

docker培训

1. 作为一个新兴的技术，问题很多，为什么要用

1. 直接原因：高大上
2. 外部原因：如果所有人只能用webx进行开发，你能呆多久？
3. docker优势如下

- 一. 一次创建或配置，可以在任意地方正常运行
- 二. docker的优势：相较传统虚拟机，docker可以做到秒级、甚至毫秒级的启动时间
- 三. docker 的镜像提供了除内核外完整的运行时环境，确保了应用运行环境一致性，不会因为环境原因造成bug

2. 离线安装

2.1 依赖包

1. 存储网络位置： \\nb-hz20084884\share\docker
2. 海康的yum源 CentOS-Base.repo
3. docker-ce-17.03.2.ce-1.el7.centos.x86_64.rpm
4. docker-ce-selinux-17.03.2.ce-1.el7.centos.noarch.rpm
5. jdk81.tar
6. docker-compose
7. docker_practice.pdf

2.2 执行下面脚本安装即可

检查yum源，新环境都需要替换掉 /etc/yum.repos.d/CentOS-Base.repo
yum remove docker docker-client docker-client-latest docker-common docker-latest docker-latest-logrotate docker-logrotate docker-selinux docker-engine-selinux docker-engine
yum install -y yum-utils device-mapper-persistent-data lvm2
yum install -y policycoreutils-python selinux-policy
yum install -y docker-ce-selinux-17.03.2.ce-1.el7.centos.noarch.rpm
yum install -y docker-ce-17.03.2.ce-1.el7.centos.x86_64.rpm
yum install -y vim
docker-compose文件放到 /usr/local/bin 目录里面
chmod u+x /usr/local/bin/docker-compose
建立软连接 ln -sf /usr/local/bin/docker-compose /usr/bin/

2.2.1 修改docker 默认安装位置

为何标红，看图，生产环境，磁盘空间总共只有8G

```
[root@HX-B2BYS-SOE1 ~]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
12c585850064	jdk81	"java -jar -Xms2g -X..."	4 months ago	Up 4 days		integral_mall
2a07a88cd258	jdk81	"java -jar -Xms2g -X..."	4 months ago	Up 4 days		hikappjob
25c03b14c738	tomcat8-jdk8	"sh /usr/local/src/a..."	6 months ago	Up 4 months		job_admin

```
[root@HX-B2BYS-SOE1 ~]# df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/mapper/centos-root	8.0G	6.1G	2.0G	76%	/
devtmpfs	3.9G	0	3.9G	0%	/dev
tmpfs	3.9G	0	3.9G	0%	/dev/shm
tmpfs	3.9G	65M	3.8G	2%	/run
tmpfs	3.9G	0	3.9G	0%	/sys/fs/cgroup
/dev/sdb1	50G	1.9G	45G	4%	/data
/dev/sda1	1014M	142M	873M	14%	/boot
overlay	8.0G	6.1G	2.0G	76%	/var/lib/docker/overlay2/094dc4c1e918f34b90c2005e9f4306e2d57c8188a80e2ad371c1965d4305407e0/merged
shm	64M	0	64M	0%	/var/lib/docker/containers/25c03b14c7387f1cdac48a579ac21efd703285631f383104313cb68ff7e7e577/mounts/shm
overlay	8.0G	6.1G	2.0G	76%	/var/lib/docker/overlay2/20a0b720b819444e51279c1140168eb67271d5d65112e403da8504a8eaddf7c5/merged
overlay	8.0G	6.1G	2.0G	76%	/var/lib/docker/overlay2/d58cca470204fde1df02dc32c58a0be00a3d2a3adcaa5f96e4d0fcad905a7903/merged
shm	64M	0	64M	0%	/var/lib/docker/containers/2a07a88cd258a7c254127788912b76d5e52a92e0912f2799b02d70d0646a741b/mounts/shm
shm	64M	0	64M	0%	/var/lib/docker/containers/12c58585006417f3710fa6da068839f819e85f6cf68ccabfd099d668a0608654/mounts/shm
tmpfs	768M	0	768M	0%	/run/user/0



异常: Free disk space is less than 10% on volume /
告警主机:HIK-B5BYS-SCHE2_10.1.174.136
状态: 异常
告警时间:2019.06.13 15:01:15
告警等级: Warning
问题详情:Free disk space on / (percentage):9.97 %
事件ID: 44973676
能多清理空间吗

2.2.2 解决方案-->修改默认安装位置

修改docker.service文件

```
vim /usr/lib/systemd/system/docker.service
```

在里面的EXECStart的后面增加后如下:

/data目录有45G的空间, 所以docker的路径改为/data/docker

```
ExecStart=/usr/bin/dockerd --graph /data/docker
```

2.3 启动docker 导入离线镜像jdk81

systemctl start docker 或者systemctl enable docker
docker load < jdk81.tar

3. 构建第一个项目

1. 利用原生docker命令

```
docker run -d --name hikapp_redis7003_1 --net=host -p 7003:7003 -v /home/redis/7003/redis.conf:/usr/local/redis/redis.conf google/cloud-cluster-redis:4.0.10
```

