第四次作业

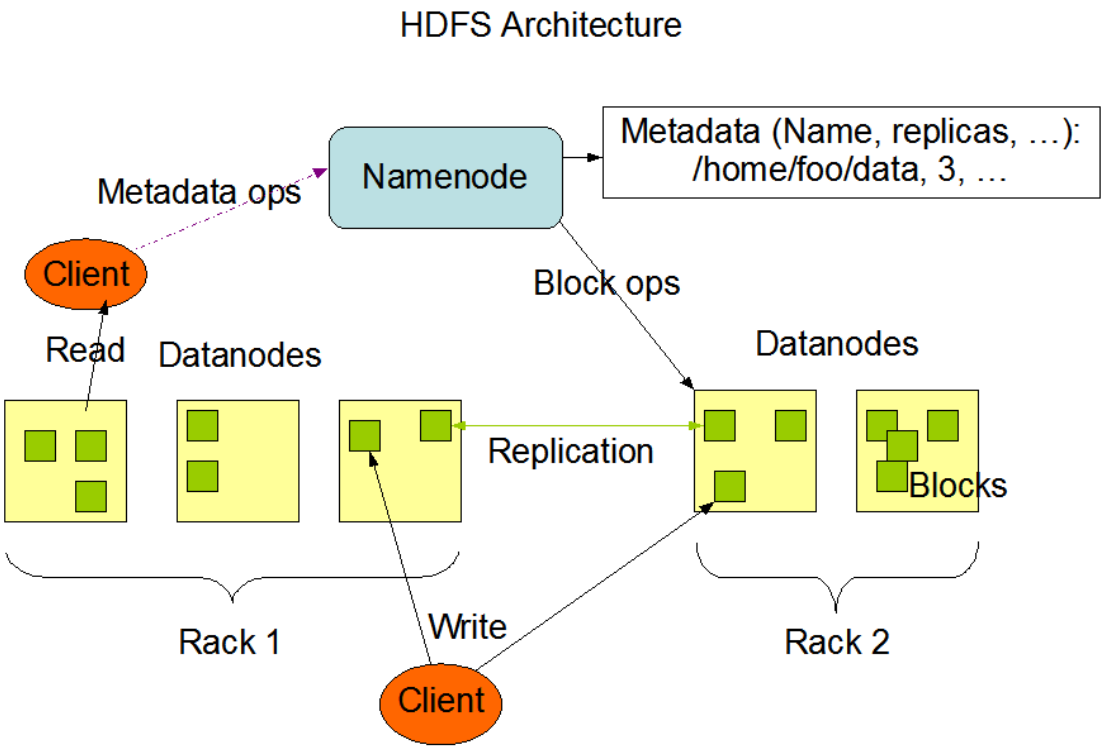
学号：1300013022

姓名：武守北

一、调研两种以上的分布式文件系统以及一种联合文件系统，说明其工作原理和特点以及使用方式。

一、HDFS：

HDFS 工作原理：



HDFS 采用 master/slave 架构。一个 HDFS 集群是由一个 Namenode 和一定数目的 Datanodes 组成。

Namenode 是一个中心服务器，负责管理文件系统的名字空间(namespace)以及客户端对文件的访问。

集群中的 Datanode 一般是一个节点一个，负责管理它所在节点

上的存储。

Namenode 全权管理数据块的复制，它周期性地从集群中的每个 Datanode 接收心跳信号和块状态报告(Blockreport)。接收到心跳信号意味着该 Datanode 节点工作正常。块状态报告包含了一个该 Datanode 上所有数据块的列表。

Namenode 和 Datanode 之间是通过 TCP/IP 协议交互。

HDFS 主要工作流程：

数据备份：

HDFS 的文件块会创建若干份副本来保证容错性，主结点维护所有块的副本信息。数据结点会定期的向主结点汇报其所有数据块的信息。文件副本的分布位置直接影响着 HDFS 的可靠性和性能。一个大型的 HDFS 文件系统一般都是需要跨很多机架的，不同机架之间的数据传输需要经过网关，并且，同一个机架中机器之间的带宽要大于不同机架机器之间的带宽。如果把所有的副本都放在不同的机架中，这样既可以防止机架失败导致数据块不可用，又可以在读数据时利用到多个机架的带宽，并且也可以很容易的实现负载均衡。但是，如果是写数据，各个数据块需要同步到不同的机架，会影响到写数据的效率。

