使用Docker制作自定义PostgreSQL

本文档使用环境：

VMware 15.5.0 build-14665864

CentOS-Linux-7-2009

Docker version 20.10.17, build 100c701

Docker Compose version v2.9.0

一、安装包准备工作

1.1 *下载和打包docker安装包+docker-compose（只需要进行一次）

```
# 添加阿里云docker镜像仓库
$ yum install -y yum-utils
$ yum-config-manager --add-repo http://mirrors.aliyun.com/docker-
ce/linux/centos/docker-ce.repo
$ yum makecache fast
# 创建/usr/local/src/dockerrpm用于保存安装所需的安装包
$ cd /usr/local/src
$ mkdir dockerrpm
# 使用--downloadonly下载docker-ce的安装包和依赖
$ yum install docker-ce --downloadonly --downloaddir=dockerrpm
```

这次使用的安装包如下，如果要保持版本一致可以进行参考：

其中需要注意 docker-ce-20.10.17-3.el7.x86_64.rpm 和 docker-ce-cli-20.10.17-3.el7.x86_64.rpm 和 docker-ce-rootless-extras-20.10.17-3.el7.x86_64.rpm 的版本。其他软件包的版本可以不同

```
[root@master src] ls dockerrpm
audit-libs-python-2.8.5-4.el7.x86_64.rpm
fuse-overlayfs-0.7.2-6.el7_8.x86_64.rpm
checkpolicy-2.5-8.el7.x86_64.rpm
libcgroup-0.41-21.el7.x86_64.rpm
containerd.io-1.6.6-3.1.el7.x86_64.rpm
libseccomp-2.3.1-4.el7.x86_64.rpm
container-selinux-2.119.2-1.911c772.el7_8.noarch.rpm
libsemanage-python-2.5-14.el7.x86_64.rpm
docker-ce-20.10.17-3.el7.x86_64.rpm
policycoreutils-python-2.5-34.el7.x86_64.rpm
docker-ce-cli-20.10.17-3.el7.x86_64.rpm
python-IPy-0.75-6.el7.noarch.rpm
docker-ce-rootless-extras-20.10.17-3.el7.x86_64.rpm
setools-libs-3.3.8-4.el7.x86_64.rpm
docker-scan-plugin-0.17.0-3.el7.x86_64.rpm
slirp4netns-0.4.3-4.el7_8.x86_64.rpm
fuse3-libs-3.6.1-4.el7.x86_64.rpm
```

除了 docker 的 rpm 安装包外，还需要下载 docker-compose 的程序。官方网址在https://github.com/docker/compose/releases

```
# 下载二进制文件
[root@master src] wget
https://github.com/docker/compose/releases/download/v2.9.0/docker-compose-linux-x86_64
或者
[root@master src] wget
https://get.daocloud.io/docker/compose/releases/download/v2.9.0/docker-compose-Linux-x86_64
# 修改文件名（注意两种下载方式大小写不同）
[root@master src] mv docker-compose-linux-x86_64 docker-compose
# 放在安装包文件夹中
[root@master src] mv docker-compose dockerrpm/
```

最后，将安装包文件夹打包

```
# 添加成压缩文件
$ tar -zcvf docker-ce-20.10.tar.gz ./dockerrpm/*
```

1.2 将docker安装包传输到其他电脑上

当压缩好了 docker-ce-20.10.tar.gz 安装包后，便可以将安装包传输给其他电脑进行安装，而不需要从官方源/阿里源重新下载 docker-ce 安装文件。并且使用统一的安装包可以保持软件的版本一致

在这里以SSH为例，可以使用 scp 命令进行文件传输，从 master 分发到 node1 和 node2，然后在各机进行解压：

```
[root@master src] scp docker-ce-20.10.tar.gz root@node1:/usr/local/src
[root@master src] scp docker-ce-20.10.tar.gz root@node2:/usr/local/src
# 在node1上
[root@node1 ~] cd /usr/local/src
[root@node1 src] tar -zxvf docker-ce-20.10.tar.gz
# 在node2上
[root@node2 ~] cd /usr/local/src
[root@node2 src] tar -zxvf docker-ce-20.10.tar.gz
```

