准备CenOs7操作系统

参考云盘devops实战下面的虚拟机安装资料

完成IP配置，用ssh工具登录。

使用脚本自动安装-docker

```
curl -sSL https://get.docker.com/ > get-docker.sh
sh get-docker.sh --mirror Aliyun
```

备用命令：

```
curl -fsSL https://get.docker.com -o get-docker.sh
```

若以上命令出现网络错误可以尝试以下命令

```
curl -sSL http://acs-public-mirror.oss-cn-hangzhou.aliyuncs.com/docker-
engine/internet | sh -
```

启动Docker进程

```
systemctl enable docker   #设置默认开机启动
systemctl start docker    #启动docker
```

检查docker是否安装成功

```
[root@localhost ~]# docker --version
Docker version 20.10.6, build 370c289
```

配置docker国内源

`/etc/docker/daemon.json` 中写入如下内容（如果文件不存在请新建该文件:mkdir /etc/docker&&touch daemon.json)

```
{
  "registry-mirrors": [
    "https://docker.mirrors.ustc.edu.cn",
    "https://registry.docker-cn.com",
    "https://hub-mirror.c.163.com",
    "https://mirror.ccs.tencentyun.com",
    "https://reg-mirror.qiniu.com"
  ]
}
```

注意，内容格式必须符合json要求，否则会导致docker启动失败！建议直接赋值以上文本即可

输入以下命令重启docker

```
systemctl daemon-reload
systemctl restart docker
```

docker练习-拉取镜像

想要运行容器必须提前下载好对应的镜像，比如我们想运行一个mysql5.6版本的容器，那么我们需要先在本地有这个镜像才能运行。

镜像需要在镜像仓库上去下载，就像代码是托管在代码仓库上的，默认情况下我们需要从公有仓库，也就是

上下载镜像。

dockerhub上有海量的镜像供我们选择，我们构建环境的时候可以直接下载其中的镜像来使用

比如下载一个mysql5.7镜像

```
[root@localhost ~]# docker pull mysql:5.7
5.7: Pulling from library/mysql
f7ec5a41d630: Pull complete
9444bb562699: Pull complete
6a4207b96940: Pull complete
181cefd361ce: Pull complete
8a2090759d8a: Pull complete
15f235e0d7ee: Pull complete
d870539cd9db: Pull complete
cb7af63cbefa: Pull complete
151f1721bdbf: Pull complete
fcd19c3dd488: Pull complete
415af2aa5ddc: Pull complete
Digest: sha256:a655529fdfcbaf0ef28984d68a3e21778e061c886ff458b677391924f62fb457
Status: Downloaded newer image for mysql:5.7
docker.io/library/mysql:5.7
```

这个拉取的语法是

```
docker pull 镜像:Tag
```

如果不加标签，默认会拉取最新版本的镜像，即latest

如：docker pull nginx

```
[root@localhost ~]# docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
bb79b6b2107f: Pull complete
111447d5894d: Pull complete
a95689b8e6cb: Pull complete
1a0022e444c2: Pull complete
32b7488a3833: Pull complete
Digest: sha256:ed7f815851b5299f616220a63edac69a4cc200e7f536a56e421988da82e44ed8
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

拉取的镜像默认保存到了系统的 `/var/lib/docker/` 目录下面，可以通过一个简单的命令查看当前系统有哪些docker镜像.

```
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
nginx               latest              f35646e83998        2 days ago
133MB
mysql               5.7                 4f36ba851740        2 days ago
302MB
```

之前安装的myslq和nginx镜像都能看到。

镜像相关命令

```
docker pull 镜像:Tag            # 拉镜像
docker push 镜像:Tag            # 推送镜像
docker rmi 镜像ID或者镜像名      # 删除镜像，需要提前删除对应的容器
```

docker练习-创建容器

有了镜像，我们可以基于镜像来运行容器。

启动容器的命令语法是

```
docker run <options> 镜像:tag commands
```

这里解释一下，options表示选项，用于运行容器时一些额外的配置，commands表示运行在容器内部的命令，比如容器内部要运行一个python脚本，commands可以替换为 python commands。

常用的选项有：