而在 Hadoop 中，如果副本数量是 3 的情况下，把第一个副本放到机架的一个结点上，另一个副本放到同一个机架的另一个结点上，把最后一个结点放到不同的机架上。这种策略减少了跨机架副本的个数提高了写的性能，也能够允许一个机架失败的情况，算是一个很好

的权衡。

读文件：

客户端通过调用这个实例的 open 方法就可以打开系统中希望读取的文件。HDFS 通过调用 NameNode 获取文件块的位置信息，对于文件的每一个块，NameNode 会返回含有该块副本的 DataNode 的结点地址，另外，客户端还会根据网络拓扑来确定它与每一个 DataNode 的位置信息，从离它最近的那个 DataNode 获取数据块的副本，最理想的情况是数据块就存储在客户端所在的结点上。

客户端发起读请求：

客户端与 NameNode 得到文件的块及位置信息列表。

客户端直接和 DataNode 交互读取数据。

读取完成关闭连接。

写文件：

客户端通过调用这个实例的 create 方法就可以创建文件。HDFS 会发送给 NameNode 一个远程过程调用，在文件系统的命名空间创建一个新文件，在创建文件前 NameNode 会做一些检查，如文件是否存在，客户端是否有创建权限等，若检查通过，NameNode 会为创建文件写一条记录到本地磁盘的 EditLog，若不通过会向客户端抛出 IOException。创建成功之后 HDFS 会返回一个 FSDataOutputStream

对象，客户端由此开始写入数据。

客户端在向 NameNode 请求之前先写入文件数据到本地文件系统的一个临时文件。

待临时文件达到块大小时开始向 NameNode 请求 DataNode 信息。

NameNode 在文件系统中创建文件并返回给客户端一个数据块及其对应 DataNode 的地址列表（列表中包含副本存放的地址）。

客户端只需通过上一步得到的信息把创建的临时文件块更新到列表中的第一个 DataNode；第一个 DataNode 在把数据块写入到磁盘中时会继续向下一个 DataNode 发送信息，以此类推，直到全部完成。

当文件关闭，NameNode 会提交这次文件创建，此时，文件在文件系统中可见。

删除文件：

一开始删除文件，NameNode 只是重命名被删除的文件到/trash 目录，因为重命名操作只是元信息的变动，所以整个过程非常快。在/trash 中文件会被保留一定间隔的时间（可配置，默认是 6 小时），在这期间，文件可以很容易的恢复，恢复只需要将文件从/trash 移出即可。

当指定的时间到达，NameNode 将会把文件从命名空间中删除。

标记删除的文件块释放空间，HDFS 文件系统显示空间增加。

HDFS 特点：

1、高容错性

数据自动保存多个副本

副本丢失后，自动恢复

2、适合批处理

移动计算而非数据

数据位置暴露给计算框架

3、适合大数据处理

GB、TB、甚至 PB 级数据

百万规模以上的文件数量

10K+节点规模

4、流式文件访问

适合一次性写入，多次读取

保证数据一致性

5、可构建在廉价机器上

通过多副本提高可靠性

提供了容错和恢复机制

HDFS 使用方式：

1、命令行接口；

2、J ava API

HDFS 使用 Java 编写的的，所以通过 Java API 可以调用所有的 HDFS 的交互操作接口，最常用的是 FileSystem 类，它也是命令行 hadoop fs 的实现。

