Boot ID: b5805a7f0cce470bac5aeacf294edfc0
    Virtualization: kvm
  Operating System: CentOS Linux 7 (Core)
       CPE OS Name: cpe:/o:centos:centos:7
            Kernel: Linux 3.10.0-229.el7.x86_64
      Architecture: x86_64
```

```
[root@ceph-node2 ~]# hostnamectl set-hostname ceph-node2
[root@ceph-node2 ~]# hostnamectl
   Static hostname: ceph-node2
```

```
                Icon name: computer-vm
                  Chassis: vm
               Machine ID: dae72fe0cc064eb0b7797f25bfaf69df
                  Boot ID: 1e86c3bde22440209176cb6fbb062fe2
        Virtualization: kvm
      Operating System: CentOS Linux 7 (Core)
           CPE OS Name: cpe:/o:centos:centos:7
                Kernel: Linux 3.10.0-229.el7.x86_64
          Architecture: x86_64
```

```
[root@ceph-node3 ~]# hostnamectl set-hostname ceph-node3
[root@ceph-node3 ~]# hostnamectl
     Static hostname: ceph-node3
                Icon name: computer-vm
                  Chassis: vm
               Machine ID: dae72fe0cc064eb0b7797f25bfaf69df
                  Boot ID: 44966891972e48f19d60049ed7c68e27
        Virtualization: kvm
      Operating System: CentOS Linux 7 (Core)
           CPE OS Name: cpe:/o:centos:centos:7
                Kernel: Linux 3.10.0-229.el7.x86_64
          Architecture: x86_64
```

这三台虚拟机各需要有 50G 的空闲硬盘。可以使用 lsblk 命令进行验证。

```
[root@ceph-node1 ~]# lsblk
NAME      MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda       253:0    0   40G  0 disk
└─vda1 253:1       0   40G  0 part /
vdb       253:16   0   50G  0 disk
```

配置 3 台虚拟机的 IP 分别为 192.168.1.101、192.168.1.102 和 192.168.1.103，

为了方便起见，修改三台虚拟机的/etc/hosts 文件，修改主机名地址映射关系。

```
192.168.1.101    ceph-node1
192.168.1.102    ceph-node2
192.168.1.103    ceph-node3
```

在 ceph-node1 节点生成 Root SSH 密钥，并将它复制到 ceph-node2 和 ceph-node3 上。这些虚拟机的 root（用户）的密码都是 000000。在使用 ssh-copy-id 命令时如果要输入该密码。

```
# ssh-keygen
# ssh-copy-id root@ceph-node2
# ssh-copy-id root@ceph-node3
```

复制密钥到 ceph-node2 和 ceph-node3 之后，ceph-node1 就可以无密钥登录到另外两台虚拟机上了。

关闭 3 台虚拟机上的防火墙。

```
# systemctl stop firewalld.service
```

禁用 3 台虚拟机上的 SELinux。

临时禁用。

```
# setenforce 0
```

永久禁用。

```
# vi /etc/selinux/config  把 selinux 改成 disablesd
# reboot                                  //生效需要重启虚拟机
```

在所有虚拟机上安装并配置 NTP 服务和网络的时钟服务器同步时间。

```
# yum install -y ntp ntpdate
# ntpdate pool.ntp.org
# systemctl enable ntpd.service
```

在所有 Ceph 节点上修改 YUM 源，均使用本地 FTP 服务器源。（软件包需要 CentOS-7-x86_64-DVD-1511.iso 和 XianDian-IaaS-v2.2.iso）

在服务器上制作本地源，安装 vsftp 服务

```
# yum install -y vsftpd
```

在配置文件中添加一条语句

```
# yum install -y vsftpd
# vi /etc/vsftpd.conf
anon_root=/opt/
# systemctl restart vsftpd.service
```

将软件包挂在到/opt/目录下

```
# mkdir /opt/centos
# mkdir /opt/iaas
# mount CentOS-7-x86_64-DVD-1511.iso /opt/centos
# mount XianDian-IaaS-v2.2.iso /opt/iaas
```

首先将/etc/yum.repos.d 下面的 repo 文件移走

```
[root@ceph-node1 yum.repos.d]# mv * /media/
[root@ceph-node1 yum.repos.d]# vi local.repo
[root@ceph-node1 yum.repos.d]# cat local.repo
[centos]
name=centos
baseurl=ftp://172.30.14.10/centos
gpgcheck=0
enabled=1
[iaas]
name=iaas
baseurl=ftp://172.30.14.10/iaas/iaas-repo
gpgcheck=0
enabled=1
[root@ceph-node1 yum.repos.d]# scp local.repo root@ceph-node2:/etc/yum.repos.d/
local.repo 100% 151 0.2KB/s 00:00
[root@ceph-node1 yum.repos.d]# scp local.repo root@ceph-node3:/etc/yum.repos.d/
```

```
    local.repo                                                  100%  151
0.2KB/s   00:00
```

2.安装和配置 Ceph

要部署这个集群，需要使用 ceph-deploy 工具在 3 台虚拟机上安装和配置 Ceph。ceph-deploy 是 Ceph 软件定义存储系统的一部分，用来方便地配置和管理 Ceph 存储集群。

（1）创建 Ceph 集群

首先，在 ceph-node1 上安装 Ceph，并配置它为 Ceph monitor 和 OSD 节点。

① 在 ceph-node1 上安装 ceph-deploy。

```
[root@ceph-node1 ~]# yum install ceph-deploy -y
```

② 通过在 cep-node1 上执行以下命令，用 ceph-deploy 创建一个 Ceph 集群。

```
[root@ceph-node1 ~]# mkdir /etc/ceph
[root@ceph-node1 ~]# cd /etc/ceph
[root@ceph-node1 ceph]# ceph-deploy new ceph-node1
```

执行命令成功的截图如下：

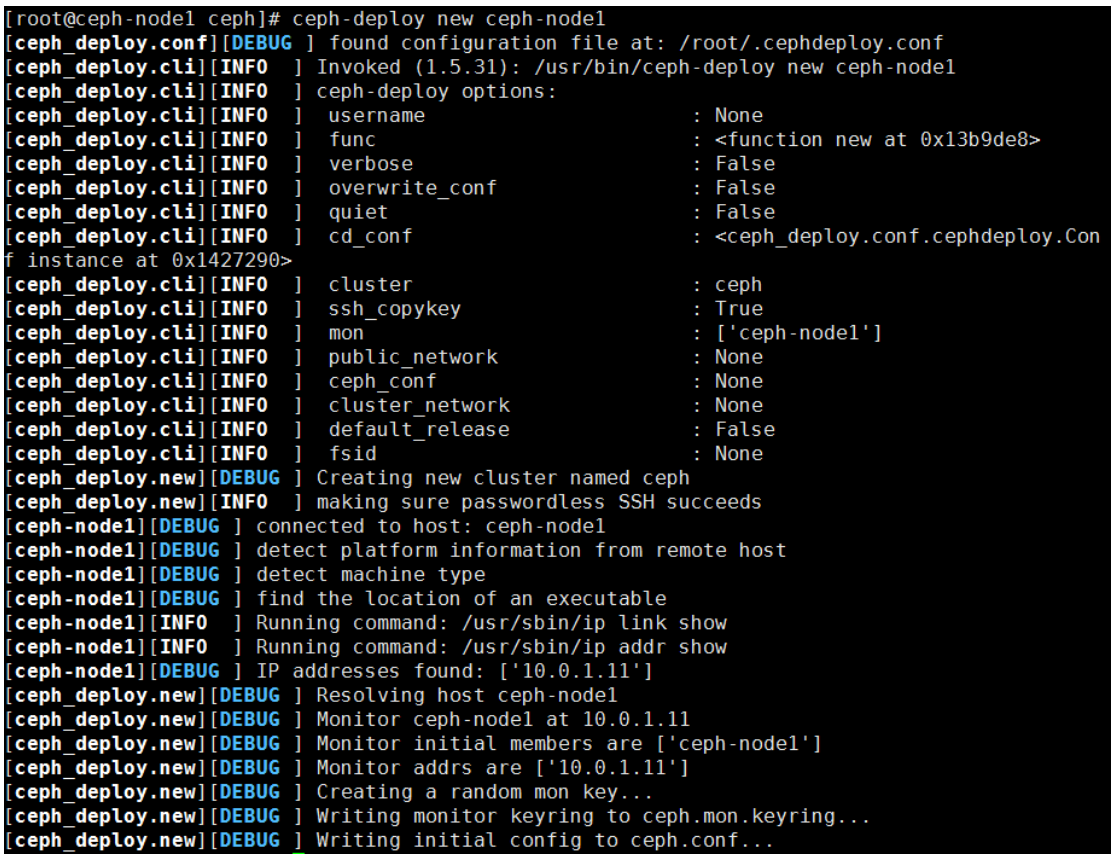


图 3-1-4 创建 Ceph 集群

③ ceph-deploy 的 new 子命令能够部署一个默认名称为 Ceph 的新集群，并且它能生成集群配置文件和密钥文件。列出当前的工作目录，可以查看到 ceph.conf 和 ceph.mon.keying 文件。

```
[root@ceph-node1 ceph]# ll
total 12
-rw-r--r-- 1 root root    229 Sep 20 16:20 ceph.conf
-rw-r--r-- 1 root root 2960 Sep 20 16:20 ceph.log
-rw------- 1 root root      73 Sep 20 16:20 ceph.mon.keyring
```

④ 在 ceph-node1 上执行以下命令，使用 ceph-deploy 工具在所有节点上安装 Ceph 二进制软件包。