缺点: 第二个人无法运维, 无法写出一模一样的语句

2. 利用docker-compose.yml构建第一个项目

2.1 docker-compose的本质是调用docker提供的api接口

2.2 编写docker-compose.yml 文件, 内容如下图

```
hikusercenter:
  image: jdk8
  volumes:
    - /home/hikapp/hikusercenter/hikusercenter.jar:/usr/local/hikusercenter.jar:ro
    - /etc/localtime:/etc/localtime:ro
    - /home/hikapp/hikusercenter_log/log_7087:/project/deploy/logs/:rw
  working_dir: /usr/local/hikusercenter
  ports:
    - "7087:7087"
  container_name: hikusercenter
  restart: on-failure
  network_mode: "host"
  entrypoint: java -jar -Xms2g -Xmx2g -XX:+HeapDumpOnOutOfMemoryError -Dspring.profiles.active=prod /usr/local/hikusercenter.jar --server.port=7087
  logging:
    driver: "fluentd"
    options:
      fluentd-address: 10.1.49.138:24224
      tag: "docker.usercenter.{{.ID}}"
      fluentd-async-connect: "true"
```

2.3 将docker-compose.yml 文件放到/data下面

2.4 必须在/data下面执行创建容器并启动服务的命令: docker-compose up d , 或者在任意目录下执行, 但是需要制定docker-compose.yml文件位置

2.5 查看docker是否成功启动: docker ps

```
[root@HIK-YUNS-PCTST1 scripts]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
961f28140c44	jdk83	"java -jar -Xms1g -X..."	2 days ago	Up 6 hours		hikvision-mall
b40b02b415c5	jdk81	"java -Xms2g -Xmx2g ..."	2 months ago	Up 3 minutes		b2bmember2
8d9842ac3ca9	jdk81	"java -jar -Xms2g -X..."	2 months ago	Up 2 weeks		cloud-merchants-product

4.排查问题

1. 查看系统运行日志命令

1.1 控制台日志

```
[root@HIK-YUNS-PCTST1 scripts]# docker logs -f --tail 200 hikvision-mall
2019-06-13 21:56:32.349 DEBUG [http-nio-8088-exec-3] com.hikvision.mall.server.dao.SalesRelationshipMapper.isSubUser Line:159 - <==
2019-06-13 21:56:32.350 DEBUG [http-nio-8088-exec-6] com.hikvision.mall.server.dao.InventoryMapper.selectInventory Line:159 - <==
2019-06-13 21:56:32.350 DEBUG [http-nio-8088-exec-4] com.hikvision.mall.server.dao.InventoryOrderMapper.selectInventorySubmitStatus Li
tock_amount, invoking.gmt_create, invoking.gmt_modify, ord.iid, ord.approve_status, ord.approve_msg FROM inventory_order AS ord LEFT J
mit 1
```

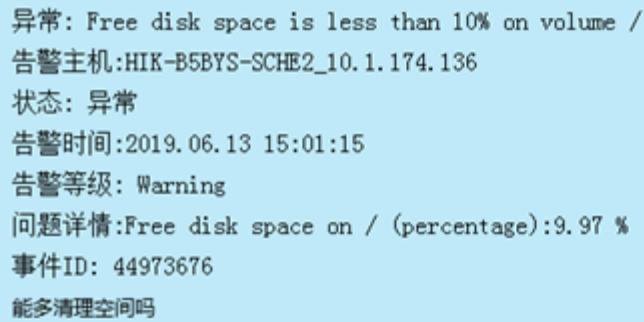
1.2 高级日志查看手段

```
[root@HIK-TST-APPSE01 hikappservice-1]# docker-compose logs -f --tail 200
Attaching to hikappservice2, hikappservice1
hikappservice2 | at org.springframework.web.filter.DelegatingFilterProxy.doFilter(DelegatingFilterProxy.java:262)
hikappservice2 | at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:193)
hikappservice2 | at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:166)
hikappservice2 | at org.springframework.web.filter.RequestContextFilter.doFilterInternal(RequestContextFilter.java:99)
hikappservice2 | at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:107)
hikappservice2 | at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:193)
hikappservice2 | at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:166)
hikappservice2 | at org.springframework.web.filter.HttpPutFormContentFilter.doFilterInternal(HttpPutFormContentFilter.java:108)
hikappservice2 | at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:107)
hikappservice2 | at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:193)
hikappservice1 | at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHa
hikappservice2 | at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:166)
hikappservice1 | at org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:8
```

1.3 本地文件查看日志

```
[root@HIK-TST-APPSE01 log_8089]# tail -fn 200 log_info.log
===2019-06-13 15:48:01.036 INFO com.hikvision.util.okhttp.UserPostUtil Line:99 - 请求成功http://10.1.44.119:9087/hikuser
"username":"17691076337","pwdIsUnify":1,"system":"APP","token":"e7b6e8b18432b808ffblc583bf887549","tokenRestTime":"1209600
===2019-06-13 15:48:01.037 INFO com.hikvision.util.okhttp.UserPostUtil Line:55 - 此次请求总时长: 35ms; 请求用户中心: htt
===2019-06-13 15:48:01.048 INFO com.hikvision.exception.AfterReturnAspect Line:55 - source:127.0.0.1:45234/hikappservice
阿拉善盟阿拉善左旗bbbb","b2BExist":0,"cityCode":"000000000001","cityName":"阿拉善盟","cloudUserId":"96070","company":"aaaa",
"17691076337","nickName":"xiaotiantian","phone":"17691076337","preToken":"07b2010bb9d63afa2c71a4eef5750aac","provinceCode"
stTime":"1209600","usertype":"1"},"message":"登录成功"}
===2019-06-13 15:54:37.001 INFO DEPRECATED Line:84 (module/moduleCount
```

2. 此种日志方式的坑