3、Web UI

还可以通过 NameNode 的 50070 端口号访问 HDFS 的 Web UI。

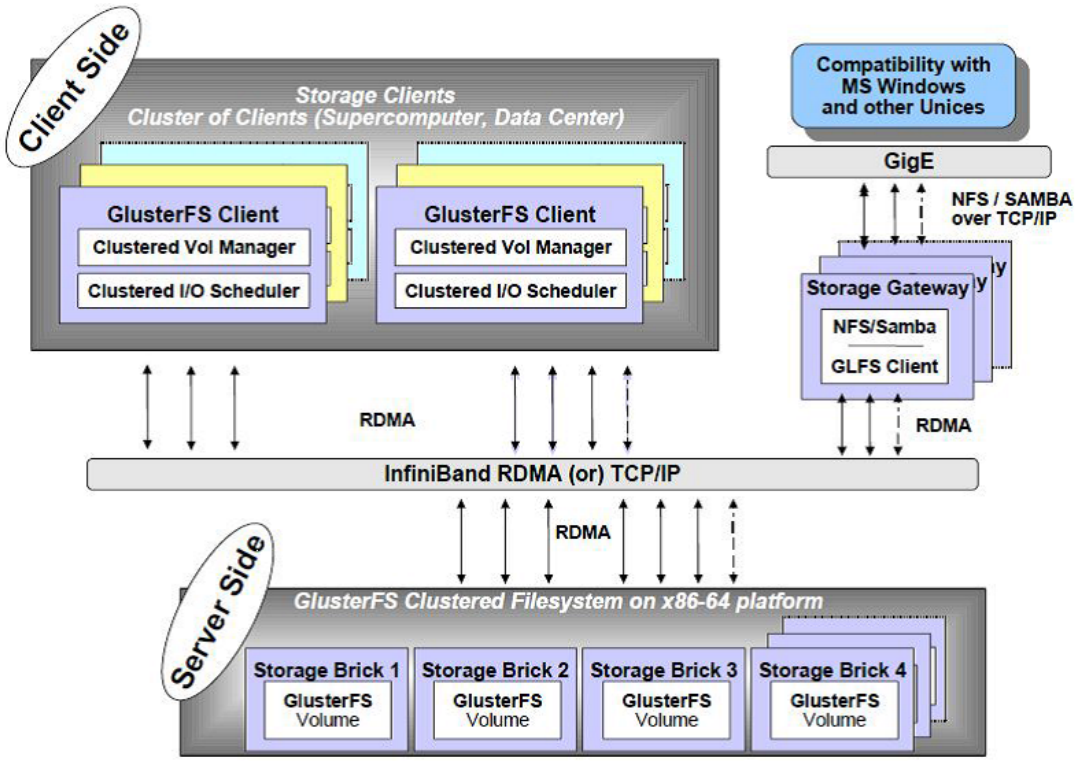
直接在浏览器中输入 master:9000（即 NameNode 的主机名:端口号）便可进入 Web UI。点击“Browse the filesystem”可以查看整个 HDFS 的目录树，点击“Namenode Logs”可以查看所有的 NameNode 的日志，这对于排查错误十分有帮助。