```
[root@ceph-node1 ceph]# ceph-deploy install ceph-node1 ceph-node2 ceph-node3
[ceph_deploy.conf][DEBUG ] found configuration file at: /root/.cephdeploy.conf
 [ceph-node1][DEBUG ] connected to host: ceph-node1
```

[ceph-node1][DEBUG] detect platform information from remote host

[ceph-node1][DEBUG] detect machine type

[ceph-node3][DEBUG] Upgrade 2 Packages (+14 Dependent packages)

[ceph-node3][DEBUG]

[ceph-node3][DEBUG] Total download size: 47 M

[ceph-node3][DEBUG] Downloading packages:

[ceph-node3][DEBUG] Delta RPMs disabled because /usr/bin/applydeltarpm not installed.

[ceph-node3][WARNIN] No data was received after 300 seconds, disconnecting...

[ceph-node3][INFO] Running command: ceph --version

[ceph-node3][DEBUG] ceph version 0.94.5 (9764da52395923e0b32908d83a9f7304401fee43)

⑤ **ceph-deploy 工具包首先会安装 Ceph 组件所有的依赖包。命令成功完成后，检查所有节点上 Ceph 的版本信息。**

[root@ceph-node1 ceph]# ceph -v

ceph version 0.94.5 (9764da52395923e0b32908d83a9f7304401fee43)

[root@ceph-node2 ~]# ceph -v

ceph version 0.94.5 (9764da52395923e0b32908d83a9f7304401fee43)

[root@ceph-node3 ~]# ceph -v

ceph version 0.94.5 (9764da52395923e0b32908d83a9f7304401fee43)

⑥ **在 ceph-node1 上创建第一个 Ceph monitor。**

[root@ceph-node1 ceph]# ceph-deploy --overwrite-conf mon create-initial

[ceph_deploy.conf][DEBUG] found configuration file at: /root/.cephdeploy.conf

[ceph_deploy.cli][INFO] Invoked (1.5.31): /usr/bin/ceph-deploy --overwrite-conf mon create-initial

[ceph_deploy.cli][INFO] ceph-deploy options:

```
[ceph_deploy.cli][INFO  ]  username                      : None
[ceph_deploy.cli][INFO  ]  verbose                       : False
[ceph_deploy.cli][INFO  ]  overwrite_conf                : True
[ceph_deploy.cli][INFO  ]  subcommand                    : create-initial
[ceph_deploy.cli][INFO  ]  quiet                         : False
[ceph_deploy.cli][INFO  ]  cd_conf                       : <ceph_deploy.conf.cephdeploy.Conf instance at 0x10bf248>
[ceph_deploy.cli][INFO  ]  cluster                       : ceph
[ceph_deploy.cli][INFO  ]  func                          : <function mon at 0x10b1a28>
[ceph_deploy.cli][INFO  ]  ceph_conf                     : None
[ceph_deploy.cli][INFO  ]  default_release               : False
[ceph_deploy.cli][INFO  ]  keyrings                      : None
[ceph_deploy.mon][DEBUG ] Deploying mon, cluster ceph hosts ceph-node1
[ceph_deploy.mon][DEBUG ] detecting platform for host ceph-node1 ...
[ceph-node1][DEBUG ] connected to host: ceph-node1
[ceph-node1][DEBUG ] detect platform information from remote host
[ceph-node1][DEBUG ] detect machine type
[ceph-node1][DEBUG ] find the location of an executable



[ceph-node1][DEBUG ] connected to host: ceph-node1
[ceph-node1][DEBUG ] detect platform information from remote host
[ceph-node1][DEBUG ] detect machine type
[ceph-node1][DEBUG ] fetch remote file
[ceph_deploy.gatherkeys][DEBUG ] Got ceph.bootstrap-mds.keyring key from ceph-node1.
[ceph_deploy.gatherkeys][DEBUG ] Checking ceph-node1 for
```

```
/var/lib/ceph/bootstrap-rgw/ceph.keyring
    [ceph-node1][DEBUG ] connected to host: ceph-node1
    [ceph-node1][DEBUG ] detect platform information from remote host
    [ceph-node1][DEBUG ] detect machine type
    [ceph-node1][DEBUG ] fetch remote file
    [ceph_deploy.gatherkeys][DEBUG ] Got ceph.bootstrap-rgw.keyring key from ceph-node1.
```

⑦ Monitor 创建成功后，检查集群的状态，这个时候 Ceph 集群并不处于健康状态。

```
    [root@ceph-node1 ceph]# ceph -s
        cluster 418b5790-5c70-44a6-9b74-cba31a1d6a44
         health HEALTH_ERR
                64 pgs stuck inactive
                64 pgs stuck unclean
                no osds
         monmap e1: 1 mons at {ceph-node1=10.0.1.11:6789/0}
                election epoch 2, quorum 0 ceph-node1
         osdmap e1: 0 osds: 0 up, 0 in
          pgmap v2: 64 pgs, 1 pools, 0 bytes data, 0 objects
                0 kB used, 0 kB / 0 kB avail
                      64 creating
```

（2）创建 OSD

① 列出 ceph-node1 上所有的可用磁盘。

```
    [root@ceph-node1 ceph]# ceph-deploy disk list ceph-node1
    [ceph_deploy.conf][DEBUG ] found configuration file at: /root/.cephdeploy.conf
    [ceph_deploy.cli][INFO  ] Invoked (1.5.36): /usr/bin/ceph-deploy disk list ceph-node1
```

```
[ceph_deploy.cli][INFO  ] ceph-deploy options:
[ceph_deploy.cli][INFO  ]  username                       : None
[ceph_deploy.cli][INFO  ]  verbose                        : False
[ceph_deploy.cli][INFO  ]  overwrite_conf                 : False
[ceph_deploy.cli][INFO  ]  subcommand                     : list
[ceph_deploy.cli][INFO  ]  quiet                          : False
[ceph_deploy.cli][INFO  ]  cd_conf                        :
<ceph_deploy.conf.cephdeploy.Conf instance at 0x7f3a3ee9d6c8>
[ceph_deploy.cli][INFO  ]  cluster                        : ceph
[ceph_deploy.cli][INFO  ]  func                           : <function
disk at 0x7f3a3ee8fb90>
[ceph_deploy.cli][INFO  ]  ceph_conf                      : None
[ceph_deploy.cli][INFO  ]  default_release                 : False
[ceph_deploy.cli][INFO  ]  disk                           :
[('ceph-node1', None, None)]
[ceph-node1][DEBUG ] connected to host: ceph-node1
[ceph-node1][DEBUG ] detect platform information from remote host
[ceph-node1][DEBUG ] detect machine type
[ceph-node1][DEBUG ] find the location of an executable
[ceph_deploy.osd][INFO  ] Distro info: CentOS Linux 7.3.1611 Core
[ceph_deploy.osd][DEBUG ] Listing disks on ceph-node1...
[ceph-node1][DEBUG ] find the location of an executable
[ceph-node1][INFO  ] Running command: /usr/sbin/ceph-disk list
[ceph-node1][DEBUG ] /dev/vda :
[ceph-node1][DEBUG ]  /dev/vda1 other, xfs, mounted on /
```

② 创建共享磁盘，3 个节点都要执行。首先对系统上的空闲硬盘进行分区、格式化、挂载的操作。

```
[root@ceph-node1 ceph]# parted /dev/vdb
GNU Parted 3.1
```

```
Using /dev/vdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) p
Model: Virtio Block Device (virtblk)
Disk /dev/vdb: 53.7GB
Sector size (logical/physical): 512B/512B
Partition Table: loop
Disk Flags:

Number  Start  End     Size    File system  Flags
 1      0.00B  53.7GB  53.7GB  fat32

(parted) mklabel
New disk label type? gpt
Warning: The existing disk label on /dev/vdb will be destroyed and all data on this disk
will be lost. Do you want to continue?
Yes/No? yes
(parted) p
Model: Virtio Block Device (virtblk)
Disk /dev/vdb: 53.7GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start  End  Size  File system  Name  Flags

(parted) mkpart
Partition name?  []?
```

```
File system type?  [ext2]?
Start? 0%
End? 100%
(parted) p
Model: Virtio Block Device (virtblk)
Disk /dev/vdb: 53.7GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
Disk Flags:

Number  Start   End     Size    File system  Name  Flags
 1      1049kB  53.7GB  53.7GB

(parted) q
Information: You may need to update /etc/fstab.
```

以上的操作是使用 parted 工具，对/dev/vdb 进行分区，首先使用 mklabel 命令改变该盘的卷标为 gpt，然后使用 mkpart 命令，将所有的空间都分给/dev/vdb1，

可以使用 lsblk 命令，查看分区情况

```
[root@ceph-node1 ceph]# lsblk
NAME      MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda       253:0    0  40G  0 disk
└─vda1 253:1    0  40G  0 part /
vdb       253:16   0  50G  0 disk
└─vdb1 253:17   0  50G  0 part
```

接下来使用命令对分区进行格式化，创建一个目录叫/opt/osd1，挂载分区到目录，最后将该目录的权限提升到 777。

```
[root@ceph-node1 ceph]# mkfs.xfs /dev/vdb1
meta-data=/dev/vdb1              isize=256    agcount=4, agsize=3276672 blks
```