```
-d 指定容器运行后台

-i 用于控制台交互

-t 支持终端登录，通常和-i参数一起使用

-p 映射容器端口，用法：-p 宿主机端口:容器端口
```

```
-v 挂载容器存储卷，用法：-v 宿主机路径:容器路径

-e 设置容器的环境变量用于容器内的应用程序进行相关配置，用法 -e 环境变量名=变量值

--name="container_name" 指定容器的名称

--rm 退出容器时删除容器  与--restart冲突

--restart="always" 停止容器时是否自动重启  与 --rm冲突
```

案例：运行一个mysql容器

```
docker run -it  -p 3306:3306 -e MYSQL_ROOT_PASSWORD=devops mysql:5.7
```

这个时候我们看到容器内部的运行情况了

```
2021-04-23T05:54:54.267519Z 0 [Note] Server hostname (bind-address): '*'; port:
3306
2021-04-23T05:54:54.268108Z 0 [Note] IPv6 is available.
2021-04-23T05:54:54.268123Z 0 [Note]   - '::' resolves to '::';
2021-04-23T05:54:54.268139Z 0 [Note] Server socket created on IP: '::'.
2021-04-23T05:54:54.275363Z 0 [Warning] Insecure configuration for --pid-file:
Location '/var/run/mysqld' in the path is accessible to all OS users. Consider
choosing a different directory.
2021-04-23T05:54:54.282909Z 0 [Note] Event Scheduler: Loaded 0 events
2021-04-23T05:54:54.283093Z 0 [Note] mysqld: ready for connections.
Version: '5.7.34'  socket: '/var/run/mysqld/mysqld.sock'  port: 3306  MySQL
Community Server (GPL)
```

mysql服务默认监听3306端口，但是容器内部的网络和宿主机不在一个网段，所以我们把该端口映射到了宿主机的3306端口，现在我们通过访问宿主机的3306端口就可以访问mysql了

此时我们的进程是运行在前台的，如果关闭窗口就可能造成容器进程退出，所以如果想让容器一直不退出，可以在运行的时候加上-d参数，让容器进程作为守护进程。

```
docker run -it -d -p 3307:3306 -e MYSQL_ROOT_PASSWORD=devops --name="db_mysql"
mysql:5.7
```

此时可以通过 `docker ps` 命令查看运行中的容器

```
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS                    NAMES
0d13dc652f11        mysql:5.6           "docker-entrypoint.s…"   About an hour
ago   Up About an hour    0.0.0.0:3306->3306/tcp   laughing_galois
```

若容器在运行过程中发生异常而退出，docker ps命令是看不到终止的进程的，可以通过-a参数列出所有的进程，包括已经停止的(状态为Exited)

```
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                          PORTS                     NAMES
0d13dc652f11        mysql:5.6           "docker-entrypoint.s…"   About an hour
ago   Up About an hour               0.0.0.0:3306->3306/tcp   laughing_galois
282438a29adb        mysql:5.6           "docker-entrypoint.s…"   About an hour
ago   Exited (1) About an hour ago
```

我们可以通过start和stop 来启动或停止一个容器 参数为容器ID或者容器名

如:

```
[root@localhost ~]# docker stop 0d13dc652f11    #停止一个容器
0d13dc652f11
[root@localhost ~]# docker start 0d13dc652f11   #恢复停止中的容器
0d13dc652f11
```

容器相关命令

```
docker run <options> 镜像:tag commands   # 新建一个容器并启动
docker stop 容器ID或名称                  # 停止运行中的容器
docker start 容器ID或名称                 # 启动停止的容器
docker restart 容器ID或名称               # 重启容器
```

用例数据链接工具登录mysql容器，新建数据库后面备用

```
CREATE DATABASE `db_autotp` DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

容器部署实战练习

部署Nginx

```
docker run -it -d -p 80:80 --name mynginx --rm nginx
```

本地没有镜像会自动从镜像仓库拉取镜像

```
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
latest: Pulling from library/nginx
f7ec5a41d630: Already exists
aa1efa14b3bf: Pull complete
b78b95af9b17: Pull complete
c7d6bca2b8dc: Pull complete
cf16cd8e71e0: Pull complete
0241c68333ef: Pull complete
Digest: sha256:75a55d33ecc73c2a242450a9f1cc858499d468f077ea942867e662c247b5e412
Status: Downloaded newer image for nginx:latest
eae4094263f2854fec41d6dc6b3441ee4069a7bc66567f9c39a2526fbda08c26
```