```
ens192: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.1.174.178 netmask 255.255.255.0 broadcast 10.1.174.255
    inet6 fe80::250:56ff:fea7:9d49 prefixlen 64 scopeid 0x20<link>
    ether 00:50:56:a7:9d:49 txqueuelen 1000 (Ethernet)
    RX packets 108014356 bytes 83276432277 (77.5 GiB)
    RX errors 0 dropped 66353 overruns 0 frame 0
    TX packets 63923351 bytes 12913355832 (12.0 GiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1 (Local Loopback)
    RX packets 318 bytes 23556 (23.0 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 318 bytes 23556 (23.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[root@HIK-SerAPP-SOLCTER2 05a9e5b5a3d29362732612f6baa0791a43a8d92c97ad6c769103b4e9faliae040]# pwd
/var/lib/docker/containers/05a9e5b5a3d29362732612f6baa0791a43a8d92c97ad6c769103b4e9faliae040
[root@HIK-SerAPP-SOLCTER2 05a9e5b5a3d29362732612f6baa0791a43a8d92c97ad6c769103b4e9faliae040]# ll
total 32159516
-rw-r-----. 1 root root 32394548579 Jun 13 22:59 05a9e5b5a3d29362732612f6baa0791a43a8d92c97ad6c769103b4e9faliae040-json.log
drwxr-xr-x. 2 root root 4096 Mar  9 22:22 checkpoints
```

log文件内容如下图

```
[root@HK-SerAPP-SOLCTER2 05a9e5b3ad29362732612f6baa0791a43a8d9c297ad6c769103b4e9fa1ae040]# more 05a9e5b3ad29362732612f6baa0791a43a8d9c297ad6c769103b4e9fa1ae040.json.log
{"log": "2019-05-27 10:56:51.354 [http-nio-10082-exec-7] INFO com.hikvision.core.interceptor.UserInterceptor - getUser by redis :{key:c1547397fd6398056294b550321fd613\n",
"log": "2019-05-27 10:56:51.355 [http-nio-10082-exec-7] INFO com.hikvision.core.interceptor.UserInterceptor - getUser by redis {userStr:{\"address\": \"江苏省南京市玄武区\\\",
玄武区少年宫中学经管部\", \"createTime\": 1524805382000, \"deviceType\": \"Android\", \"enable\": 1, \"icon\": \"https://hikbusiness.hikvision.com/upload/app/sncode/2019/05/27/0196
name\": \"15951819620\", \"openId\": \"0NWC4uQ_nI0dZTZW_p8vbmK-ho\", \"phone\": \"15951819620\", \"provinceCode\": \"100\", \"provinceName\": \"江苏\", \"pwd\": \"21c3fi8a7ac308f48e
\", \"userType\": \"经销商\", \"stream\": \"stdout\", \"time\": \"2019-05-27T02:56:51.3562828pwsd\"}"}
{"log": "2019-05-27 10:56:51.398 [http-nio-10082-exec-7] INFO R_HTTP_LOG - http-nio-10082-exec-7 - 764122 - 接口调用请求: API Http Request - Url : {https://mall.hikvision.com
:56:51 | Body : { | User : {id:764122,userName:15951819620,mobile:15951819620,nickName:null} \\n\", \"stream\": \"stdout\", \"time\": \"2019-05-27T02:56:51.3989152592\"}"}
{"log": "2019-05-27 10:56:51.398 [http-nio-10082-exec-7] INFO R_HTTP_LOG - http-nio-10082-exec-7 - 764122 - 返回信息: {\"code\": 1, \"message\": \"\\\"success\\\"\"}\\n\", \"stream\": \"stdout\"}
{"log": "2019-05-27 10:56:51.399 [http-nio-10082-exec-7] INFO ACCESS_LOG - http-nio-10082-exec-7 - 764122 - 访问接口:10.1.32.64:38734/app_merchants/package/order/counts;请求
\"message\": \"\\\"success\\\"\"}\\n\", \"stream\": \"stdout\", \"time\": \"2019-05-27T02:56:51.399606065Z\"}"}
{"log": "2019-05-27 10:56:52.031 [http-nio-10082-exec-1] INFO com.hikvision.core.interceptor.UserInterceptor - RequestURI : /app_merchants/collection/quantity/total/get | Met
rk: rkey:d8973ad8b45fedf5d672714cf8bb8 | cookie-rk rkey:d8973ad8b45fedf5d672714cf8bb8\\n\", \"stream\": \"stdout\", \"time\": \"2019-05-27T02:56:52.0323499162\"}"}
{"log": "2019-05-27 10:56:52.031 [http-nio-10082-exec-1] INFO com.hikvision.core.interceptor.UserInterceptor - RequestURI : /app_merchants/collection/quantity/total/get | Met
rk: rkey:d8973ad8b45fedf5d672714cf8bb8 | cookies=JSESSIONID=780AF012711371712705C890C0293FDD\\n\", \"stream\": \"stdout\", \"time\": \"2019-05-27T02:56:52.032364017\"}"}

```

从上图可知其实log内容就是 (docker logs f -tail 200 容器名) 命令看到的控制台内容

3. 解决方案

临时解决方案echo ">05a9e5b5a3d29362732612f6baa0791a43a8d92c97ad6c769103b4e9fa1ae040-json.log

永久解决方案：日志引擎

4. 日志引擎

1. json-file 默认形式配置如下图