```
                =                       sectsz=512   attr=2, projid32bit=1
                =                       crc=0        finobt=0
      data      =                       bsize=4096   blocks=13106688,
imaxpct=25
                =                       sunit=0      swidth=0 blks
      naming    =version 2              bsize=4096   ascii-ci=0 ftype=0
      log       =internal log           bsize=4096   blocks=6399, version=2
                =                       sectsz=512   sunit=0   blks,
lazy-count=1
      realtime =none                    extsz=4096   blocks=0, rtextents=0
      [root@ceph-node1 ceph]# mkdir /opt/osd1
      [root@ceph-node1 ceph]# mount /dev/vdb1 /opt/osd1
      [root@ceph-node1 ceph]# df -h
      Filesystem      Size  Used Avail Use% Mounted on
      /dev/vda1        40G  1.1G   39G   3% /
      devtmpfs        2.0G     0  2.0G   0% /dev
      tmpfs           2.0G     0  2.0G   0% /dev/shm
      tmpfs           2.0G   17M  2.0G   1% /run
      tmpfs           2.0G     0  2.0G   0% /sys/fs/cgroup
      /dev/vdb1        50G   33M   50G   1% /opt/osd1
      [root@ceph-node1 ceph]# chmod 777 /opt/osd1/
```

在另外两个节点做相同的操作。

```
      [root@ceph-node1 ceph]# ll /opt/
      total 0
      drwxrwxrwx 2 root root 6 Sep 20 17:40 osd1
      [root@ceph-node1 ceph]# lsblk
      NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
      vda    253:0    0  40G  0 disk
       └─vda1 253:1    0  40G  0 part /
```

```
vdb       253:16     0    50G    0 disk
 └─vdb1 253:17      0    50G    0 part /opt/osd1
[root@ceph-node2 ~]# ll /opt/
total 0
drwxrwxrwx 2 root root 6 Sep 20 17:49 osd2
[root@ceph-node2 ~]# lsblk
NAME      MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda       253:0       0    40G    0 disk
 └─vda1 253:1       0    40G    0 part /
vdb       253:16     0    50G    0 disk
 └─vdb1 253:17      0    50G    0 part /opt/osd2
[root@ceph-node3 ~]# ll /opt/
total 0
drwxrwxrwx 2 root root 6 Sep 20 17:51 osd3
[root@ceph-node3 ~]# lsblk
NAME      MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda       253:0       0    40G    0 disk
 └─vda1 253:1       0    40G    0 part /
vdb       253:16     0    50G    0 disk
 └─vdb1 253:17      0    50G    0 part /opt/osd3
```

③ 在 node1 节点使用 ceph-deploy 工具创建 OSD 节点。

```
[root@ceph-node1 ceph]# ceph-deploy osd prepare ceph-node1:/opt/osd1 ceph-node2:/opt/osd2 ceph-node3:/opt/osd3
```

④ 在 node1 节点使用 ceph-deploy 工具激活 OSD 节点。在激活 OSD 节点前，把创建的三个节点上的 osd1、osd2、osd3 目录中的文件权限改为 777。

```
[root@ceph-node1 ceph]# cd /opt/osd1/
[root@ceph-node1 osd1]# ll
```

```
total 12
-rw-r--r-- 1 root root 37 Sep 20 17:52 ceph_fsid
-rw-r--r-- 1 root root 37 Sep 20 17:52 fsid
-rw-r--r-- 1 root root 21 Sep 20 17:52 magic
[root@ceph-node1 osd1]# chmod 777 *
[root@ceph-node1 osd1]# ll
total 12
-rwxrwxrwx 1 root root 37 Sep 20 17:52 ceph_fsid
-rwxrwxrwx 1 root root 37 Sep 20 17:52 fsid
-rwxrwxrwx 1 root root 21 Sep 20 17:52 magic
```

接下来开始激活 osd 节点

```
[root@ceph-node1 ceph]# ceph-deploy osd activate ceph-node1:/opt/osd1 ceph-node2:/opt/osd2 ceph-node3:/opt/osd3
[ceph_deploy.conf][DEBUG ] found configuration file at: /root/.cephdeploy.conf
[ceph_deploy.cli][INFO  ] cluster : ceph
[ceph-node1][WARNIN] DEBUG:ceph-disk:OSD uuid is 6c5cd76e-f68c-48b2-b4b9-97d550970e65
[ceph-node1][WARNIN] DEBUG:ceph-disk:Allocating OSD id...

[ceph-node2][WARNIN] DEBUG:ceph-disk:Cluster uuid is 06b91a6e-8faf-4992-b95c-84d192f30096

[ceph-node3][WARNIN] DEBUG:ceph-disk:OSD uuid is 70b154fa-fa5e-4e78-9d83-bcc18e09b9b6
[ceph-node3][WARNIN] a requirement dependency on it.
[ceph-node3][WARNIN] 3) A unit may be started when needed via activation (socket, path, timer,
[ceph-node3][WARNIN] D-Bus, udev, scripted systemctl call, ...).
```

⑤ 检查 **Ceph** 集群的状态。此时，集群是 **HEALTH_OK** 状态。

```
[root@ceph-node1 ceph]# ceph -s
    cluster 418b5790-5c70-44a6-9b74-cba31a1d6a44
     health HEALTH_OK
     monmap e1: 1 mons at {ceph-node1=10.0.1.11:6789/0}
            election epoch 2, quorum 0 ceph-node1
     osdmap e14: 3 osds: 3 up, 3 in
      pgmap v23: 64 pgs, 1 pools, 0 bytes data, 0 objects
            15459 MB used, 134 GB / 149 GB avail
                  64 active+clean
```

⑥ 开放权限给其他节点，进行灾备处理。

```
# ceph-deploy admin ceph-node{1,2,3}
[ceph_deploy.conf][DEBUG ] found configuration file at: /root/.cephdeploy.conf
[ceph_deploy.cli][INFO  ] Invoked (1.5.31): /usr/bin/ceph-deploy admin ceph-node1 ceph-node2 ceph-node3
[ceph_deploy.admin][DEBUG ] Pushing admin keys and conf to ceph-node1
[ceph_deploy.admin][DEBUG ] Pushing admin keys and conf to ceph-node2
[ceph-node2][DEBUG ] connected to host: ceph-node2
[ceph-node2][DEBUG ] detect platform information from remote host
[ceph_deploy.admin][DEBUG ] Pushing admin keys and conf to ceph-node3
[ceph-node3][DEBUG ] connected to host: ceph-node3
[ceph-node3][DEBUG ] write cluster configuration to /etc/ceph/{cluster}.conf
# chmod +r /etc/ceph/ceph.client.admin.keyring
```

（3）Ceph 集群运维

有了可运行的 Ceph 集群后，现在可以用一些简单的命令来体验 Ceph。

① 检查 **Ceph** 的安装状态。

```
# ceph status
```

② 观察集群的健康状况。

```
# ceph -w
```

③ 检查 Ceph monitor 仲裁状态。

```
# ceph quorum_status --format json-pretty
```

④ 导出 Ceph monitor 信息。

```
# ceph mon dump
```

⑤ 检查集群使用状态。

```
# ceph df
```

⑥ 检查 Ceph Monitor、OSD 和 PG（配置组）状态。

```
# ceph mon stat
# ceph osd stat
# ceph pg stat
```

⑦ 列表 PG。

```
# ceph pg dump
```

⑧ 列表 Ceph 存储池。

```
# ceph osd lspools
```

⑨ 检查 OSD 的 CRUSH。

```
# ceph osd tree
```

⑩ 列表集群的认证密钥。

```
# ceph auth list
```

3.Ceph 的使用

安装 Ceph 客户端

创建一台 centos7 系统的虚拟机，名字叫 ceph-client，并配置好 yum 源。

```
[root@ceph-client ~]# hostnamectl set-hostname ceph-client
```

```
[root@ceph-client ~]# hostnamectl
     Static hostname: ceph-client
           Icon name: computer-vm
             Chassis: vm
          Machine ID: dae72fe0cc064eb0b7797f25bfaf69df
             Boot ID: 3db7ec8e4709477a814c21ae0a1986ec
      Virtualization: kvm
    Operating System: CentOS Linux 7 (Core)
         CPE OS Name: cpe:/o:centos:centos:7
              Kernel: Linux 3.10.0-229.el7.x86_64
        Architecture: x86_64
[root@ceph-client yum.repos.d]# cat local.repo
[centos]
name=centos
baseurl=ftp://172.30.14.10/centos
gpgcheck=0
enabled=1
[iaas]
name=iaas
baseurl=ftp://172.30.14.10/iaas-repo
gpgcheck=0
enabled=1
```

回到 ceph-node1 节点通过 ceph-deploy 把 Ceph 安装到 ceph-client 节点。首先要在/etc/hosts 中，加入 ceph-client 的 IP 和主机名，然后执行命令安装 ceph 客户端。

```
[root@ceph-node1 ceph]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
```

```
10.0.1.11    ceph-node1
10.0.1.12    ceph-node2
10.0.1.13    ceph-node3
10.0.1.14    ceph-client
[root@ceph-node1 ceph]# ceph-deploy install ceph-client
[ceph-client][DEBUG ] Dependency Updated:
[ceph-client][DEBUG ]     cryptsetup-libs.x86_64 0:1.6.7-1.el7
[ceph-client][DEBUG ]
[ceph-client][DEBUG ] Complete!
[ceph-client][INFO  ] Running command: ceph --version
[ceph-client][DEBUG ] ceph version 0.94.5 (9764da52395923e0b32908d83a9f7304401fee43)
```

在 ceph-node1 上，用 ceph-deploy 把 Ceph 配置文件和 ceph.client.admin.keyring 拷贝到 ceph-client 。ceph-deploy 工具会把密钥环复制到 /etc/ceph 目录，要确保此密钥环文件有读权限（如 sudo chmod +r /etc/ceph/ceph.client.admin.keyring ）。