二、docker的安装和启动

2.1 解压和安装docker安装包

```
# 解压压缩包
[root@master src] tar -zxvf docker-ce-20.tar.gz
# 使用安装文件夹下的所有rpm包
[root@master src] rpm -ivh dockerrpm/*.rpm --force --nodeps
```

2.2 安装docker-compose

```
# 将docker-compose文件放在/usr/local/bin目录下
[root@master src] cp dockerrpm/docker-compose /usr/local/bin/
# 添加执行权限
[root@master src] chmod +x /usr/local/bin/docker-compose
# 测试安装是否成功
[root@master src] docker-compose --version
Docker Compose version v2.9.0
```

2.3 配置docker（可选）

参考https://docs.docker.com/engine/install/linux-postinstall/

为了避免使用 root 产生误操作，需要添加名为 docker 的用户来使用 docker。（使用 root 用户或者 sudo 命令进行）

```
# 创建docker组
$ groupadd docker
# 添加docker用户
$ useradd docker -g docker
$ passwd docker
#（设置docker用户密码）
# 添加docker用户到docker组中
$ usermod -aG docker docker
```

启动docker服务，并添加到自启动项

```
$ systemctl start docker
$ systemctl status docker
$ systemctl enable docker
```

配置镜像源（可选）

docker所使用的镜像可以从远程源中拉取，如果配置了国内的镜像源可以进行加速

```
$ mkdir /etc/docker
$ touch /etc/docker/daemon.json
$ cat<<EOF > /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {"max-size":"200m", "max-file":"3"}
  "registry-mirrors" : [
    "http://hub-mirror.c.163.com",
    "http://registry.docker-cn.com",
    "http://docker.mirrors.ustc.edu.cn"
  ]
}
EOF
$ systemctl restart docker
```

2.4 启动docker并测试

安装完成后，建议使用 su docker 切换到 docker 用户进行操作

```
# 查看docker版本
[docker@master ~] docker -v
Docker version 20.10.17, build 100c701
# 查看本地镜像文件
[docker@master ~] docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
# 查看运行中的容器，加上-a可以查看所有状态的容器
[docker@master ~] docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
# 查看运行中容器的系统占用，使用Ctrl+C退出
[docker@master ~] docker stat
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT   MEM %     NET I/O   BLOCK
I/O   PIDS
```

运行简单的hello-world示例：

运行 docker run hello-world，由于没有名为 hello-world 的本地镜像文件，因此会从官方源中进行拉取。接下来 docker 会根据 hello-world 镜像创建一个容器并启动。启动后便可以看到容器所输出的消息

```
[docker@master ~]$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:53f1bbee2f52c39e41682ee1d388285290c5c8a76cc92b42687eecf38e0af3f0
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

运行完 hello-world 后，可以查看镜像和根据镜像所创建的容器

```
[docker@master ~]$ docker images
REPOSITORY           TAG       IMAGE ID       CREATED             SIZE
hello-world          latest    feb5d9fea6a5   10 months ago       13.3kB
[docker@master ~]$ docker ps -a
CONTAINER ID   IMAGE               COMMAND                  CREATED
STATUS                      PORTS     NAMES
1fb50fbdd2a9   hello-world         "/hello"                 4 minutes ago
Exited (0) 4 minutes ago               flamboyant_mestorf
```

删除容器和镜像（可选）

```
# 根据id或者名字删除容器
$ docker rm 1fb50fbdd2a9
或者
$ docker rm flamboyant_mestorf
# 根据id或者仓库:tag删除镜像
$ docker rmi feb5d9fea6a5
或者
$ docker rmi hello-world:latest
```

三、PostgreSQL官方镜像安装（可选）

3.1 镜像下载和容器启动

docker对镜像使用<仓库:tag>作为标识。如果不加 tag 默认为 <仓库:latest>

docker run 相当于 docker create + docker start

其中 -v 命令为挂载目录，使用 -v 本机目录:容器内目录 的方式，将容器内的 /data/pgdata 映射到本机的 /data/docker/pgdata 目录下。这样就可以在本机中查看容器内的文件目录，并且在删除容器后，挂载目录的内容仍然保留

```
# 创建用于挂载的本地文件夹
[root@master src] mkdir /data/docker
[root@master src] chown -R docker /data/docker
```