```
logging:
  driver: "json-file"
  options:
    max-size: "1g"
```

必需配置max-size来限制上图中的最大日志大小，否则会无限叠加

优点是可以通过docker logs 或者docker-compose 来查看实时日志

2. EFK

```
logging:
  driver: "fluentd"
  options:
    fluentd-address: 10.1.49.138:24224
    tag: "docker.usercenter.{{.ID}}"
    fluentd-async-connect: "true"
```

优点便于在ES search上搜索日志

缺点：无法通过docker-compose查看实时日志

3. 系统日志

直接查看进程日志：journalctl -f -u docker.service

```
[root@HIK-SRV-UC2 sbin]# cd sbin/packet_write_wait: Connection to 10.1.174.110 port 22: Broken pipe
[root@HIK-TST-APPSE01 apollo]# journalctl -f -u docker.service
-- Logs begin at Sat 2019-01-19 10:29:03 CST. --
Jun 04 14:28:39 HIK-TST-APPSE01 dockerd[676384]: time="2019-06-04T14:28:39.857381639+08:00" level=error msg="Error closing logger: invalid argument"
Jun 04 16:18:03 HIK-TST-APPSE01 dockerd[676384]: time="2019-06-04T16:18:03.820954868+08:00" level=error msg="Handler for GET /v1.25/images/jdk83/json
Jun 04 16:18:06 HIK-TST-APPSE01 dockerd[676384]: time="2019-06-04T16:18:06.810673697+08:00" level=error msg="Not continuing with pull after error: err
Jun 04 16:18:06 HIK-TST-APPSE01 dockerd[676384]: time="2019-06-04T16:18:06.810760317+08:00" level=info msg="Ignoring extra error returned from registr
Jun 04 16:18:06 HIK-TST-APPSE01 dockerd[676384]: time="2019-06-04T16:18:06.822398113+08:00" level=info msg="Translating \"denied: requested access to
Jun 04 16:18:06 HIK-TST-APPSE01 dockerd[676384]: time="2019-06-04T16:18:06.822541301+08:00" level=error msg="Handler for POST /v1.25/images/create reti
Jun 12 14:47:18 HIK-TST-APPSE01 dockerd[676384]: time="2019-06-12T14:47:18.627773190+08:00" level=error msg="Handler for GET /v1.27/images/get returne
Jun 12 15:14:12 HIK-TST-APPSE01 dockerd[676384]: time="2019-06-12T15:14:12.484324684+08:00" level=error msg="Handler for POST /v1.22/containers/create
Jun 12 15:15:29 HIK-TST-APPSE01 dockerd[676384]: time="2019-06-12T15:15:29.940851606+08:00" level=error msg="Handler for POST /v1.22/containers/create
Jun 13 09:45:14 HIK-TST-APPSE01 dockerd[676384]: time="2019-06-13T09:45:14.739004527+08:00" level=info msg="Container 83c1547da540a73bacbfd96cb7d10c6a
```

默认只能查看error级别日志，基本看不出来问题，所以需要改变日志级别

```
#vi /etc/docker/daemon.json
{
  "insecure-registries": ["192.168.50.50:5000"],
  "debug": true, //debug级别日志
}
#systemctl daemon-reload
#systemctl restart docker
```

systemctl daemon-reload 是保存进程当前的现状，否则直接重启docker会报错，好处是docker restart后所有的服务会自动启动

debug级别可以用来调试docker本身的bug，比如docker pull images 拉不下来可以看到原因

5. 镜像原理

1. jdk制作镜像

```
FROM centos:7.4.1708
MAINTAINER hikvision

COPY sources.list /etc/apt/sources.list

RUN yum update -y && yum makecache

RUN rm -rf /etc/localtime && ln -s /usr/share/zoneinfo/Asia/Shanghai /etc/localtime #修改时区
RUN yum -y install kde-l10n-Chinese && yum -y reinstall glibc-common #安装中文支持
RUN localedef -c -f UTF-8 -i zh_CN zh_CN.utf8 #配置显示中文

ADD jdk-8u161-linux-x64.tar.gz /usr/local/java

ADD server_new.crt /usr/local/java/jdk1.8.0_161/jre/bin

ADD server_new.crt /usr/local/java/jdk1.8.0_161/jre/lib/security

ENV TZ Asia/Shanghai
ENV LANG zh_CN.UTF-8
ENV LANGUAGE zh_CN:zh
ENV LC_ALL zh_CN.UTF-8
ENV JAVA_HOME /usr/local/java/jdk1.8.0_161
ENV CLASSPATH $JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
ENV PATH $PATH:$JAVA_HOME/bin