二、GlusterFS：

GlusterFS 是大规模网络分布式文件系统，适合云存储和媒体流处理等数据密集型任务；

1、工作原理：

外部架构：



它主要由存储服务器（BrickServer）、客户端以及 NFS/Samba 存储网关组成。

内部架构：

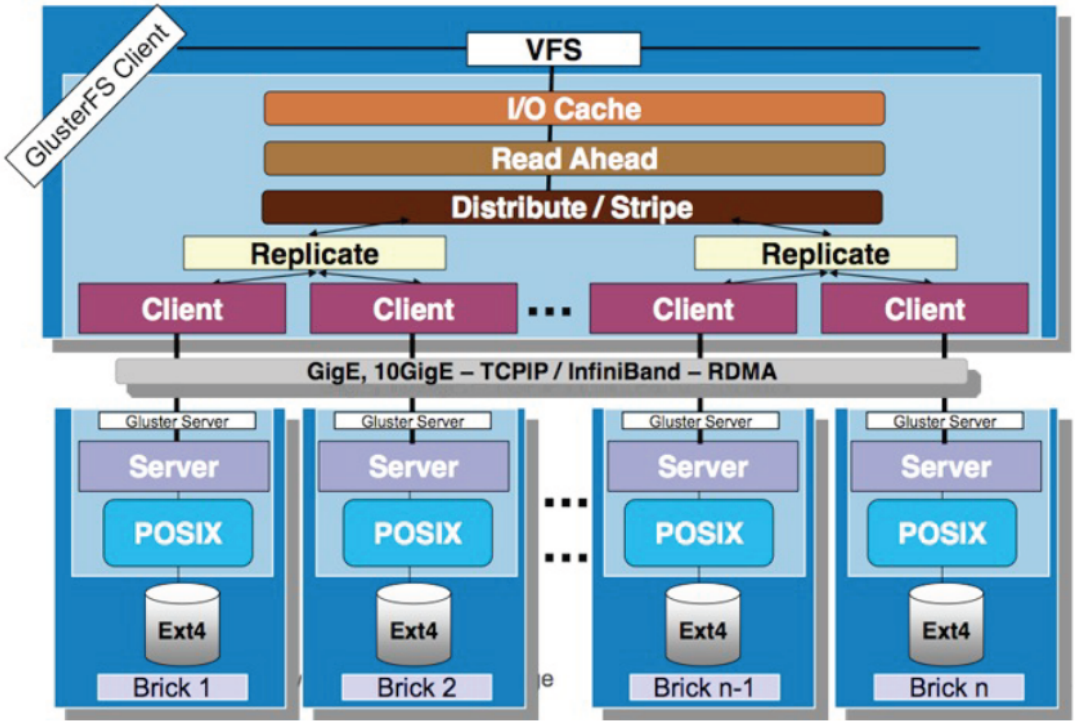
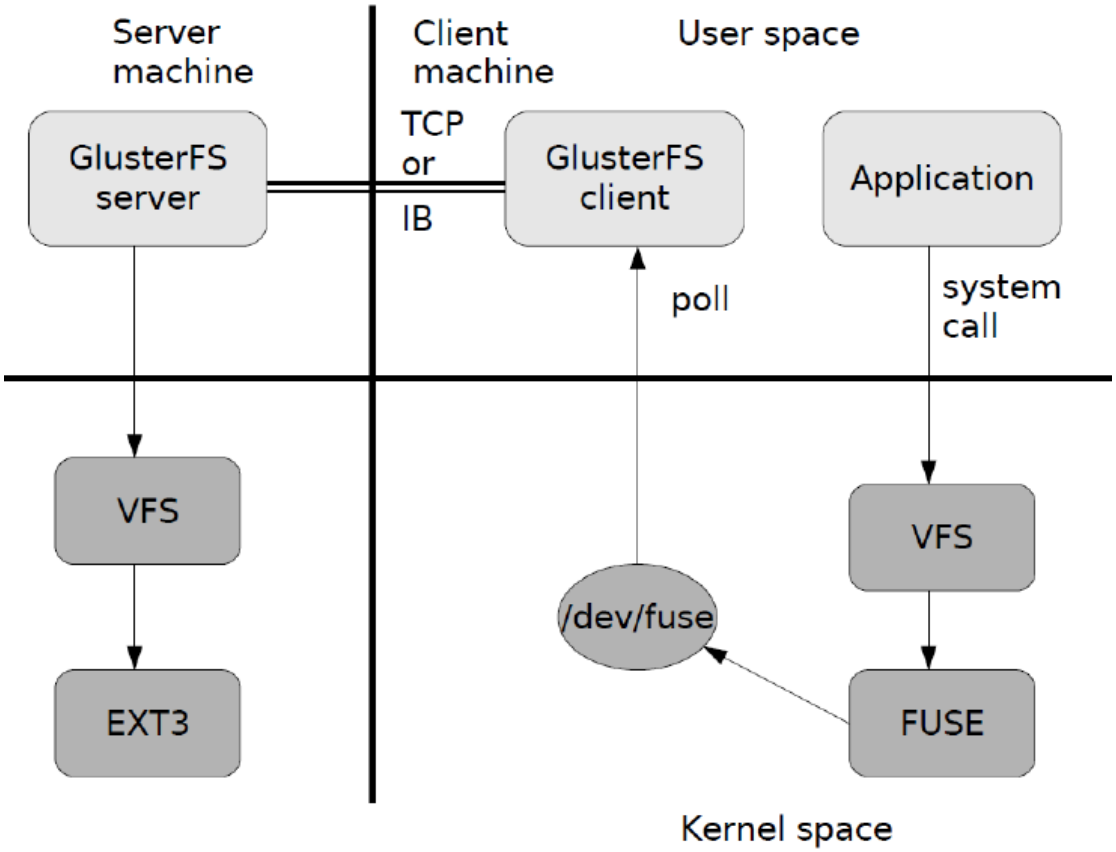


图3 GlusterFS模块化堆栈式设计

GlusterFS 是模块化堆栈式的架构设计，如上图所示。模块称为 Translator，是 GlusterFS 提供的一种强大机制，借助这种良好定义的接口可以高效简便地扩展文件系统的功能

工作流程图：



1、在服务器端，每台服务器上运行 GlusterFs，将本地文件系统的一部分暴露给用户。

2、在客户端，用户通过 GlusterFS 的 mount point 操作来读写数据， 对于用户来说，集群系统的存在对用户是完全透明的，用户感觉不到是操作本地系统还是远端的集群系统。