```
[root@master src] su docker
# 从仓库拉取postgres12.5官方镜像
[docker@master ~]$ docker pull postgres:12.5
# 从官方镜像创建容器
[docker@master ~]$ docker run -d \
    --name=postgres_app \
    -e POSTGRES_PASSWORD=postgres \
    -e PGDATA=/data/pgdata1 \
    -v /data/docker/pgdata1:/data/pgdata1 \
    -p 5432:5432 \
    postgres:12.5
控制台提示：
36f85d3b68f29849acd4aa0647d63f6730ef308066472bb8dbe1e12d4b99d0a9
```

控制台提示的这一串字符即为创建的容器ID，但是在实际使用中不需要完整的ID，取前面12位

在Names一栏如果不进行自定义的话，docker会自动生成一个名字

用Name或者ID都可以指定容器

```
# 查看刚才创建并运行的容器
[docker@master ~]$ docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED
STATUS              PORTS                                       NAMES
36f85d3b68f2   postgres:12.5   "docker-entrypoint.s…"   About a minute ago   Up
About a minute   0.0.0.0:5432->5432/tcp, :::5432->5432/tcp   postgres_app
```

3.2 进入容器使用psql

此时，容器已经在运行中，使用 docker exec -it postgres_app /bin/bash 进入容器内部

其中 -it 的含义为以带命令提示符的交互终端的形式进入容器中，/bin/bash 的含义为进入时执行 bash 脚本，通过 shell 来和容器的 Linux 进行交互。在终端使用 exit 退出

```
[docker@master docker_postgres]$ docker exec -it postgres_app /bin/bash
root@36f85d3b68f2:/# ls
bin   data  docker-entrypoint-initdb.d  etc   lib    media  opt   root  sbin  sys
usr
boot  dev   docker-entrypoint.sh        home  lib64  mnt    proc  run   srv   tmp
var
root@36f85d3b68f2:/# su postgres
postgres@36f85d3b68f2:/$ psql
psql (12.5 (Debian 12.5-1.pgdg100+1))
Type "help" for help.

postgres=# SELECT version();
                     version
-------------------------------------------------
 PostgreSQL 12.5 (Debian 12.5-1.pgdg100+1) on x86_64-pc-linux-gnu, compiled by
gcc (Debian 8.3.0-6) 8.3.0, 64-bit
(1 row)
```

可以使用命令对容器进行关闭和重启

```
$ docker stop postgres_app
$ docker start postgres_app
```

容器内可以使用 psql 来对数据库进行操作。而由于前面使用了 -p 5432:5432 进行端口映射，将容器的 5432 端口映射到了主机的 5432 端口。因此可以通过本机访问容器的数据库。例如我的虚拟机 IP 为 192.168.182.131，在笔记本本机的 Navicat 连接 192.168.182.131:5432，就能连接到容器内部的数据库

四、制作自定义PostgreSQL镜像

为了在官方镜像的基础上安装软件、修改 PostgreSQL 的配置文件等，可以使用 Dockerfile 来进行自定义的进行制作。再使用 docker-compose 添加文件挂载、端口映射等，将镜像部署成容器或者服务

作为示例，本文档以官方的 postgres:12.5 镜像为基础，添加自定义的 postgresql.conf 配置文件，然后打包制作成新的镜像

4.1 制作过程介绍

postgres 官方镜像页https://hub.docker.com/_/postgres/

官方镜像主要有`POSTGRES_USER`、`POSTGRES_PASSWORD`、`POSTGRES_DB`、`PGDATA`等几个重要参数，可以用 docker run -e 参数=值 的方式来进行指定

在使用 docker run 从镜像初始化成容器时，首先会根据`POSTGRES_USER`（默认为postgres）和`POSTGRES_PASSWORD`（必须要定义）创建用户，然后在`PGDATA`（默认为/var/lib/postgresql/data，建议使用/data/pgdata）执行 initdb，创建 postgresql 的数据目录。而初始数据库的名称为`POSTGRES_DB`（默认为`POSTGRES_USER`的值）

在容器的根目录下会有 /docker-entrypoint-initdb.d/ 文件夹，初始为空，用户可以将所需要的自定义文件放进去。在初始化容器时，执行 initdb 完毕后，会自动扫描该文件夹下的所有 .sh 和 .sql 脚本，进行自动执行

那么就可以在 /docker-entrypoint-initdb.d/ 文件夹里，放入自定义的 postgresql.conf 和自己编写的 updateConfig.sh 脚本。在初始化时自动执行脚本，将 .conf 配置文件复制到 $PGDATA 目录下并重启数据库。这样就可以将自定义的配置写入到容器的数据库了

4.2 *使用Dockerfile制作自定义镜像（只需要进行一次）

首先准备一个空的文件夹，作为工作目录。并在文件夹中准备一份自定义的 postgresql.conf 文件