```
[root@ceph-node1 ceph]# ceph-deploy admin ceph-client
[ceph_deploy.admin][DEBUG ] Pushing admin keys and conf to ceph-client
[ceph-client][DEBUG ] connected to host: ceph-client
[ceph-client][DEBUG ] detect platform information from remote host
[ceph-client][DEBUG ] detect machine type
[ceph-client][DEBUG ] write cluster configuration to /etc/ceph/{cluster}.conf
```

回到 ceph-client 节点，在客户端，创建块设备。

```
[root@ceph-client ceph]#rbd create foo --size 4096 -m 10.0.1.11 -k /etc/ceph/ceph.client.admin.keyring
[root@ceph-client ceph]# rbd map foo --name client.admin -m 10.0.1.11 -k /etc/ceph/ceph.client.admin.keyring
/dev/rbd0
```

```
[root@ceph-client ceph]# lsblk
NAME      MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda       253:0    0   40G  0 disk
└─vda1 253:1    0   40G  0 part /
rbd0      252:0    0    4G  0 disk
[root@ceph-client ceph]# mkfs.ext4 /dev/rbd0
mke2fs 1.42.9 (28-Dec-2013)
Discarding device blocks: done
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=1024 blocks, Stripe width=1024 blocks
262144 inodes, 1048576 blocks
52428 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=1073741824
32 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done

[root@ceph-client ceph]# mount /dev/rbd0 /mnt/
```

```
[root@ceph-client ceph]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        40G  1.1G   39G   3% /
devtmpfs        2.0G     0  2.0G   0% /dev
tmpfs           2.0G     0  2.0G   0% /dev/shm
tmpfs           2.0G   17M  2.0G   1% /run
tmpfs           2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/rbd0       3.9G   16M  3.6G   1% /mnt
```

创建块设备的顺序是，在 ceph-client 节点上使用 rbd create 命令创建一个块设备 image，然后用 rbd map 命令把 image 映射为块设备，最后对映射出来的/dev/rbd0 格式化并挂载。就可以当成普通了块设备使用了。

下面学习如何调整块设备的大小。

Ceph 块设备映像是精简配置，只有在你开始写入数据时它们才会占用物理空间。然而，它们都有最大容量，就是你设置的 --size 选项。如果你想增加（或减小） Ceph 块设备映像的最大尺寸，执行下列命令：查看 foo 的大小，调整 foo 的大小，使用 resize2fs 命令调整文件系统的大小，查看。

```
[root@ceph-client ceph]# rbd info foo
rbd image 'foo':
    size 4096 MB in 1024 objects
    order 22 (4096 kB objects)
    block_name_prefix: rb.0.101c.238e1f29
    format: 1
[root@ceph-client ceph]# rbd resize --size 8192 foo
Resizing image: 100% complete...done.
[root@ceph-client ceph]# rbd info foo
rbd image 'foo':
    size 8192 MB in 2048 objects
    order 22 (4096 kB objects)
    block_name_prefix: rb.0.101c.238e1f29
```

```
    format: 1
[root@ceph-client ceph]# resize2fs /dev/rbd0
resize2fs 1.42.9 (28-Dec-2013)
Filesystem at /dev/rbd0 is mounted on /mnt; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 1
The filesystem on /dev/rbd0 is now 2097152 blocks long.

[root@ceph-client ceph]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        40G  1.1G   39G   3% /
devtmpfs        2.0G     0  2.0G   0% /dev
tmpfs           2.0G     0  2.0G   0% /dev/shm
tmpfs           2.0G   17M  2.0G   1% /run
tmpfs           2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/rbd0       7.8G   18M  7.4G   1% /mnt
```

删除块设备，先卸载设备，再删除

```
[root@ceph-client ceph]# umount /mnt/
[root@ceph-client ceph]# rbd rm foo
Removing image: 100% complete...done.
```

（二）构建超融合基础架构

任务描述

1. 配置 OpenStack 的 Ceph 客户端。
2. 配置 OpenStack Glance 服务对接 Ceph。
3. 配置 OpenStack Nova 服务对接 Ceph。

任务实施

1.配置 OpenStack 作为 Ceph 客户端

（1）安装 All-in-one 节点。

安装一台 All-in-one 的 OpenStack 节点作为 Ceph 的客户端。（关于如何安装 All-in-one 的 OpenStack 平台，这边不再赘述）。

这边我们使用一台 All-in-one 的节点名字叫 xiandian。配置 xiandian 节点的 yum 源，使用之前的 ftp 源，如下所示。

```
[root@xiandian ~]# cat /etc/yum.repos.d/local.repo
[centos]
name=centos
baseurl=ftp://172.30.14.10/centos
gpgcheck=0
enabled=1
[iaas]
name=iaas
baseurl=ftp://172.30.14.10/iaas/iaas-repo
gpgcheck=0
enabled=1
```

（2）配置 xiandian 节点为 Ceph 的客户端。

配置 ceph-node1 节点的/etc/hosts 文件，将 xiandian 节点加进去。

```
[root@ceph-node1 ceph]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

10.0.1.11   ceph-node1
10.0.1.12   ceph-node2
10.0.1.13   ceph-node3
10.0.1.14   ceph-client
```

```
10.0.1.15    xiandian
```

在 ceph-node1 节点执行命令，安装 xiandian 节点的客户端。

```
ceph-deploy install xiandian
```

等待命令执行完毕后，执行命令，将配置文件拷贝到 xiandian 节点。

```
ceph-deploy admin xiandian
```

现在 xiandian 节点就成为了 ceph 集群的一个客户端。

（3）配置存储池。

为 Cinder、Glance、Nova 创建 Ceph 存储池。开发者也可以使用任何可用的存储池，这里会创建三个存储池作为 3 种存储的后端存储池，创建完成后可以检查当前的存储池信息。首先看下默认存储池的信息。

```
# ceph osd pool stats
pool rbd id 0
   nothing is going on
```

//创建 images 池，对应 Glance 服务

```
ceph osd pool create images 128
pool 'images' created
```

// 创建 vms 池，对应 Nova 服务

```
ceph osd pool create vms 128
pool 'vms' created
```

//创建 volumes 池，对应 Cinder 服务

```
ceph osd pool create volumes 128
pool 'volumes' created
```

查看创建的存储池。

```
[root@xiandian ~]# ceph osd pool stats
pool rbd id 0
   nothing is going on

pool volumes id 1
```

```
    nothing is going on


pool images id 2
    nothing is going on


pool vms id 3
    nothing is going on
```

（4）创建 Ceph 用户。

为存储池创建认证用户。在 ceph-node1 节点上执行。

```
[root@ceph-node1 ceph]# ceph auth get-or-create client.glance mon 'allow r' osd 'allow class-read object_prefix rbd_children, allow rwx pool=images'
[client.glance]
    key = AQBIVaVbC057GxAAeYLdlvKp2DzjHFyeiA82lg==
```

（5）拷贝 keyring

创建 xiandian 节点的 keyring。在 ceph-node1 节点上执行。

```
[root@ceph-node1 ceph]# ceph auth get-or-create client.glance | ssh xiandian tee /etc/ceph/ceph.client.glance.keyring
[client.glance]
    key = AQBIVaVbC057GxAAeYLdlvKp2DzjHFyeiA82lg==
```

（6）修改权限。

修改 xiandian 节点的 keyring 权限。在 ceph-node1 节点上执行。

```
[root@ceph-node1 ceph]# ssh xiandian chown glance:glance /etc/ceph/ceph.client.glance.keyring
```

2.配置 Glance 服务

现在已经完成了 Ceph 侧所需的配置，接下来通过配置 OpenStack Glance，将 Ceph 用作后端存储，配置 OpenStack Glance 模块来将其虚拟机镜像存储在 Ceph RDB 中。

（1）修改 Glance 配置文件。

登录到 xiandian 节点，然后编辑/etc/glance/glance-api.conf 文件的[DEFAULT]和[glance_store]的配置并做如下修改。

```
# vi /etc/glance/glance-api.conf
[DEFAULT]
rpc_backend = rabbit
show_image_direct_url = True
[glance_store]
#stores = file,http
#file =
#filesystem_store_datadir = /var/lib/glance/images/
stores = rbd
default_store = rbd
rbd_store_pool = images
rbd_store_user = glance
rbd_store_ceph_conf = /etc/ceph/ceph.conf
rbd_store_chunk_size = 8
```

（2）重新启动服务。

重新启动 OpenStack Glance 服务。

```
[root@xiandian ~]# openstack-service restart glance-api
```

（3）检查结果。

① 转换镜像。

要在 Ceph 中启动虚拟机，Glance 镜像的格式必须为 RAW。这里可以利用本教材提供的 cirros-0.3.4-x86_64-disk.img 镜像，将镜像类型从 QCOW2 转换成 RAW 格式。这里也可以使用任何 RAW 格式的其他镜像。

```
# qemu-img convert -p -f qcow2 -O raw cirros-0.3.4-x86_64-disk.img cirros.raw
```

② 上传镜像。

将修改的镜像上传到系统。