WORKDIR /usr/local/java/jdk1.8.0_161/jre/bin

RUN keytool -import -v -trustcacerts -alias server -file server_new.crt -noprompt -storepass changeit -keystore /usr/local/java/jdk1.8.0_161/jre/lib/security/cacerts
```

2. 指令解释

dockerfile构建说明，所有需要的文件都需要放到上下文路径中，不然找不到，指定上下文路径为，即当前目录下新建目录 /tmp/dockerDockerFile

创建Dockerfile

创建镜像 docker build -t nginx:v3 . //最后面的.指定上下文路径，所有的文件复制都需要相对路径，构建会将目录下面所有的文件都复制到镜像中，所以不要放多余的东西，否则镜像会很大

COPY指令 COPY hom* /mydir 支持通配符，hom*表示上下文路径 /mydir 是容器内的绝对路径

ADD指令，同复制，但是源文件可以是远程文件或者压缩文件，为压缩文件的时候会自动解压（主要用途）

CMD 指定进程的启动命令，即容器就是一个进程，控制容器的启动命令

一般容器会有默认的CMD为 /bin/bash

VOLUME 定义匿名卷 VOLUME /data 或者 VOLUME ['/data', '/var'] 匿名卷可以写入 没有指定卷的动态文件

docker run -d -v mydata:/data 容器ID // -v mydata:/data是指定/data的非匿名卷会覆盖掉匿名卷

EXPOSE '端口' 这个只是申明这个端口，方便开发查看，然后真正起作用的是在启动的时候通过 -p <宿主端口>:<容器端口> 来映射端口

dockerfile中的每个run 都相当于一个全新的容器，只不过是上一个容器集成而来，所以两个RUN的运行环境不搭嘎

WORKDIR 是指定所有RUN都在一个目录下执行命令

RUN指定是编译镜像时的动作是编译期 可以有多个

CMD 指定启动容器时默认行为或启动时执行的命令 示例 CMD echo "this is a test" 一个dockerfile只能有一个CMD

ENTRYPOINT在容器启动时，可以接受命令行里面的参数 和CMD一样会覆盖默认的容器启动指令 启动容器内的服务用

3. docker构建

docker build -t jdk8/hikvisionmall-crt:latest .

必须在dockerfile文件所在的目录下执行，默认工作区，会把此路径下所有的文件复制进镜像所在工作区中，工作区为所有命令执行的目录

4. 镜像本质

解压镜像后文件如下图：

名称	修改日期
7da9afeee1c25ec58d3db492e24927363c2f714d6950ac9dc6843a774d24248b	2019/6/17 19:36
046a4f911021e36459851699931e030f00d8975a985c41ca5961d4c6a22c1215	2019/6/17 19:36
62a43704da3dfaaa0a1a921aa76027632cc9dcc94f5ca10199b93f7ccf7767fa	2019/6/17 19:36
62f5929382bce3c981730ed3342f693f91a4feaедcdc8f3627fa5865def2c0d2	2019/6/17 19:45
304f7e447a81330422dd6337d597c2017e35c79d6f7600bcd81f84671ed09157	2019/6/17 19:36
314b10b61467aecdc7294fbc887298ad6c3859dac48146219b9f2d5c6a760b68	2019/6/17 19:36
480f0dd4d84bf727cbd968fbefdc1da832cb72f465c06938e2921bee65af3e17	2019/6/17 19:36
a62f45debe57e619702455660fb17d7d57f8bfb75fac42ce7cc82d0fb35fe622	2019/6/17 19:36
cc2885cbfa9068dd206dec7db33b6dcc23fddaa0c291b69dabf401824b9a66e5	2019/6/17 19:36
e552ff231c07a54e706026437ede1221ad0874c8405fce2e74161d7b935d361a	2019/6/17 19:36
3f44019a234c04b3179628e1d7a5f9bc4cea3be236bad94c249a271e6a1e0f1b.json	2019/3/13 20:52
manifest.json	
repositories	

下图是镜像说明文件：

```

{
  "Config": "3f44019a234c04b3179628e1d7a5f9bc4cea3be236bad94c249a271e6a1e0f1b.json",
  "RepoTags": ["jdk83:latest"],
  "Layers": [
    "62f5929382bce3c981730ed3342f693f91a4feaедcdc8f3627fa5865def2c0d2/layer.tar",
    "e552ff231c07a54e706026437ede1221ad0874c8405fce2e74161d7b935d361a/layer.tar",
    "480f0dd4d84bf727cbd968fbefdc1da832cb72f465c06938e2921bee65af3e17/layer.tar",
    "a62f45debe57e619702455660fb17d7d57f8bfb75fac42ce7cc82d0fb35fe622/layer.tar",
    "314b10b61467aecdc7294fbc887298ad6c3859dac48146219b9f2d5c6a760b68/layer.tar",
    "304f7e447a81330422dd6337d597c2017e35c79d6f7600bcd81f84671ed09157/layer.tar",
    "62a43704da3dfaaa0a1a921aa76027632cc9dcc94f5ca10199b93f7ccf7767fa/layer.tar",
    "046a4f911021e36459851699931e030f00d8975a985c41ca5961d4c6a22c1215/layer.tar",
    "7da9afeee1c25ec58d3db492e24927363c2f714d6950ac9dc6843a774d24248b/layer.tar",
    "cc2885cbfa9068dd206dec7db33b6dcc23fddaa0c291b69dabf401824b9a66e5/layer.tar"
  ]
}

```

根 layer 内容
















images	jdk83	62f5929382bce3c981730ed3342f693f91a4feaедcdc8f3627fa5865def2c0d2	
共享 新建文件夹			
名称	修改日期	类型	大小
VERSION	2019/3/13 20:52	文件	1 KB
layer.tar	2019/3/13 20:52	WinRAR 压缩文件	200,331 KB
json	2019/3/13 20:52	文件	1 KB
layer	2019/6/17 19:45	文件夹	

非顶层json说明文件案例：