```
[docker@master ~]$ cd ~/docker_postgres/
[docker@master docker_postgres]$ ls
postgresql.conf
```

postgresql.conf 部分配置的参考：

其中修改了 log 的默认目录，改为 /var/log/pglog，如果没有在 Dockerfile 或者 docker-compose.yml 中进行配置，会导致运行时出现找不到目录的错误

```
# Connections and Authentication
listen_addresses= '*'
port= 5432
max_connections= 280
# Disk
wal_level= logical
wal_compression= on
# Reporting and Logging
log_destination= 'stderr'
logging_collector= on
log_directory= '/var/log/pglog'
log_filename= 'postgresql-%Y-%m-%d_%H%M%S.log'
log_timezone= 'Asia/Shanghai'
# Client connection defaults
datestyle= 'iso, mdy'
timezone= 'Asia/Shanghai'
```

创建一份 updateConfig.sh 脚本，用于在容器启动后用自定义配置替换默认配置

```
[docker@master docker_postgres]$ vim updateConfig.sh
#!/usr/bin/env bash
cat /docker-entrypoint-initdb.d/postgresql.conf > $PGDATA/postgresql.conf
pg_ctl restart
```

创建 Dockerfile，使用 FROM postgres:12.5 选择官方镜像作为基础

使用 COPY 将 postgresql.conf 和 updateConfig.sh 放入/docker-entrypoint-initdb.d/ 文件夹

使用 RUN 创建日志目录 /var/log/pglog，并修改相应的文件权限

使用 EXPOSE 将 5432 端口暴露到外部

```
FROM postgres:12.5
MAINTAINER tanjj
COPY postgresql.conf updateConfig.sh /docker-entrypoint-initdb.d/
RUN chmod a+r /docker-entrypoint-initdb.d/* \
    && mkdir /var/log/pglog \
    && chown 999:999 /var/log/pglog \
    && chmod 775 /var/log/pglog
EXPOSE 5432
```

此时，工作文件夹中已经有了 Dockerfile、postgresql.conf、updateConfig.sh 三个文件

```
[docker@master docker_postgres]$ ls
Dockerfile  postgresql.conf  updateConfig.sh
```

接下来，就可以使用 docker build 构建镜像

其中 -t 为新建镜像的名称:tag，可以自定义。而后面跟着的 . 含义为指定当前目录为工作目录，COPY 步骤中的 postgresql.conf 和 updateConfig.sh 都是从工作目录中复制到镜像目录中

```
[docker@master docker_postgres]$ docker build -t postgres_conf:12.5 .
Sending build context to Docker daemon  32.77kB
Step 1/5 : FROM postgres:12.5
 ---> 5fa7773911d6
Step 2/5 : MAINTAINER tanjj
 ---> Using cache
 ---> e8b7e7f0481e
Step 3/5 : COPY postgresql.conf updateConfig.sh /docker-entrypoint-initdb.d/
 ---> Using cache
 ---> feee6eaa38f7
Step 4/5 : RUN chmod a+r /docker-entrypoint-initdb.d/*     && mkdir
/var/log/pglog     && chown 999:999 /var/log/pglog     && chmod 775
/var/log/pglog
 ---> Using cache
 ---> 72f77773b405
Step 5/5 : EXPOSE 5432
 ---> Using cache
 ---> 7ec31b241cc2
Successfully built 7ec31b241cc2
Successfully tagged postgres_conf:12.5
```

构建完毕后，可以查看新创建的镜像 postgres_new:12.5