```
[root@xiandian ~]# glance image-create --name="CirrOS-ceph" --disk-format=raw --container-format=bare < /root/cirros.raw
+------------------+--------------------------------------------------------------------------------------+
| Property | Value |
+------------------+--------------------------------------------------------------------------------------+
| checksum | 56730d3091a764d5f8b38feeef0bfcef |
| container_format | bare |
| created_at | 2018-09-21T21:53:20Z |
| direct_url | rbd://418b5790-5c70-44a6-9b74-cba31a1d6a44/images/e27ba20e-51e5-4e74-9ba8-90bd7b002aae/snap |
```

```
| disk_format | raw
|
| id | e27ba20e-51e5-4e74-9ba8-90bd7b002aae
|
| min_disk | 0
|
| min_ram | 0
|
| name | CirrOS-ceph
|
| owner | 0ab2dbde4f754b699e22461426cd0774
|
| protected | False
|
| size | 41126400
|
| status | active
|
| tags | []
|
| updated_at | 2018-09-21T21:53:25Z
|
| virtual_size | None
|
| visibility | private
|
+------------------+--------------------------------------------------------------------------
------+
```

③ 在 Ceph 的镜像池中查询镜像。

开发者可以在 Ceph 的镜像池中查询镜像 ID 来验证新添加的镜像。

```
[root@xiandian ~]# rbd ls images
e27ba20e-51e5-4e74-9ba8-90bd7b002aae
```

可以发现存储在 ceph 存储池中的 id 与我们创建的镜像 id 一致。而原本 glance 的默认存储路径中没有镜像，如下所示。

```
[root@xiandian ~]# ll /var/lib/glance/images/
total 0
```

现在已经将 Glance 的默认存储后端配置改为 Ceph，所有上传的 Glance 镜像都将存储在 Ceph 中。

3.配置 Cinder 块存储服务

（1）创建 cinder 认证

在 ceph-node1 节点上执行（nova 使用 cinder 用户，就不单独创建了）

```
[root@ceph-node1 ceph]# ceph auth get-or-create client.cinder mon 'allow r' osd 'allow class-read object_prefix rbd_children, allow rwx pool=volumes, allow rwx pool=vms, allow rx pool=images'
[client.cinder]
    key = AQC+a6VbHREEHhAAZ2Cz/EKHEe3rvUHTa94Jow==
```

（2）拷贝 keyring

在 ceph-node1 节点上执行

```
[root@ceph-node1 ceph]# ceph auth get-or-create client.cinder | ssh xiandian tee /etc/ceph/ceph.client.cinder.keyring
[client.cinder]
    key = AQC+a6VbHREEHhAAZ2Cz/EKHEe3rvUHTa94Jow==
```

（3）修改权限

在 ceph-node1 节点上执行

```
[root@ceph-node1 ceph]# ssh xiandian chown cinder:cinder /etc/ceph/ceph.client.cinder.keyring
```

（4） 在 OpenStack 的计算节点（xiandian 节点）上生成 UUID，定义 secret.xml 文件，设置密钥给 Libvirt，这里在 xiandian 节点上进行操作。

① 使用如下代码生成 UUID。

```
[root@xiandian ceph]# uuidgen
83656c02-e3cc-4e98-9aa0-b22a897bf78d
```

② 创建密钥文件，并将 UUID 设置给它。

```
[root@xiandian ~]# cat > secret.xml <<EOF
<secret ephemeral='no' private='no'>
<uuid>83656c02-e3cc-4e98-9aa0-b22a897bf78d</uuid>
<usage type='ceph'>
<name>client.cinder secret </name>
</usage>
</secret>
EOF
```

③ 定义（define）密钥文件，并保证生成的保密字符串是安全的。在接下来的步骤中需要使用这个保密的字符串值。

```
[root@xiandian ~]# virsh secret-define --file secret.xml
Secret 83656c02-e3cc-4e98-9aa0-b22a897bf78d created
```

④ 在 virsh 里设置好最后一步生成的保密字符串值，创建完成后查看系统的密钥文件。

```
[root@xiandian ~]# ceph auth get-key client.cinder > ./client.cinder.key
[root@xiandian ~]# virsh secret-set-value --secret 83656c02-e3cc-4e98-9aa0-b22a897bf78d --base64 $(cat ./client.cinder.key)
Secret value set
[root@xiandian ~]# virsh secret-list
 UUID                                   Usage
--------------------------------------------------------------------------------
 83656c02-e3cc-4e98-9aa0-b22a897bf78d   ceph client.cinder secret
```

（5）修改配置文件

OpenStack 需要一个驱动和 Ceph 块设备交互。还得指定块设备所在的存储池名。编辑 xiandian 节点上的 /etc/cinder/cinder.conf ，改成如下内容：rbd_secret_uuid 就是上面生成的秘钥值。

```
[DEFAULT]
rpc_backend = rabbit
auth_strategy = keystone
my_ip = 127.0.0.1
#enabled_backends = lvm
enabled_backends = ceph
glance_api_servers = http://xiandian:9292

[ceph]
volume_driver = cinder.volume.drivers.rbd.RBDDriver
rbd_pool = volumes
rbd_ceph_conf = /etc/ceph/ceph.conf
rbd_flatten_volume_from_snapshot = false
rbd_max_clone_depth = 5
```

```
rbd_store_chunk_size = 4
rados_connect_timeout = -1
glance_api_version = 2
rbd_user = cinder
rbd_secret_uuid = 83656c02-e3cc-4e98-9aa0-b22a897bf78d
```

（6）重启服务

```
[root@xiandian ~]# systemctl restart      openstack-cinder-volume.service
```

（7）创建块设备并验证

```
[root@xiandian ~]# cinder create --name ceph-block1 1
+------------------------------+------------------------------------+
|                Property                |                Value
|
+------------------------------+------------------------------------+
|                attachments                |                []
|
|             availability_zone             |                nova
|
|                bootable                |                false
|
|           consistencygroup_id            |                None
|
|                created_at                |          2018-09-22T14:39:29.000000
|
|                description                |                None
|
|                encrypted                |                False
```

```
|
    |                                id                                |
15792bdd-3829-4851-99fc-a1a08e5b906b |
    |                            metadata                              |                                {}
|
    |                         migration_status                         |                              None
|
    |                          multiattach                             |                              False
|
    |                              name                                |                          ceph-block1
|
    |                    os-vol-host-attr:host                         |                              None
|
    | os-vol-mig-status-attr:migstat |                              None                              |
    |    os-vol-mig-status-attr:name_id    |                              None
|
    |    os-vol-tenant-attr:tenant_id    |    0ab2dbde4f754b699e22461426cd0774    |
    |                       replication_status                         |                          disabled
|
    |                              size                                |                                 1
|
    |                          snapshot_id                             |                              None
|
    |                          source_volid                            |                              None
|
    |                              status                              |                          creating
|
    |                          updated_at                              |                              None
|
```

```
| user_id | 53a1cf0ad2924532aa4b7b0750dec282 |
| volume_type | None |
+-------------------------------+----------------------------------+
[root@xiandian ~]# cinder list
+--------------------------------------+-----------+-------------+------+-------------+----------+-------------+
| ID | Status | Name | Size | Volume Type | Bootable | Attached to |
+--------------------------------------+-----------+-------------+------+-------------+----------+-------------+
| 15792bdd-3829-4851-99fc-a1a08e5b906b | available | ceph-block1 | 1 | - | false | |
+--------------------------------------+-----------+-------------+------+-------------+----------+-------------+
[root@xiandian ~]# rbd ls volumes
volume-15792bdd-3829-4851-99fc-a1a08e5b906b
```

4.配置 Nova 服务

其实 nova compute 使用 RBD 有两种功能：

一种是将 cinder volume 挂接给虚拟机；

另一种是从 cinder volume 上启动虚拟机，此时 nova 需要创建一个 RBD image，把 glance image 的内容导入，再交给 libvirt。

这边我们就验证第一种 nova 使用 ceph 的情况。

（1）修改配置文件

修改/etc/nova/nova.conf 配置文件，并重启 nova-compute 服务。

```
[libvirt]
```

```
virt_type = qemu
inject_key = True
rbd_user = cinder
rbd_secret_uuid =83656c02-e3cc-4e98-9aa0-b22a897bf78d
```

（2）创建虚拟机

net-id 可以通过 neutron net-list 查询。

```
[root@xiandian ceph]# nova boot --flavor  m1.tiny --image "CirrOS-ceph" --nic
net-id=bd923693-d9b1-4094-bd5b-22a038c44827  ceph-vm1
+--------------------------------------+----------------------------------------------------+
| Property                             | Value                                              |
+--------------------------------------+----------------------------------------------------+
| OS-DCF:diskConfig                    | MANUAL                                             |
| OS-EXT-AZ:availability_zone          |                                                    |
| OS-EXT-SRV-ATTR:host                 | -                                                  |
| OS-EXT-SRV-ATTR:hostname             | ceph-vm1                                           |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | -                                                  |
| OS-EXT-SRV-ATTR:instance_name        | instance-00000005                                  |
| OS-EXT-SRV-ATTR:kernel_id            |                                                    |
| OS-EXT-SRV-ATTR:launch_index         | 0                                                  |
```