```
repositoriesX jsonX 3f44019a234c04b3179628e1d7a5f9bc4cea3be236bad94c249a271e6a1e0f1b.jsonX jsonX jsonX
1 {
2   "id": "304f7e447a81330422dd6337d597c2017e35c79d6f7600bcd81f84671ed09157",
3   "parent": "314b10b61467aecdc7294fbc887298ad6c3859dac48146219b9f2d5c6a760b68",
4   "created": "2019-03-13T12:52:10.122039821Z",
5   "container_config": {
6     "Hostname": "",
7     "Domainname": "",
8     "User": "",
9     "AttachStdin": false,
10    "AttachStdout": false,
11    "AttachStderr": false,
12    "Tty": false,
13    "OpenStdin": false,
14    "StdinOnce": false,
15    "Env": null,
16    "Cmd": null,
17    "Image": "",
18    "Volumes": null,
19    "WorkingDir": "",
20    "Entrypoint": null,
21    "OnBuild": null,
22    "Labels": null
23  }
24 }
```

解压layer看看

jdk83 ▶ 62f5929382bce3c981730ed3342f693f91a4feaedcdc8f3627fa5865def2c0d2 ▶ layer ▶

新建文件夹			
名称	修改日期	类型	大小
 dev	2019/6/17 19:45	文件夹	
 etc	2019/6/17 19:45	文件夹	
 home	2019/6/17 19:45	文件夹	
 lost+found	2019/6/17 19:45	文件夹	
 media	2019/6/17 19:45	文件夹	
 mnt	2019/6/17 19:45	文件夹	
 opt	2019/6/17 19:45	文件夹	
 proc	2019/6/17 19:45	文件夹	
 root	2019/6/17 19:45	文件夹	
 run	2019/6/17 19:45	文件夹	
 srv	2019/6/17 19:45	文件夹	
 sys	2019/6/17 19:45	文件夹	
 tmp	2019/6/17 19:45	文件夹	
 usr	2019/6/17 19:45	文件夹	
 anaconda-post.log	2017/9/11 23:48	文本文档	16

总结:

每次执行 FROM COPY RUN ADD 这些指令都会新建一个layer,

跟据命令执行的顺序形成单向依赖的链, 后执行的命令依赖它之前的层

利用镜像创建容器即执行所有的指令, 形成最终的工作区, 即在简化的linux中安装运行jar包所需要的软件和环境

综上: 镜像的目的是构建一个能运行最终服务所依赖的linux或其他操作系统的环境

5. 镜像仓库

公司镜像仓库地址: <http://habor.hikvision.com.cn>

1. 镜像仓库需要登录才能push镜像, 所以先登录: `docker login habor.hikvision.com.cn` 然后输入用户名和密码 (如有需求找我要)

2. 为当前服务器已经存在的镜像 (`docker images -a` 可看到) 重命名 (tag即docker重命名的方式)

`docker tag jdk8/hikvisionmall-crt habor.hikvision.com.cn/cloud-merchants/jdk8/hikvisionmall-crt:latest`

命令解释: `jdk8/hikvisionmall-crt`为本地 (`docker images -a` 能看到的) 镜像

`habor.hikvision.com.cn`前缀是固定值, 向habor中推送镜像必须的前缀, habor文档所规定

`cloud-merchants` 是habor中建立的项目名称,

`cloud-merchants/jdk8/hikvisionmall-crt:latest` 为镜像名称: 版本号

推送镜像:

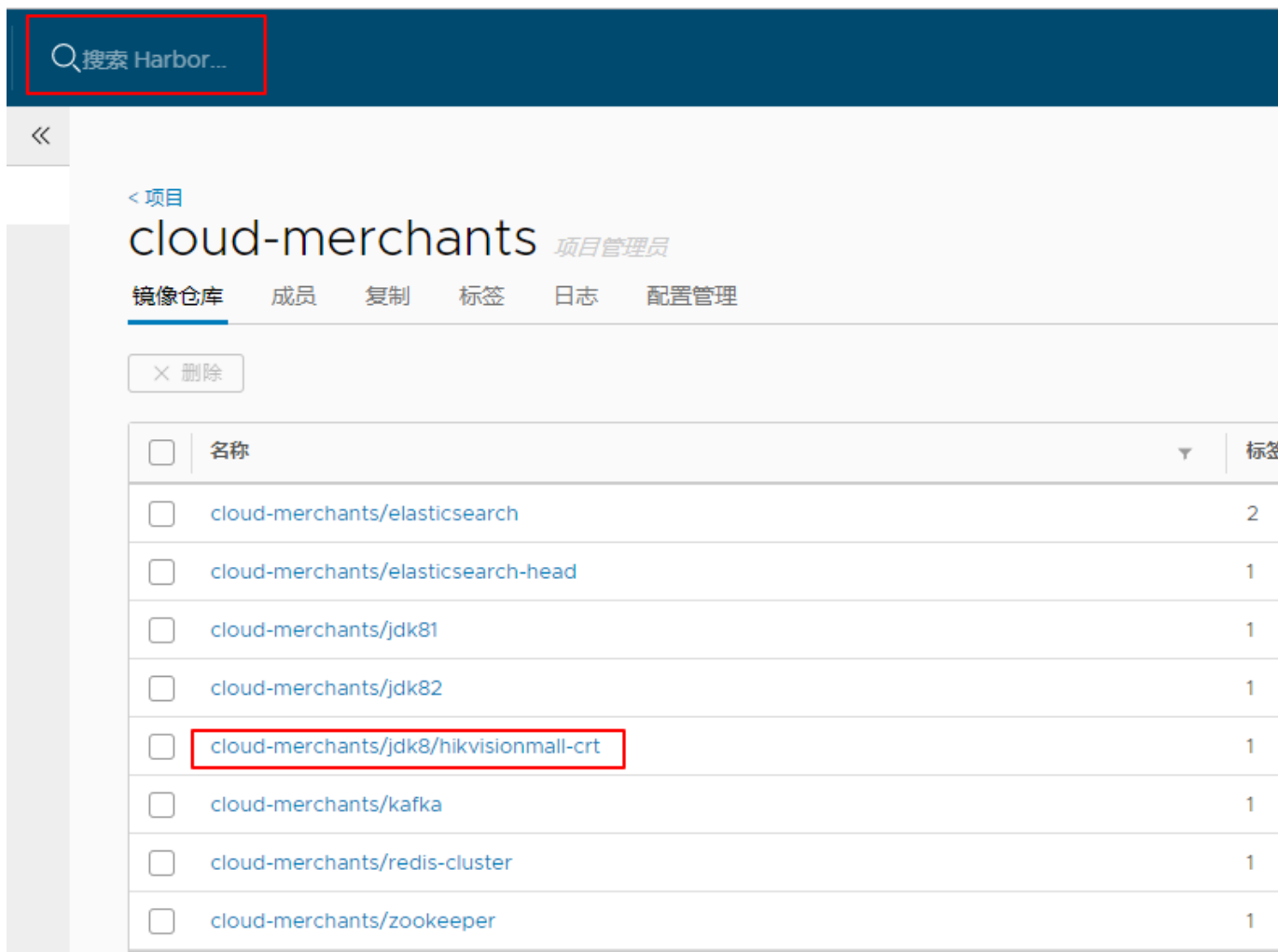
`docker push habor.hikvision.com.cn/cloud-merchants/jdk8/hikvisionmall-crt`

拉取镜像

`docker pull habor.hikvision.com.cn/cloud-merchants/jdk8/hikvisionmall-crt`

habor示意图





6. 镜像生成容器并启动一个服务

1. 容器和服务

服务的本质是为客户提供有价值的内涵，例如：提供接口服务，nginx负载服务，mysql数据库服务

容器是提供服务运行所必须的环境

2. 容器内部一探究竟

`docker exec it 容器名称 /bin/bash`，进入上面创建的第一个容器

`docker exec -it hikusercenter bash` `// bash = /bin/bash`