```
[docker@master docker_postgres]$ docker images
REPOSITORY           TAG       IMAGE ID       CREATED          SIZE
postgres_conf        12.5      7ec31b241cc2   7 minutes ago    314MB
postgres             12.5      5fa7773911d6   18 months ago    314MB
```

4.3 使用文件迁移镜像

使用 docker save 可以将本地的镜像保存为 .tar.gz 文件，原来 314MB 的镜像压缩后变成 106MB

然后使用 scp 等工具可以将打包好的镜像传输到其他设备上

```
[docker@master ~]$ docker save postgres_conf:12.5 | gzip > postgres_conf-
12.5.tar.gz
[docker@master ~]$ scp postgres_conf-12.5.tar.gz docker@node1:/home/docker
docker@node1's password:
postgres_conf-12.5.tar.gz                     100%  106MB  88.5MB/s   00:01
```

在传输完成后，使用 docker load 即可将镜像加载到本地。可以看到 ID、tag 等信息与原来保持一致，属于同一个镜像

```
[docker@node1 ~]$ docker load -i postgres_conf-12.5.tar.gz
a55b203bc4be: Loading layer  30.21kB/30.21kB
6ca858a23f56: Loading layer  32.26kB/32.26kB
Loaded image: postgres_conf:12.5
[docker@node1 ~]$ docker images
REPOSITORY           TAG       IMAGE ID       CREATED          SIZE
postgres_conf        12.5      7ec31b241cc2   18 minutes ago     314MB
```

4.4 使用阿里云网络仓库管理镜像（仅供参考）

前往阿里云容器镜像服务控制台，创建个人实例https://cr.console.aliyun.com/cn-guangzhou/instances

地域推荐选则广州，创建镜像仓库，其中命名空间为自定义，仓库名称建议与镜像保持一致。这里使用postgres_new作为仓库名

其中登录、上传、下载的命令都会在网页中给出

上传流程为：登录--给镜像打tag--推送到远程仓库

```
$ docker login --username=[阿里云用户名] registry.cn-guangzhou.aliyuncs.com
$ docker tag [ImageId] registry.cn-guangzhou.aliyuncs.com/[阿里云用户
名]/postgres_conf:[镜像版本号]
$ docker push registry.cn-guangzhou.aliyuncs.com/[阿里云用户名]/postgres_conf:[镜像
版本号]
```

下载流程为：登录--从远程仓库拉取镜像

```
$ docker login --username=[阿里云用户名] registry.cn-guangzhou.aliyuncs.com
$ docker pull registry.cn-guangzhou.aliyuncs.com/[阿里云用户名]/postgres_conf:[镜像
版本号]
```

4.5 搭建内外私有仓库管理镜像（待更新）

参考https://www.cnblogs.com/leozhanggg/p/12050322.html、https://developer.aliyun.com/article/47292

```
$ docker run -d \              # 后台运行
--name registry-srv \     # 指定容器名
--restart=always \         # 设置自动启动
-p 5000:5000 \             # 端口映射宿主机，通过宿主机地址访问
-v /opt/zwx-registry:/var/lib/registry \      # 把镜像存储目录挂载到本地，方便管理和持久化
-v /opt/zwx-registry/srv-config.yml:/etc/docker/registry/config.yml \     # 把配置文件挂载到本地，方便修改和保存
registry
```

五、使用docker-compose加载镜像

docker-compose的安装参考 **2.2** 部分

在第四节中，我们已经在官方 postgres 镜像的基础上，加入自己的 postgresql.conf 配置文件，制作了新的镜像。接下来，需要根据自定义镜像来创建和运行容器

比起使用 docker run 加上 -e/-v/-p 等命令参数，可以使用 docker-compose.yml 将配置写入文件之中，只需要 docker-compose up 命令就可以简单地建立容器

5.1 编写docker-compose.yml

创建 docker-compose.yml 文件（建议在 docker_postgres 目录下）：

image 指定所使用的镜像

container_name 指定所使用的容器名称

ports 指定端口映射，格式为 主机端口:容器端口

volumes 指定目录挂载（可选），格式为 主机目录:容器目录。这里将 postgres 的数据目录 pgdata 映射到了主机上

environment 指定容器启动时的环境变量，特别注意 `PGDATA` 的值，需要与上面挂载的目录对应

下面是 docker-compose.yml 文件的内容，仅供参考：