```
| OS-EXT-SRV-ATTR:ramdisk_id                                                |
|
| OS-EXT-SRV-ATTR:reservation_id                | r-67j8gksb
|
| OS-EXT-SRV-ATTR:root_device_name                | -
|
| OS-EXT-SRV-ATTR:user_data                                | -
|
| OS-EXT-STS:power_state                                | 0
|
| OS-EXT-STS:task_state                                | scheduling
|
| OS-EXT-STS:vm_state                                | building
|
| OS-SRV-USG:launched_at                                | -
|
| OS-SRV-USG:terminated_at                                | -
|
| accessIPv4                                                |
|
| accessIPv6                                                |
|
| adminPass                                | BrkmHwKTL39e
|
| config_drive                                                |
|
| created                                | 2018-09-22T15:27:24Z
|
| description                                                | -
```

```
|
| flavor                                  | m1.tiny (1)
|
| hostId                                  |
|
| host_status                             |
|
| id                                      |
090c35c1-9af7-43d2-905c-dcece07f0ed2      |
| image                                   | CirrOS-ceph
(e27ba20e-51e5-4e74-9ba8-90bd7b002aae) |
| key_name                                | -
|
| locked                                  | False
|
| metadata                                | {}
|
| name                                    | ceph-vm1
|
| os-extended-volumes:volumes_attached    | []
|
| progress                                | 0
|
| security_groups                         | default
|
| status                                  | BUILD
|
| tenant_id                               |
0ab2dbde4f754b699e22461426cd0774          |
```

```
| updated                              | 2018-09-22T15:27:26Z
|
| user_id                              |
53a1cf0ad2924532aa4b7b0750dec282     |
+--------------------------------------+--------------------------------------------------+
```

（3）挂载云硬盘

```
[root@xiandian ceph]# nova volume-attach
090c35c1-9af7-43d2-905c-dcece07f0ed2 d0fb7440-47c3-4f16-801c-88df8333afa4
+----------+--------------------------------------+
| Property | Value                                |
+----------+--------------------------------------+
| device   | /dev/vdb                             |
| id       | d0fb7440-47c3-4f16-801c-88df8333afa4 |
| serverId | 090c35c1-9af7-43d2-905c-dcece07f0ed2|
| volumeId | d0fb7440-47c3-4f16-801c-88df8333afa4 |
+----------+--------------------------------------+
```

（4）查看挂载

```
[root@xiandian ceph]# cinder list
+--------------------------------------+--------+-------------+------+-------------+---------
-+--------------------------------------+
|                  ID                  | Status |     Name    | Size |
Volume Type | Bootable |             Attached to              |
+--------------------------------------+--------+-------------+------+-------------+---------
-+--------------------------------------+
| d0fb7440-47c3-4f16-801c-88df8333afa4 | in-use | ceph-block1 |  1   |
-           |  false   | 090c35c1-9af7-43d2-905c-dcece07f0ed2 |
```

```
    +-----------------------------------+--------+------------+------+------------+---------
-+-----------------------------------+
```

挂载成功，此验证可以通过 dashboard 页面上的操作来验证。

二.自定义 Rancher 应用商店

（一）rancher 应用商店增加 Nginx 服务

任务描述

1.创建模板文件

2.搭建本地 git 仓库

3.配置 git 仓库免密访问

4.上传文件至 git 仓库

5.添加应用商店

6.成果验证

任务实施

1.环境介绍

本实验是在 rancher 平台的应用商店中自定义一个 Nginx 应用。

镜像源：CentOS7-1511； XianDian-PaaS-v2.2.iso

两个节点：192.168.233.157 server ； 192.168.233.166 client

Docker Version：1.12.6

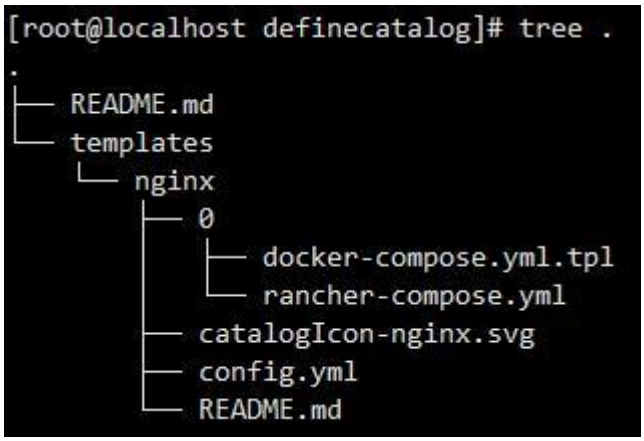
Rancher Version：1.6.5

2.开始

（1）创建模板文件

基本目录层级是这样的：

```
#   cd ./definecatalog
#   tree .
```



说明几个文件的作用：

- README.md 帮助别人快速了解你的项目。
- 0 代表这是我的第一个版本，后续如果有更新，每个版本加 1。
- docker-compose.yml.tpl 和 rancher-compose.yml 是在 rancher 中使用 Docker Compose 启动服务必须提供的两个文件，这两个文件被保存在版本文件夹中（即 0）。
- docker-compose.yml 为一个可以使用 docker-compose up 来启动的文件。该服务 遵循 docker-- compose 格式。
- rancher-compose.yml 将包含帮助你自定义应用模板的其他信息。
- catalogIcon-*.svg 是为应用选择一个好看的 logo。
- config.yml 包含了此应用的一些基本信息。

下面贴出 docker-compose.yml.tpl 和 rancher-compose.yml 文件：

```
# cat docker-compose.yml.tpl
version: '2'
services:
   nginx:
      image: nginx:latest
{{- if ne .Values.db_link ""}}
      external_links:
```

```
            - ${db_link}:db
{{- else}}
        links:
            - db:db
        ports:
        - ${http_port}:80
    db:
        image: mysql:8.0
        environment:
            MYSQL_ROOT_PASSWORD: ${mysql_password}
            MYSQL_USER: ${mysql_user}
            MYSQL_PASSWORD: ${mysql_password}
            MYSQL_DATABASE: ${mysql_db}
        volumes:
            - nginx-db:/var/lib/mysql
{{- end}}
volumes:
    nginx-data:
        driver: ${volume_driver}
{{- if eq .Values.db_link ""}}
    nginx-db:
        driver: ${volume_driver}
{{- end}}


# cat rancher-compose.yml
version: '2'
catalog:
    name: "nginx"
    version: "v0.0.1"
```

```
description: "A web service."
minimum_rancher_version: v0.51.0
questions:
  - variable: http_port
    description: "http port to access nginx"
    label: "Http Port"
    required: true
    default: "9999"
    type: "int"
  - variable: "volume_driver"
    description: "Volume driver to associate with this service"
    label: "Volume Driver"
    required: true
    default: "local"
    type: enum
    options: # List of options if using type of `enum`
      - local
      - rancher-nfs
      - rancher-efs
      - rancher-ebs
  - variable: "db_link"
    description: |
      DB external service link cluster.
    label: "External db service"
    default: ""
    required: false
    type: "service"
  - variable: mysql_db
    description: "mysql db"
```

```
            label: "Mysql db"
            required: true
            default: "nginx"
            type: "string"
        - variable: mysql_user
            description: "mysql user"
            label: "Mysql User"
            required: true
            default: "nginx"
            type: "string"
        - variable: mysql_password
            description: "mysql root password"
            label: "Mysql Password"
            required: true
            default: "default_pass"
            type: "password"
services:
    nginx:
        scale: 1
        retain_ip: true
        health_check:
            response_timeout: 2000
            healthy_threshold: 2
            port: 80
            unhealthy_threshold: 3
            initializing_timeout: 300000
            interval: 2000
            strategy: recreate
            request_line: GET "/" "HTTP/1.0"
```

```
reinitializing_timeout: 120000
```

（2）搭建本地 git 仓库

在 server 节点完成搭建内容。

安装 git

```
# yum -y install git
```

创建用户

```
# useradd git
# passwd git
```

创建本地仓库目录

```
# mkdir /home/git/project/
# cd /home/git/project/
# git init --bare definecatalog.git
Initialized empty Git repository in /home/git/project/definecatalog.git/
# chown -R git:git /home/git/project/
```

（3）配置 git 仓库免密访问

```
# cat /etc/ssh/sshd_config
```

```
#LoginGraceTime 2m
#PermitRootLogin yes
StrictModes no
#MaxAuthTries 6
#MaxSessions 10

RSAAuthentication yes
PubkeyAuthentication yes

# The default is to check both .ssh/authorized_keys and .ssh/authorized_keys2
# but this is overridden so installations will only check .ssh/authorized_keys
AuthorizedKeysFile      /home/git/.ssh/authorized_keys
```

```
# systemctl restart sshd
```

登录 rancher 容器生成密钥对：

```
# docker exec -it e02a22da13a9 /bin/bash
```

root@e02a22da13a9:/# ssh-keygen -t rsa

一路回车即可。

拷贝公钥至 server 节点的 authorized_keys 文件中：

root@e02a22da13a9:/# scp /root/.ssh/id_rsa.pub 192.168.233.157:/home/git/.ssh/authorized_keys

```
root@e02a22da13a9:/# scp /root/.ssh/id_rsa.pub 192.168.233.157:/home/git/.ssh/authorized_keys
root@192.168.233.157's password:
id_rsa.pub                                    100%  399     0.4KB/s   00:00
```

查看：

```
[root@server .ssh]# pwd
/home/git/.ssh
[root@server .ssh]# ll
total 4
-rw-r--r--. 1 root root 399 Mar 27 02:23 authorized_keys
```

在 server 节点修改权限，注意此处必须修改为这样的权限，不然会访问不了：

chmod 700 /home/git/.ssh/

chmod 600 /home/git/.ssh/authorized_keys