查看镜像

```
[root@localhost ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
mysql        5.7       87eca374c0ed   4 days ago    447MB
nginx        latest    62d49f9bab67   10 days ago   133MB
```

查看容器

```
[root@localhost ~]# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
PORTS                               NAMES
9a45489808a5   nginx     "/docker-entrypoint.…"   3 seconds ago   Up 2 seconds
0.0.0.0:80->80/tcp, :::80->80/tcp   mynginx
```

访问主机地址，http://192.168.21.142/

看到Nginx

从容器拷贝nginx的配置文件到宿主机，方便后续修改，因为容器内没有vi命令不方便修改。

```
docker cp mynginx:/etc/nginx conf
```

上面的mynginx是你的容器名或者ID

命令格式为：docker cp 容器名/ID:容器目录 本地目录

将静态文件拷贝到 html 目录

```
[root@localhost ~]# cp -rf /root/software/autotpsite/dist html
[root@localhost ~]#
[root@localhost ~]# ls html/
assets          environments.html  jira_bug.html        jira_project.html
plan_view.html     report_view.html  testreport.html    webinterface_view.html
building.html   env_view.html      jira_bug_view.html   jira_project_view.html
projects.html      scripts           users.html
case_view.html  images             jira_case.html       js
project_view.html  testcase.html     vendors
css             index.html         jira_case_view.html  login.html
register.html      testplan.html     webinterface.html
```

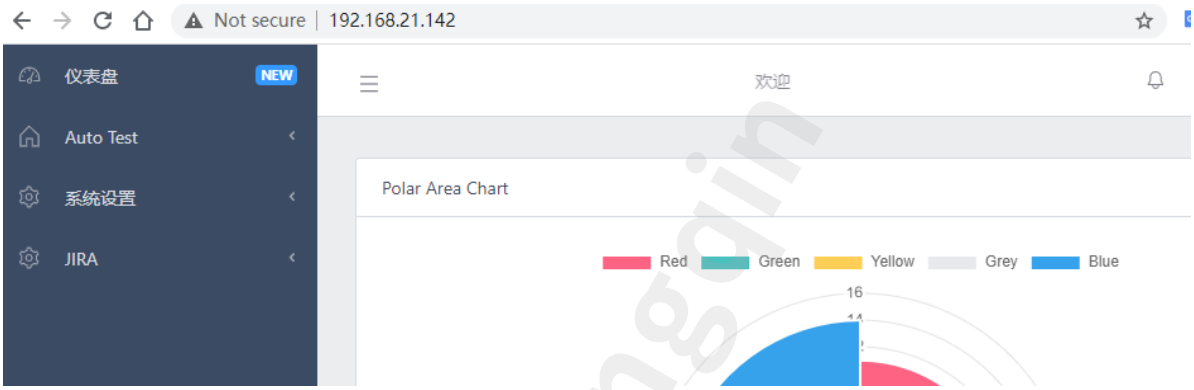
停掉刚刚启动的docker容器

```
[root@localhost ~]# docker stop mynginx
mynginx
```

重新以挂载目录的方式启动Nginx

```
docker run \
  --rm \
  --name mynginx \
  --volume "$PWD/html":/usr/share/nginx/html \
  --volume "$PWD/conf":/etc/nginx \
  -p 80:80 \
  -d \
  nginx
```

刷新页面，看到了测试平台页面



部署后台服务

部署测试平台后台服务 Django+uwsgi

首先服务都依赖Python3，所以拉取python3镜像

```
[root@localhost ~]# docker pull python:3.8
3.8: Pulling from library/python
bd8f6a7501cc: Pull complete
44718e6d535d: Pull complete
efe9738af0cb: Pull complete
f37aabde37b8: Pull complete
3923d444ed05: Pull complete
1ecef690e281: Pull complete
48673bbfd34d: Pull complete
b761c288f4b0: Pull complete
57ed4d1661b5: Pull complete
Digest: sha256:ff2d0720243a476aae42e4594527661b3647c98cbf5c1735e2bb0311374521f4
Status: Downloaded newer image for python:3.8
docker.io/library/python:3.8
```