3、用户的操作请求被递交给本地 Linux 系统的 VFS 来处理。

4、VFS 将数据递交给 FUSE 内核文件系统。

5、client 对数据进行一些指定的处理。

6、在 client 的处理末端，通过网络将数据递交给 server，并将数据写入到服务器所控制的存储设备上。

使用方式：

GlusterFS 服务端安装：

```
yum -y install glusterfs glusterfs-server

chkconfig glusterd on

service glusterd start
```

基本操作：

```
$ gluster peer probe server2 # add server2 to the storage pool
$ gluster volume create gv0 replica 2 server1:/data/brick1/gv0
server2:/data/brick1/gv0 # create the copying volume
$ gluster volume start gv0 # start the copying volume
$ mount -t glusterfs server1:/gv0 /mnt # mount the colume in
clients
$ gluster volume info # see the information of a volume
$ gluster peer status # see the nodes' information
```

三、AUFS（联合文件系统）

AUFS 是一种联合文件系统（UnionFS），联合文件系统会将不同物理位置的目录合并 mount 到同一个目录中。默认写在前边的上层目录是 read-write 的，后边的下层目录是 read-only 的。

1、工作原理：

AUFS 在合并目录时有一系列的规则：

显然目录中重复文件名的文件只保留一个呈现在最终的目录中，且只会呈现最上层的文件。

默认情况下，修改在原上层的可读写目录中的文件会直接修改掉原文件。

默认情况下，修改在原下层的只读目录中的文件会在原上层目录

中新创建一个修改后的文件，原下层目录中的文件不会改变。

若手动将所有的合并目录更改为可读写，修改重复文件名的文件也只会影响到最上层的文件。

删除文件时，若删除的是原最高层目录中的文件，则会删掉原最高层目录中的文件，并且在原最高层目录中打上.wh.*的隐藏文件标志表示屏蔽下层目录中的该文件。

删除文件时，若删除的是下层只读的文件，则直接会在原最高层目录打上.wh.*的隐藏文件标志表示屏蔽下层目录中的该文件。

若删除一个文件夹后创建一个同名文件夹，若该文件夹在下层也有，则在原最高层目录新文件夹里加入原有的所有文件的隐藏文件标志。

文件在原来的地方被修改时，可以通过设置 udba 的参数 none、reval、notify 来不更新、总是更新、通过消息发送更新。

union 的目录（分支）的相关权限：rw 表示可写可读 read-write。ro 表示 read-only，如果你不指权限，那么除了第一个外 ro 是默认值，对于 ro 分支，其永远不会收到写操作，也不会收到查找 whiteout 的操作。rr 表示 real-read-only，与 read-only 不同的是，rr 标记的是天生就是只读的分支，这样，AUFS 可以提高性能，比如不再设置 inotify 来检查文件变动通知。

10、若手动将所有的合并目录更改为可读写，在挂在是可以通过设置 create 选项来指定新创建文件的保存的目录，如轮转、可用空间最优或二者结合等。

2、特点：

节省存储空间：多个 container 可以共享 base image 存储。

快速部署：如果要部署多个 container，base image 可以避免多次拷贝。

内存更省：因为多个 container 共享 base image，以及 OS 的 disk 缓存机制，多个 container 中的进程命中缓存内容的几率大大增加。

升级更方便：相比于 copy-on-write 类型的 FS，base image 也是可以挂载为可写的，可以通过更新 baseimage 而一次性更新其之上的 container。

允许在不更改 base-image 的同时修改其目录中的文件，所有写操作都发生在最上层的可写层中，这样可以大大增加 base image 能共享的文件内容。

3、使用方式：

```
mount -t aufs -o br=(upper)=rw:(base)=ro+wh none (rootfs)
```

可以实现挂载。

二、安装配置一种分布式文件系统，要求启动容错机制，即一台存储节点挂掉仍然能正常工作。在报告里阐述搭建过程和结果。

1、在 1000、1001 上安装 GlusterFS 服务器版本，切换到 root 模式：

```
add-apt-repository ppa:gluster/glusterfs-3.10
apt update
apt install glusterfs-server
```