```
version: '2'

services:
  postgres:
    image: postgres_new:12.5
    container_name: postgres_app
    ports:
      - '5432:5432'
    volumes:
      - /data/docker/pgdata:/data/pgdata
      - /data/docker/pglog:/var/log/pglog
    environment:
      - POSTGRES_DB=postgres
      - PGDATA=/data/pgdata
```

```
      - POSTGRES_USERNAME=postgres
      - POSTGRES_PASSWORD=postgres
```

5.2 挂载目录权限问题

在进行挂载时，容器内的文件目录会挂载到主机指定的目录下。如果本机中原本没有挂载目录的话，系统会创建相应的目录。但是这样就可能会出现文件权限冲突的问题

在本文档的例子中，将容器的 /data/pgdata 挂载到了本机的 /data/docker/pgdata，将容器的 /var/log/pglog 挂载到了本机的 /data/docker/pglog 下

其中，本机的 pgdata 文件夹可以交给容器自动挂载，但是 pglog 文件夹需要提前手动创建

pgdata:

假设本机中没有 /data/docker 目录，甚至没有 /data 目录。那么在启动时就会自动创建 /data/docker 目录，并且目录权限为 `drwxr-xr-x root:root`

接下来，在容器内的数据库初始化期间，会创建 $PGDATA=/data/pgdata 目录，并赋予 postgres 用户相应的文件所属权限。相应地，在主机的 /data/docker/pgdata 目录权限会变成 `drwx------ polkitd:root`，而目录内文件的权限会变成 `drwx------ polkitd:input` 或者 `-rw------- polkitd:input`

```
[root@master ~]# ll /data/docker/
drwx------ 19 polkitd root    4096 Aug 10 10:53 pgdata
[root@master ~]# ll /data/docker/pgdata
drwx------ 5 polkitd input    41 Aug 10 10:53 base
-rw------- 1 polkitd input    55 Aug 10 10:53 current_logfiles
```

而本机中的 polkitd(uid=999) 用户，实际上对应的就是容器中的 postgres(uid=999) 用户。他们的 uid 是一样的，只不过在本机和容器中显示名称不同

```
[root@master ~]# id polkitd
uid=999(polkitd) gid=998(polkitd) groups=998(polkitd)
```

pglog:

由于在本例中，容器内的 /var/log/pglog 目录是在 Dockerfile 内创建的，而在容器的数据库初始化期间没有进行权限修改。因此，在挂载时默认创建的本机 pglog 目录权限是 `drwxr-xr-x root:root`

虽然容器内的 pglog 权限属于 postgres，但是本机的 pglog 权限属于 root。在数据库写日志时，postgres 需要在 pglog 目录下创建日志文件，但是没有本机 pglog 目录的写权限。因此会触发权限问题

```
postgres  | 2022-08-10 11:33:48.124 CST [68] FATAL:  could not open log file
"/var/log/pglog/postgresql-2022-08-10_113348.log": Permission denied
```

因此，在创建和启动容器前，需要在本机先创建 pglog 文件夹，并赋予 polkitd(uid=999) 相应的权限

```
# 创建pglog目录，如果没有对应的/data/docker则创建根目录
[root@master ~]# mkdir -p /data/docker/pglog
# 将文件夹所属权给polkitd
[root@master ~]# chown 999:999 /data/docker/pglog
[root@master ~]# chmod 775 /data/docker/pglog
# 查看文件夹权限
[root@master ~]# ll /data/docker
total 0
drwxrwxr-x 2 polkitd input 6 Aug 10 11:41 pglog
```

5.3 启动容器

启动容器只需要一条命令即可：

```
[docker@master docker_postgres]$ docker-compose up
```

如果需要让容器在后台进行，则需要使用 -d 参数：

```
[docker@master docker_postgres]$ docker-compose up -d
```

控制台输出的过程：首先会用 initdb 启动数据库，然后执行 /docker-entrypoint.sh，扫描到 /docker-entrypoint-initdb.d/updateConfig.sh 并执行。updateConfig.sh 会修改 postgresql.conf 并重启数据库