查看镜像

```
[root@localhost ~]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
mysql        5.7       87eca374c0ed   4 days ago    447MB
nginx        latest    62d49f9bab67   10 days ago   133MB
python       3.8       02583ab5c95e   13 days ago   883MB
```

启动容器-挂载后端目录

```
docker run -it -d -v /root/software/autotpsite:/opt --rm --name autotpenv
python:3.8
```

修改uwsgi配置文件

```
[uwsgi]
chdir = ./
module = autotpsite.wsgi:application
http-socket = 0.0.0.0:8081
master = true
pidfile = ./uwsgi8081.pid
daemonize = ./uwsgi_server.log
# 只记录错误信息
disable-logging = true
socket-timeout=10
```

进入容器

```
docker exec -it autotpenv bash
```

并执行安装库操作

```
root@afc771e62714:/# cd opt/
root@afc771e62714:/opt# ls
autotpsite  dist  jiraExt  jira_client.py  manage.py  requirements.txt  sqtp
testcase  tmp.json  uwsgi.ini  uwsgi8081.pid  uwsgi_server.log  项目任务.txt
root@afc771e62714:/opt#  pip install -r requirements.txt -i
http://pypi.douban.com/simple/  --trusted-host pypi.douban.com
```

清华源：

```
-i https://pypi.tuna.tsinghua.edu.cn/simple  --trusted-host
pypi.tuna.tsinghua.edu.cn
```

如果换源方式下载库异常，可以配置pip.conf文件

```
mkdir -p /root/.config/pip && touch /root/.config/pip/pip.conf && echo "
[global]\ntimeout = 60\nindex-url = https://pypi.doubanio.com/simple\ntrusted-
host = pypi.doubanio.com" > /root/.config/pip/pip.conf
```

测试一下能否正常启动，注意配置文件的mysql IP要写成局域网的

```
root@2a703747f4d5:/opt/autotpsite# python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 24, 2021 - 04:02:11
Django version 3.1.7, using settings 'autotpsite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

安装uWSGI服务

```
pip install uwsgi -i http://pypi.douban.com/simple/  --trusted-host
pypi.douban.com
```

启动uWSGI

```
root@afc771e62714:/opt# uwsgi uwsgi.ini
[uWSGI] getting INI configuration from uwsgi.ini
root@afc771e62714:/opt# ps -ef |grep uwsgi
root        638      0  3 04:17 ?        00:00:00 uwsgi uwsgi.ini
root        640    638  0 04:17 ?        00:00:00 uwsgi uwsgi.ini
root        642     14  0 04:17 pts/1    00:00:00 grep uwsgi
```

容器环境调试OK，但是端口没有暴露出来，为了下次方便随时启动新容器，我们可以把容器内部的操作写成shell

```
# auto_deploy.sh
cd opt/ && pip install -r requirements.txt -i http://pypi.douban.com/simple/  --
trusted-host pypi.douban.com && pip install uwsgi -i
http://pypi.douban.com/simple/  --trusted-host pypi.douban.com && uwsgi uwsgi.ini
&& tail -f > /dev/null
```

将文件保存到root/software/autotpsite

重启启动容器

```
docker run -it -d -v /root/software/autotpsite:/opt -p 8081:8081 --rm --name
autotpenv python:3.8 sh /opt/auto_deploy.sh
```

修改nginx的配置文件，前面映射出来了，所以不用进入容器内部去修改了

```
[root@localhost ~]# vi conf/conf.d/default.conf

# 加入server块里面
location ~/(api/|jira/) {
    include        uwsgi_params;
    uwsgi_pass     127.0.0.1:8081;
    # proxy_pass   http://127.0.0.1:8081;   # 通用配置方式
}
```

重启nginx容器

```
docker restart mynginx
```

发现还是无法访问，因为容器的网络是和主机隔离的，有自己的一套网络

查看主机的网络发现有个docker0网卡，这个是docker使用的网卡，docker就通过这个网卡来进行网络通信。