```
[root@server ~]# ll -d /home/git/.ssh/
drwx------. 2 root root 28 Mar 27 02:23 /home/git/.ssh/
[root@server ~]# ll /home/git/.ssh/authorized_keys
-rw-------. 1 root root 399 Mar 27 02:23 /home/git/.ssh/authorized_keys
```

（4）上传文件至 git 仓库

将模板文件夹拷贝到本地 git 仓库目录中：

cp -r definecatalog/* /home/git/project/definecatalog.git/

上传文件

cd /home/git/project/definecatalog.git/

git init

git add .

git remote add origin git@192.168.233.157:/home/git/project/definecatalog.git/

git config --global user.email "yormng@yormng.com"

```
# git config --global user.name "yormng"
# git commit -m "first commit"
# git push origin master
```

看到如下字样即为成功：

```
Everything up-to-date
```

（5）添加应用商店

系统管理——系统设置——添加应用商店

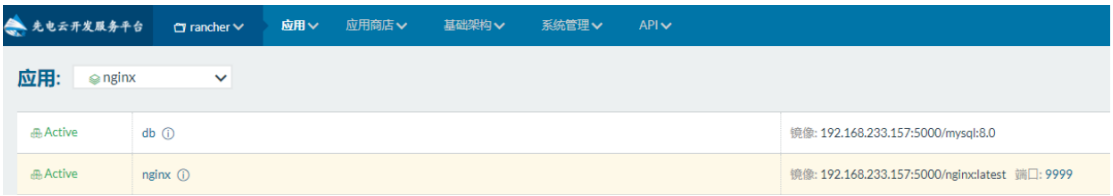
填写如下：

根据提示，地址应该填写一个可以 git clone 操作的 url。

刷新界面，即可看到新添加的应用：

Note：我这里没有给 catalogIcon-*.svg 内容，所以没有 logo 显示出来。

（6）成果验证



可以看到应用里的两个容器正常启动，9999 端口也可以正常访问。

并且数据库容器中已链接到 nginx：

```
[root@client ~]# docker exec -it ef386d610136 /bin/bash
root@ef386d610136:/# mysql -uroot -p000000
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 8.0.3-rc-log MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| nginx              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
```

三.自定义 Ambari 服务

（一）Ambari 服务增加自定义服务

任务描述

1.创建服务目录结构

2.编辑 metainfo.xml

3.编写 Master 组件生命周期回调脚本

4.编写 Slave 组件生命周期回调脚本

5.重启 ambari-server

6.页面添加服务

任务实施

1.环境介绍

知识讲解：Ambari Server 会读取 Stack 和 Service 的配置文件。当用 Ambari 创建服务的时候，Ambari Server 传送 Stack 和 Service 的配置文件以及 Service 生命周期的控制脚本到 Ambari Agent。Agent 拿到配置文件后，会下载安装公共源里软件包（Redhat，就是使用 yum 服务）。安装完成，Ambari Server 会通知 Agent 去启动 Service。之后 Ambari Server 会定期发送命令到 Agent 检查 Service 的状态，Agent 上报给 Server，并呈现在 Ambari 的 GUI 上。

镜像源：centos 7-1511 ； XianDian-BigData-v2.2.iso

两个节点：192.168.233.131 master；192.168.233.132 slave1

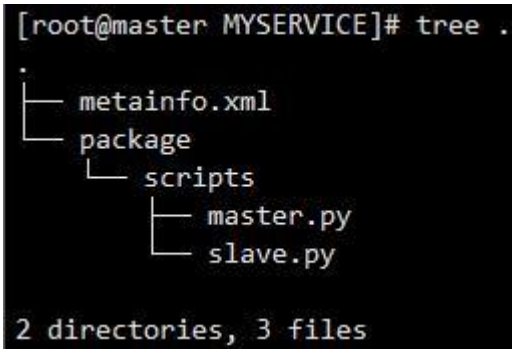
2.开始

在 master 节点完成所有内容。

（1）创建服务目录结构

```
# cd /var/lib/ambari-server/resources/stacks/HDP/2.6/services
# mkdir MYSERVICE
# cd MYSERVICE
```

```
# tree .
```



（2）编辑 metainfo.xml

```
# cat metainfo.xml
<?xml version="1.0"?>
<metainfo>
  <schemaVersion>2.0</schemaVersion>
  <services>
  <service>
  <!--    -->
  <!-- 编写 Service 名称和 Service 信息 -->
  <name>MYSERVICE</name>
  <displayName>My Service</displayName>
  <comment>this is comment</comment>
  <version>1.0</version>
  <components>
  <component>
  <!-- 编写 Master 组件 -->
  <name>MYMASTER</name>
  <displayName>My Master</displayName>
  <category>MASTER</category>
  <cardinality>1</cardinality>
```

```
  <commandScript>
  <script>scripts/master.py</script>
  <scriptType>PYTHON</scriptType>
  <timeout>5000</timeout>
  </commandScript>
  </component>
  <component>
  <!-- 编写 Slave 组件 -->
  <name>MYSALVE</name>
  <displayName>My Slave</displayName>
  <category>SLAVE</category>
  <cardinality>1+</cardinality>
  <commandScript>
  <script>scripts/slave.py</script>
  <scriptType>PYTHON</scriptType>
  <timeout>5000</timeout>
  </commandScript>
  </component>
  </components>
  <osSpecifics>
  <osSpecific>
  <osFamily>any</osFamily>
  </osSpecific>
  </osSpecifics>
  </service>
  </services>
</metainfo>
```

几个标签的说明：

- components 下编写多个组件。
- category 为组件的角色，支持 Master、Slave、和 Client
- cardinality 为节点数量，可以为 1、1+、或数值范围 1-2
- commandScript 为组件生命周期回调的脚本

（3）编写 Master 组件生命周期回调脚本

```
# cat master.py
import sys, os
from resource_management import *
from resource_management.core.exceptions import ComponentIsNotRunning
from resource_management.core.environment import Environment
from resource_management.core.logger import Logger

class Master(Script):
    def install(self, env):
        print "Install My Master"

    def configure(self, env):
        print "Configure My Master"

    def start(self, env):
        print "Start My Master"

    def stop(self, env):
        print "Stop My Master"

    def status(self, env):
        print "Status..."
```

```
if __name__ == "__main__":
Master().execute()
```

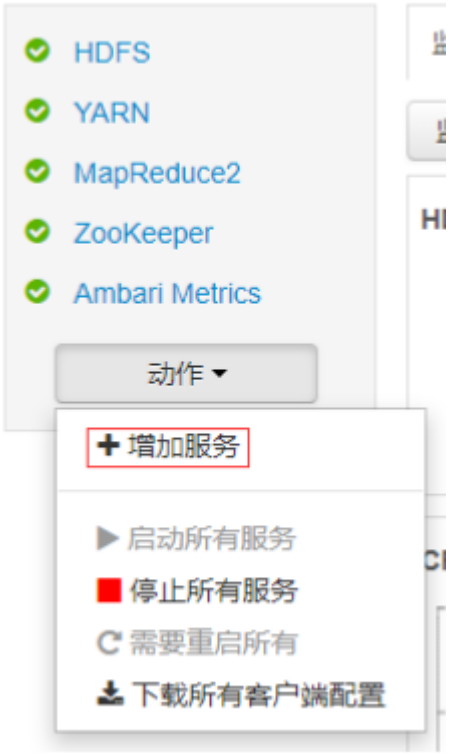
（4）编写 Slave 组件生命周期回调脚本

```
# cat slave.py
import sys, os
from resource_management import *
from resource_management.core.exceptions import ComponentIsNotRunning
from resource_management.core.environment import Environment
from resource_management.core.logger import Logger

class Slave(Script):
    def install(self, env):
        print "Install My Slave"

    def configure(self, env):
        print "Configure My Slave"

    def start(self, env):
        print "Start My Slave"

    def stop(self, env):
        print "Stop My Slave"

    def status(self, env):
        print "Status..."
```

```
if __name__ == "__main__":
    Slave().execute()
```

（5）重启 ambari-server

```
# ambari-server restart
```



验证新添加 My Service 成功。

四.小程序后台服务开发

（一）微信小程序后台接口开发

任务实施

1.商品列表接口示例（在 xueqing-client 工程中开发）

案例界面实现详见《先电云计算软件服务-O2O 商城小程序应用开发手册》手册，手册说明实现商品列表的的 wxml、wxss、js 的文件编写方法。以下说明本功能实现对接数据库的后台服务开发过程，本案例利用 xueqing-client 项目基础之上的参考实现，也可以使用 PC 的开发环境自己构建。

2.建表

Sql 如下；

```
/*
MySQL Data Transfer
Source Host: 192.168.0.108
Source Database: wachat_mini_program
Target Host: 192.168.0.108
Target Database: wachat_mini_program
Date: 2019/5/5 15:41:00
*/

SET FOREIGN_KEY_CHECKS=0;
-- ----------------------------
-- Table structure for commodity
-- ----------------------------
CREATE TABLE `commodity` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `content` varchar(255) DEFAULT NULL,
  `flag` int(11) DEFAULT NULL,
```

```
  `img` varchar(255) DEFAULT NULL,
  `imgs` varchar(255) DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  `price` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=12 DEFAULT CHARSET=utf8;