2、在 1002 上安装 GlusterFS 客户端版本，切换到 root 模式：

```
add-apt-repository ppa:gluster/glusterfs-3.10
apt update
apt install glusterfs-client
```

3、修改 1000 的 Host 文件：

```
127.0.0.1       server1
127.0.1.1       oo-lab.cs1cloud.internal        oo-lab
172.16.6.244     server2
```

4、修改 1001 的 Host 文件

```
127.0.0.1       server2
127.0.1.1       oo-lab.cs1cloud.internal        oo-lab
172.16.6.271    server1
```

其中 server1 对应的 IP 需要在 1000 上通过 ifconfig 指令查看

5、在 1000 和 1001 上创建 brick：

```
mkdir -p /data/brick1
```

6、在 1000（也可以在 1001）创建复制卷 test_volume，并启动该卷：

```
gluster volume create test_volume replica 2

server1:/data/brick1 server2:/data/brick1 force

gluster volume start test_volume

gluster volume info
```

查看该卷的信息，表明创建成功：

```
Volume Name: test_volume
Type: Replicate
Volume ID: f9e54e0b-0f0b-4869-b473-27cd0c2e8a51
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 2 = 2
Transport-type: tcp
Bricks:
Brick1: server1:/data/brick1
Brick2: server2:/data/brick1
Options Reconfigured:
transport.address-family: inet
nfs.disable: on
```

7、修改 1002 上的 hosts 文件：

```
127.0.0.1       localhost

127.0.1.1       oo-lab.cs1cloud.internal        oo-lab

172.16.6.244    server1

172.16.6.271    server2
```

8、在 1002 上创建挂载点，挂载 test_volume 卷

```
mkdir -p /storage

mount -t glusterfs server1:/test_volume /storage
```

向/storage 里存入 50 个文件，为 hw4.txt 的拷贝

```
for i in `seq -w 1 10`; do cp -rp

/home/pkusei/test.txt /storage/copy-test-$i; done
```

检查客户端的存入情况：

```
cd /storage
ls
copy-test-01  copy-test-02  copy-test-03  copy-test-04
copy-test-05
copy-test-06  copy-test-07  copy-test-08  copy-test-09
copy-test-10
```

检查服务器端的存入情况：

```
copy-test-01  copy-test-02  copy-test-03  copy-test-04
copy-test-05
copy-test-06  copy-test-07  copy-test-08  copy-test-09
copy-test-10
```

由于创建卷的方式为复制卷且副本个数为 2，故当一个服务器挂掉时，另一个服务器还保存有副本，具有容错机制。

三、将 web 服务器的主页提前写好在分布式文件系统里，在 docker 容器启动的时候将分布式文件系统挂载到容器里，并将该主页拷贝到正确的路径下，使得访问网站时显示这个主页。

1、在 1000 和 1001 上分别新建一个 brick 来存储主页内容：

```
mkdir -p /data/brick2
```

2、在 1001（也可以在 1000）创建复制卷 homepage，并启动该卷：

```
gluster volume create homepage replica 2
server1:/data/brick2 server2:/data/brick2 force
gluster volume start homepage
```

3、在 1002 上创建挂载点，挂载 homepage 卷

```
mkdir -p /html
mount -t glusterfs server2:/homepage /html
```

存入主页文件

```
vim /html/index.nginx-debian.html
```

4、在 1002 上创建容器并运行 nginx

```
docker run -it --name hw4_docker  ubuntu_docker2
/bin/bash
```

5、进入新创建的容器后，修改/etc/nginx/sites-enabled/default 中 root 的路径为/html

```
vim /etc/nginx/sites-enabled/default
```

6、修改完成后退出容器，将该容器保存为 ubuntu_docker_hw4 镜像

```
docker commit hw4_docker ubuntu_docker_hw4
```

7、将该镜像从 1001 传送到 1002 上：

```
docker save -o save.tar ubuntu_docker_hw4
scp -P 1002 save.tar pkusei@162.105.174.40:
```

8、在 1002 中以 ubuntu_docker_hw4 镜像创建后台容器并运行 nginx，将/html 挂载到容器中的/html 中，将容器的 80 端口映射到宿主机的 4040 端口：

```
docker load < save.tar
docker run -v /html:/html -p 4040:80 -d --name hw4 \
ubuntu_docker_hw4 nginx -g 'daemon off;'
```