```
[root@localhost ~]# ip addr
# 其他网卡
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:f6:c3:5a:f4 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:f6ff:fec3:5af4/64 scope link
```

查看容器有哪些网络

```
[root@localhost ~]# docker network ls
NETWORK ID     NAME      DRIVER    SCOPE
9f606136c768   bridge    bridge    local
c7fca13d33a9   host      host      local
b590d278d4b1   none      null      local
```

默认情况容器是挂载在bridge网络的，所以可以查看bridge网络的信息，查询到相关容器的内部IP，让后通过内部IP互相访问。

```
[root@localhost ~]# docker network inspect bridge
[
# ....其他信息.....

        "Containers": {
            "1e37b1bcf76f3b7e3f4925e6b59db1f6b500d1700e518334cc448bfe07b39392":
{
                "Name": "mynginx",
                "EndpointID":
"87320728c976851869c93c50c842d747ec837b1988f0645898c42d82d21408e1",
                "MacAddress": "02:42:ac:11:00:04",
                "IPv4Address": "172.17.0.4/16",
                "IPv6Address": ""
            },
            "b46ee986ac0321bb8e4d3b60a843d7be7cbbdad7a9d52224d03b12fa5b4c8a20":
{
                "Name": "sweet_morse",
                "EndpointID":
"67ae1f5567947346288f995a2fdb0e4632beb1846fa4a536a60f880a109b78d7",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            },
            "c79d6e834de69b66f5c1d3859b2709702f15cc4ff1e1b4e5a6bcf98033768a26":
{
                "Name": "autotpenv",
```

```
                "EndpointID":
"874b3e11137e7858ccbd059aaa5c90a660654b4d8e4ed933ba7c13f6d58ec1b1",
                "MacAddress": "02:42:ac:11:00:05",
                "IPv4Address": "172.17.0.5/16",
                "IPv6Address": ""
            },
            "eac8346299b4f94415150ad2dedba25649444a802a0ff9cefb51d3a33ff154cb":
{
                "Name": "inspiring_diffie",
                "EndpointID":
"d512e5e0c73c5862fca4bd94f83a05b3857bdaa7c0f318c64b7f4b9530aac86c",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
```

这里可以看到后台服务autotpenv的IP是172.17.0.5/16，所以以上配置文件改成

```
[root@localhost ~]# vi conf/conf.d/default.conf

# 加入server块里面
location ~/(api/|jira/) {
    include        uwsgi_params;
    uwsgi_pass     127.0.0.1:8081;
    # proxy_pass   http://172.17.0.5:8081;   # 通用配置方式
}
```

重启nginx服务就可以了

```
docker restart mynginx
```

附录

容器相关命令

```
# 查看容器日志
docker logs -f 容器ID/名称
# 查看容器详细信息
docekr inspect 容器ID/名称
```

卸载docker

1. 杀死所有运行容器

```
# docker kill $(docker ps -a -q)
```

2. 删除所有容器

```
# docker rm $(docker ps -a -q)
```

3. 删除所有镜像

```
# docker rmi $(docker images -q)
```

4. 停止 docker 服务

```
# systemctl stop docker
```

5. 删除存储目录

```
# rm -rf /etc/docker
# rm -rf /run/docker
# rm -rf /var/lib/dockershim
# rm -rf /var/lib/docker
```

```
rm -rf /etc/docker /run/docker /var/lib/dockershim /var/lib/docker
```

如果发现删除不掉，需要先 umount，如

```
umount /var/lib/docker/devicemapper
```

6. 卸载 docker

查看已安装的 docker 包

```
# yum list installed | grep docker
```

卸载相关包

```
# yum remove -y containerd.io.x86_64 docker-ce.x86_64 docker-ce-cli.x86_64
```

卸载旧版本（可选）

Docker改版之前 称为 `docker` 或者 `docker-engine`，若之前安装过旧版本需要卸载，使用以下命令卸载旧版本：

```
yum remove docker \
           docker-client \
           docker-client-latest \
           docker-common \
           docker-latest \
           docker-latest-logrotate \
           docker-logrotate \
           docker-selinux \
           docker-engine-selinux \
           docker-engine
```