```
[root@B2B-TST07 docker-hikusercenter]# docker exec -it hikusercenter1 bash
[root@B2B-TST07 hikusercenter]# pwd
/usr/local/hikusercenter
[root@B2B-TST07 hikusercenter]# cd /
[root@B2B-TST07 /]# ll
total 20
-rw-r--r--. 16 root root 15836 9月 11 2017 anaconda-post.log
lrwxrwxrwx. 1 root root 7 5月 24 2018 bin -> usr/bin
dr-xr-xr-x. 2 root root 6 4月 11 2018 boot
drwxr-xr-x 5 root root 340 6月 11 17:01 dev
drwxr-xr-x 1 root root 66 4月 4 09:43 etc
drwxr-xr-x. 2 root root 6 11月 5 2016 home
lrwxrwxrwx. 1 root root 7 5月 24 2018 lib -> usr/lib
lrwxrwxrwx. 1 root root 9 5月 24 2018 lib64 -> usr/lib64
drwx----- 2 root root 6 9月 11 2017 lost+found
drwxr-xr-x. 2 root root 6 11月 5 2016 media
drwxr-xr-x. 2 root root 6 11月 5 2016 mnt
drwxr-xr-x. 2 root root 6 11月 5 2016 opt
dr-xr-xr-x 212 root root 0 6月 11 17:01 proc
drwxr-xr-x 3 root root 20 4月 4 09:43 project
dr-xr-x--- 1 root root 52 6月 17 21:27 root
drwxr-xr-x. 11 root root 143 7月 30 2018 run
lrwxrwxrwx. 1 root root 8 5月 24 2018/sbin -> usr/sbin
drwxr-xr-x. 2 root root 6 11月 5 2016 srv
dr-xr-xr-x 13 root root 0 11月 22 2018 sys
drwxrwxrwt. 1 root root 4096 6月 11 17:01 tmp
drwxr-xr-x. 1 root root 19 5月 24 2018 usr
drwxr-xr-x. 18 root root 238 7月 30 2018 var
```

容器在物理机上的文件映射，默认在/var/lib下面

文件1:/var/lib/docker/containers/* 容器运行时文件