-- ----------------------------
-- Records
-- ----------------------------
INSERT INTO `commodity` VALUES ('1', 'Android 云存储客户端开发', '0', '/imgs/android-down.jpg', '/imgs/android-down.jpg,/imgs/android-down.jpg', 'Android 云存储客户端开发', '48.00');
INSERT INTO `commodity` VALUES ('2', 'Docker 容器技术与应用', '0', '/imgs/docker-down.jpg', '/imgs/docker-down.jpg,/imgs/docker-down.jpg', 'Docker 容器技术与应用', '60.00');
INSERT INTO `commodity` VALUES ('3', 'Hadoop 大数据平台构建和应用', '0', '/imgs/hadoop-down.jpg', '/imgs/android-down.jpg,/imgs/hadoop-down.jpg', 'Hadoop 大数据平台构建和应用', '75.00');
INSERT INTO `commodity` VALUES ('4', '云存储技术与应用', '0', '/imgs/ycc-down.jpg', '/imgs/ycc-down.jpg,/imgs/ycc-down.jpg', '云存储技术与应用', '40.00');
INSERT INTO `commodity` VALUES ('5', '软件定义网络（SDN）技术与应用', '0', '/imgs/sdn-down.jpg', '/imgs/sdn-down.jpg,/imgs/sdn-down.jpg', '软件定义网络（SDN）技术与应用', '56.00');
```

在数据库添加表 commodity，表结构如下图：

字段	类型	备注
id	BIGINT NOT NULL	主键

content	VARCHAR(255)	简介
img	VARCHAR(255)	图片
name	VARCHAR(255)	名称
price	VARCHAR(255)	价格
imgs	VARCHAR(255)	轮播图片

3.后端接口开发

在 com.xiandian.cloud.entity.core 包下新建 Commodity.java 文件，如下：

```
package com.xiandian.cloud.entity.core;

import javax.persistence.Entity;
import javax.persistence.Table;

import com.xiandian.cloud.entity.BaseEntity;

@Entity
@Table(name = "commodity")
public class Commodity extends BaseEntity {
private String content;
private int flag;
private String img;
private String imgs;
private String name;
private String price;

public String getContent() {
return content;
}
```

```
public void setContent(String content) {
this.content = content;
}

public int getFlag() {
return flag;
}

public void setFlag(int flag) {
this.flag = flag;
}

public String getImg() {
return img;
}

public void setImg(String img) {
this.img = img;
}

public String getImgs() {
return imgs;
}

public void setImgs(String imgs) {
this.imgs = imgs;
}

public String getName() {
```

```
return name;
}

public void setName(String name) {
this.name = name;
}

public String getPrice() {
return price;
}

public void setPrice(String price) {
this.price = price;
}

}
```

在 com.xiandian.cloud.dao.core 包下新建 CommodityDao.java 文件，如下：

```
package com.xiandian.cloud.dao.core;

import java.util.List;

import org.springframework.stereotype.Repository;

import com.xiandian.cloud.dao.BaseDao;
import com.xiandian.cloud.entity.core.Commodity;

@Repository
public class CommodityDao extends BaseDao<Commodity> {
```

```
    public List<Commodity> getAll() {
        String hql = "from Commodity";
        return find(hql);

    }
}
```

在 com.xiandian.cloud.service 包下新建 CommodityService.java 文件，如下：

```
package com.xiandian.cloud.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.xiandian.cloud.dao.core.CommodityDao;
import com.xiandian.cloud.entity.core.Commodity;

@Service
public class CommodityService {
    @Autowired
    private CommodityDao commodityDao;

    public List<Commodity> getAll(){
        return commodityDao.getAll();
    }
}
```

在 com.xiandian.cloud.web 包下新建 WechatController.java 文件，如下：

```
package com.xiandian.cloud.web;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import com.xiandian.cloud.common.bean.MessageBean;
import com.xiandian.cloud.service.CommodityService;

@Controller
@RequestMapping("/wetchat")
public class WechatController {
    @Autowired
    private CommodityService commodityService;

    @RequestMapping("/commodity/list")
    @ResponseBody
    public MessageBean getCommodityList() {
        return new MessageBean(true, "", commodityService.getAll());

    }
}
```

运行结果如下：


```
{"success":true,"msg":"","other":[
{"id":1,"content":"Android 云存储客户端开发","flag":0,"img":"/imgs/android-down.jpg","imgs":"/imgs/android-down.jpg,/imgs/android-down.jpg","name":"Android 云存储客户端开发","price":"48.00"},
```

{"id":2,"content":"Docker 容器技术与应用","flag":0,"img":"/imgs/docker-down.jpg","imgs":"/imgs/docker-down.jpg,/imgs/docker-down.jpg","name":"Docker 容器技术与应用","price":"60.00"},
{"id":3,"content":"Hadoop 大数据平台构建和应用","flag":0,"img":"/imgs/hadoop-down.jpg","imgs":"/imgs/android-down.jpg,/imgs/hadoop-down.jpg","name":"Hadoop 大数据平台构建和应用","price":"75.00"},
{"id":4,"content":" 云存储技术与应用","flag":0,"img":"/imgs/ycc-down.jpg","imgs":"/imgs/ycc-down.jpg,/imgs/ycc-down.jpg","name":"云存储技术与应用","price":"40.00"},
{"id":5,"content":" 软件定义网络（SDN）技术与应用","flag":0,"img":"/imgs/sdn-down.jpg","imgs":"/imgs/sdn-down.jpg,/imgs/sdn-down.jpg","name":"软件定义网络（SDN）技术与应用","price":"56.00"},
{"id":6,"content":"JavaWeb 云应用开发","flag":0,"img":"/imgs/javaweb-down.jpg","imgs":"/imgs/javaweb-down.jpg,/imgs/javaweb-down.jpg","name":"JavaWeb 云应用开发","price":"92.00"},
{"id":7,"content":" 云计算综合运维管理","flag":0,"img":"/imgs/yjszhywgl-down.jpg","imgs":"/imgs/yjszhywgl-down.jpg,/imgs/yjszhywgl-down.jpg","name":"云计算综合运维管理","price":"51.00"},
{"id":8,"content":" 视频直播 APP 应用开发","flag":0,"img":"/imgs/videoapp.jpg","imgs":"/imgs/videoapp.jpg,/imgs/videoapp.jpg","name":"视频直播 APP 应用开发","price":"45.00"},
{"id":9,"content":" 微信小程序应用开发","flag":0,"img":"/imgs/wechat-down.jpg","imgs":"/imgs/wechat-down.jpg,/imgs/wechat-down.jpg","name":"微信小程序应用开发","price":"26.00"},
{"id":10,"content":" 人脸识别和美化应用开发","flag":0,"img":"/imgs/rlsb.jpg","imgs":"/imgs/rlsb.jpg,/imgs/rlsb.jpg","name":" 人脸识别和美化应用开发","price":"29.00"},
{"id":11,"content":" 大数据分析应用开发","flag":0,"img":"/imgs/dsjfx.jpg","imgs":"/imgs/dsjfx.jpg,/imgs/dsjfx.jpg","name":"大

```
数据分析应用开发","price":"120.00"}]}
```

4.小程序端

数据请求，在 list.js 代码如下：

```
const app = getApp()
Page({
    data:{
      url:'',
        goods:[],
      imgUrls: [
        {
          url: '/res/manager/tabs/bg1.jpg'
        }, {
          url: '/res/manager/tabs/bg2.jpg'
        }
      ],
      indicatorDots: true,
      autoplay: true,
      interval: 5000,
      duration: 1000,
      searchValue:'',
      isLogin:false
    },

    // 链接到详情页
    detail:function(e){
        var data = e.target.dataset;
        wx.navigateTo({ url: '/pages/store/products/detail/detail?id=' + data.id });
    },
```

```
// 链接到搜索页
search: function (e) {
  var datas = this.data.searchValue;
  wx.navigateTo({ url: '/pages/store/products/search/search?key=' + datas });
},

inputeidt: function (e) {
  console.log(e)
  this.setData({
    searchValue: e.detail.value,
  })
},


// 获取商品列表
getlist: function (e) {
  var that = this
  wx.request({
    url: app.globalData.url + "/wechat/commodity/list",
    method: "post",
    success: function (res) {
      console.log(res)
      if (res.data. success== true) {
        let list = res.data.other   //此处 res.data 为后端返回值，前端可以此处进行数据的处理
        that.setData({
          goods: list,
        })
```

```
                    console.log(that.data.goods[0].corver)
                }
                else {
                    wx.showToast({
                        title: '加载失败',
                        icon: 'loading'
                    })
                }
            }
        })
    },

    onShow: function () {
        this.getlist()
        this.setData({
            url: app.globalData.url,
            isLogin: app.globalData.hasLogin,
        })
        console.log(this.data.isLogin)
    },
    onHide: function () {
        console.log("onHide")
    },
});
```

app.js 添加如下代码：

```
App({
  onLaunch: function () {
    console.log('App Launch')
  },
```

```
  onShow: function () {
    console.log('App Show')
  },
  onHide: function () {
    console.log('App Hide')
  },
  globalData: {
    hasLogin: false,
    url: 'http://127.0.0.1:8080',
    userId:'',
    userName:'',
    homeaddress:'',
    companyaddress:'',
    Name:'',
    balance:'0',
    cardnumber:'',
    phone:''
  },

})
```

效果如下图：


O2O商城
Docker容器技术与应用
VR世界
虚拟现实体验馆
VR
极致体验
视不可挡
Android云存储客户端开发
￥48.00
查看
自营
Docker容器技术与应用
￥60.00
查看
自营
Hadoop大数据平台构建和应用
￥75.00
查看
自营
云存储技术与应用
商店
售后
订单
账户