四、Docker 中使用了联合文件系统来提供镜像服务，了解 docker 的镜像机制，并仿照其工作机制完成一次镜像的制作，具体要求为:创建一个 docker 容器，找到其文件系统位置，并将其保存，然后在该容器中安装几个软件包，找到容器的只读层（保存着这几个软件包），将其保存，之后通过 aufs 将这两个保存的文件系统挂载起来，使用 docker import 命令从其中创建镜像，并从该镜像中创建容器，要求此时可以使用之前安装好的软件包。在报告中详细阐述过程。

docker 镜像具有分层结构，包括 rootfs 和 bootfs 两个文件系统。

rootfs 主要包括 Linux 系统中的 /dev、/proc、/bin 等常规的目录和文件。

bootfs 主要包括 bootloader 和 kernel，其中 bootloader 的主要功能是引导和加载 kernel。

镜像中每个层结构的文件都是静态的，在对镜像进行修改时，修

改操作会被存在一层之中。同时，不同的镜像可以共享底层的数据，当一个镜像启动新容器时，镜像会在其层结构上面再加一层，并令容器在这一层进行读写操作。

制作镜像：

1、利用 Ubuntu 镜像创建容器并启动

```
docker create -it --name hw4_container ubuntu
/bin/bash
docker start -i hw4_container
```

2、在新的终端内查看系统中各个容器的挂载记录

```
sudo su
df -hT
```

3、对文件系统标识为 none 的记录，查看其文件系统结构

```
cd /var/lib/docker/aufs/layers
cat
0040199945f13da0de3cc39364ddf5a558b9cc580d4e19fbcf89a
981160553e5
```

得到文件系统层次结构

```
0040199945f13da0de3cc39364ddf5a558b9cc580d4e19fbcf89a981160553e5-init
224b05bde06843f08071e4d8f3be95253e56eb923eca01931ff6e03526e78839
```

```
b8c86b22b60d1a7235d2c53f6eb53c382865669622c38bdb621ed6c355c9f47e
dd7388470fb0a5991fb700c098381ff24c6e55e8f97b569ea08d8c1cf2150211
4315d67bb0bd296cf30a995ffca1e59e1e5c4691935daafdf447e124c4e2973d
0e399e82f3b3e83573f6ba9f2b0a6480e5695888aa87bde47945e361d49d94cd
```

4、由于在对运行中的容器进行操作时，不会使 -init 以外的层被更改，可以先将它们拷贝出来，存放在新建的目录中：

```
mkdir /home/pkusei/hw4_image
cp -r
0e399e82f3b3e83573f6ba9f2b0a6480e5695888aa87bde47945e361d49d94
cd /home/pkusei/hw4_image/0
cp -r
4315d67bb0bd296cf30a995ffca1e59e1e5c4691935daafdf447e124c4e297
3d /home/pkusei/hw4_image/1
cp -r
dd7388470fb0a5991fb700c098381ff24c6e55e8f97b569ea08d8c1cf21502
11 /home/pkusei/hw4_image/2
cp -r
b8c86b22b60d1a7235d2c53f6eb53c382865669622c38bdb621ed6c355c9f4
7e /home/pkusei/hw4_image/3
cp -r
224b05bde06843f08071e4d8f3be95253e56eb923eca01931ff6e03526e788
39 /home/pkusei/hw4_image/4
```

5、在运行中的容器内安装 vim 和 Python3：

```
apt update

apt install vim

apt install python3
```

6、将容器最上一层文件结构拷贝：

```
cp -r

0040199945f13da0de3cc39364ddf5a558b9cc580d4e19fbcf89a

981160553e5 /home/pkusei/hw4_image/5
```

7、按序挂载文件系统的拷贝

```
mkdir /home/pkusei/hw_mount
mount -t aufs -o
br=/home/pkusei/hw4_image/5=ro:/home/pkusei/hw4_image/4=ro:/home/pkusei/hw4
_image/3=ro:/home/pkusei/hw4_image/2=ro:/home/pkusei/hw4_image/1=ro:/home/p
kusei/hw4_image/0=ro none /home/pkusei/hw4_mount
```

8、在挂载点创建新镜像，并启动

```
cd /home/pkusei/hw4_mount
tar -c . | docker import - hw4_images
docker run -it --name hw4_container2 hw4_image /bin/bash
```

制作完成，可以再新容器中使用 vim 和 Python3 了。