```
[+] Running 2/0
 ⠿ Network docker_postgres_default  Created
              0.0s
 ⠿ Container postgres_app           Created
              0.0s
Attaching to postgres_app
postgres_app  | The files belonging to this database system will be owned by user
"postgres".
postgres_app  | This user must also own the server process.
# 中间略
postgres_app  | server started
postgres_app  |
postgres_app  | /usr/local/bin/docker-entrypoint.sh: ignoring /docker-entrypoint-
initdb.d/postgresql.conf
postgres_app  |
postgres_app  | /usr/local/bin/docker-entrypoint.sh: sourcing /docker-entrypoint-
initdb.d/updateConfig.sh
postgres_app  | 2022-08-09 09:47:43.756 UTC [47] LOG:  received fast shutdown
request
# 中间略
postgres_app  | 2022-08-09 17:47:44.074 CST [1] LOG:  starting PostgreSQL 12.5
(Debian 12.5-1.pgdg100+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 8.3.0-
6) 8.3.0, 64-bit
# ......
postgres_app  | 2022-08-09 17:47:44.094 CST [1] HINT:  Future log output will
appear in directory "/var/log/pglog".
```

接下来就可以通过主机 IP 和 5432 端口来访问容器内的 postgres 数据库

而如果需要删除容器，则也是使用一条命令即可：

```
[docker@master docker_postgres]$ docker-compose down
```

六、打包并编写shell脚本实现一键安装（可选）

为了方便使用者的安装，建议将安装包和配置文件打包在一个文件夹中，方便启动。同时，编写一个 shell 脚本用于 docker 的安装以及镜像的导入。

本文档的按照目录结构如下，包括了 docker 相关安装文件、postgres 自定义镜像、postgres 镜像配置文件以及一键安装脚本：

```
docker_postgres_setup # 安装文件夹
    dockerrpm # docker-compose安装文件与docker安装rpm包
        docker-compose-Linux-x86_64
        docker-ce-20.10.17-3.el7.x86_64.rpm
        ......
    postgres_build # postgres镜像相关配置文件
        docker-compose.yml
        Dockerfile
        postgresql.conf
        updateConfig.sh
    postgres_conf-12.5.tar.gz # postgres自定义镜像
    setup.sh # 安装脚本
```

其中 setup.sh 参考的内容如下：

```
#!/bin/bash

echo "---------------------------------"
echo "Installing docker..."
echo "---------------------------------"
# install docker rpm
rpm -ivh dockerrpm/*.rpm --force --nodeps
# install docker-compose
cp dockerrpm/docker-compose-Linux-x86_64 /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
# setup docker user
groupadd docker
useradd docker -g docker
echo "docker:docker" | chpasswd
usermod -aG docker docker
# check docker and docker-compose
systemctl start docker
systemctl enable docker
echo "---------------------------------"
echo "docker and docker-compose version"
echo "---------------------------------"
docker -v
docker-compose --version

echo "---------------------------------"
echo "Loading image..."
echo "---------------------------------"
```

```
docker load -i postgres_conf-12.5.tar.gz

echo "----------------------------------"
echo "Creating file path..."
echo "----------------------------------"
mkdir -p /data/docker/pgdata
mkdir -p /data/docker/pglog
chown -R 999:999 /data/docker/
chmod -R 755 /data/docker
echo "ls -l /data/docker"
ls -l /data/docker

echo "----------------------------------"
echo "Finished!"
```

尾注：在新主机上配置PostgreSQL

如果需要在新的主机上安装已经自定义好的 postgres 镜像，需要提供以下文件：

1. docker-ce 的若干 rpm 安装包
2. docker-compose 二进制程序文件
3. 制作完成的自定义 postgres 镜像（必须）
4. docker-compose.yml 配置文件

在前面教程的 [1.2 将docker安装包传输到其他电脑上]、[4.3 使用文件迁移镜像] 等小节，示例了使用 scp 方式传输文件的方式，但是使用了分散的文件传输方式

建议参考第六节，将所有需要的文件放在一个目录下，然后打包成压缩包再传输。这样在新主机上只需要解压压缩包并运行安装脚本，即可完成镜像的导入。