```
[root@B2B-TST07 664742a53f2c071dc8e181e0bfec9e2bfec999854475e74d9a46305333c602b]# docker ps
CONTAINER ID        IMAGE               COMMAND             STATUS              PORTS              NAMES
664742a53f2c        jdk81              "java -jar -Xms1g ..." 4 weeks ago         Up 3 weeks          api
29f482b3317b        jdk81              "java -jar -Xms1g ..." 4 weeks ago         Up 4 weeks          appservice
3cfccc24c249        jdk81              "java -jar -Xms2g ..." 6 weeks ago         Restarting (1) 3 days ago  cloudsaaureka
b714ba0b63c7        jdk8               "java -jar -Xms2g ..." 2 months ago        Up 6 days          hikusercenter2
83ae68a7b365        jdk8               "java -jar -Xms2g ..." 2 months ago        Up 6 days          hikusercenter1
994792e0f90a        jdk81              "java -jar -Xms1g ..." 2 months ago        Up 2 months        security
fca7f09d0259        jdk81              "java -jar -Xms1g ..." 3 months ago        Up 3 months        web
00eb92c61e9b        jdk81              "java -jar -Xms1g ..." 3 months ago        Up 5 weeks         mallsec
594766dc4844        jdk81              "java -jar -Xms1g ..." 3 months ago        Up 3 months        eureka
31a4dd0b5c8d        mysql:5.7          "docker-entrypoint..." 9 months ago        Up 6 months        mysql_slave
e588aca51906        mysql:5.7          "docker-entrypoint..." 9 months ago        Up 6 months        mysql_master
378ea0844a6e        mysql:5.7          "docker-entrypoint..." 9 months ago        Up 6 months        mysql
01c4e4384a3d        hikappservice/cluster-redis:4.0.10 "/usr/local/redis/..." 9 months ago        Up 6 months        hikapp_redis7002_1
104e4486d68f        hikappservice/cluster-redis:4.0.10 "/usr/local/redis/..." 9 months ago        Up 6 months        hikapp_redis7001_1
7b4edec6ebbb        hikappservice/cluster-redis:4.0.10 "/usr/local/redis/..." 9 months ago        Up 6 months        hikapp_redis7003_1
[root@B2B-TST07 664742a53f2c071dc8e181e0bfec9e2bfec999854475e74d9a46305333c602b]# pwd
/var/lib/docker/containers/664742a53f2c071dc8e181e0bfec9e2bfec999854475e74d9a46305333c602b
[root@B2B-TST07 664742a53f2c071dc8e181e0bfec9e2bfec999854475e74d9a46305333c602b]# ll
总用量 65608
-rw-r----- 1 root root 67156100 6月 17 21:55 664742a53f2c071dc8e181e0bfec9e2bfec999854475e74d9a46305333c602b-.json.log
drwx----- 2 root root 6 5月 15 20:17 checkpoints
-rw-r----- 1 root root 4312 5月 23 15:29 config.v2.json
-rw-r----- 1 root root 1344 5月 23 15:29 hostconfig.json
-rw-r----- 1 root root 10 5月 23 15:29 hostname
-rw-r----- 1 root root 158 5月 23 15:29 hosts
-rw-r----- 1 root root 72 5月 23 15:29 resolv.conf
drwxrwxrwt 2 root root 40 5月 23 15:29 tmp
[root@B2B-TST07 664742a53f2c071dc8e181e0bfec9e2bfec999854475e74d9a46305333c602b]#
```

文件2:/var/lib/docker/overlay

这里存放的是镜像的每一层layer解压后的结果，以及基于每一个镜像生成容器后，对镜像合并挂载后的目录和对应的init目录

docker ps -a 的容器数量等于此文件下的xxx-init的文件数量，容器创建(docker run)和销毁 (docker rm)时会创建和删除对应的文件

xxx-init为容器起始文件，里面的内容为顶层layer的所在目录，然后按照 依赖链可以找到所有的layer

每个容器 docker inspect 容器名称，可以看到init文件目录名称，即下图中的MergedDir,LowerDir为顶层layer的目录名称

```

"GraphDriver": {
  "Name": "overlay",
  "Data": {
    "LowerDir": "/var/lib/docker/overlay/3446685d59390dbe2c37ecc9a87da0384234ed3900780d5680dda3e203006293/root",
    "MergedDir": "/var/lib/docker/overlay/9015655c6ee62d2cdfd280ba0e8325deff462a7f5c65b5ba5d7f73a834edf567/merged",
    "UpperDir": "/var/lib/docker/overlay/9015655c6ee62d2cdfd280ba0e8325deff462a7f5c65b5ba5d7f73a834edf567/upper",
    "WorkDir": "/var/lib/docker/overlay/9015655c6ee62d2cdfd280ba0e8325deff462a7f5c65b5ba5d7f73a834edf567/work"
  }
},

```

文件案例

```

[root@B2B-TST07 e9a0a0d1e66f8e6cc3cea32650ac983daa58c86b5040b1c7de10be1249108fad-init]# ll
总用量 4
-rw-r--r-- 1 root root 64 3月  5 14:15 lower-id
drwx----- 2 root root  6 3月  5 14:15 merged
drwxr-xr-x 4 root root 46 3月  5 14:15 upper
drwx----- 3 root root 18 3月  5 14:15 work
[root@B2B-TST07 e9a0a0d1e66f8e6cc3cea32650ac983daa58c86b5040b1c7de10be1249108fad-init]# more lower-id
aaecab8c1a03ef2c2838c896df8c1a71b95d258b2a1ae65a3a0aad48eababbf6
[root@B2B-TST07 e9a0a0d1e66f8e6cc3cea32650ac983daa58c86b5040b1c7de10be1249108fad-init]# cd ../aaecab8c1a03ef2c2838c896df8c1a71b95d258b2a1ae65a3a0aad48eababbf6
[root@B2B-TST07 aaecab8c1a03ef2c2838c896df8c1a71b95d258b2a1ae65a3a0aad48eababbf6]# ll
总用量 0
drwxr-xr-x 18 root root 267 2月  20 21:11 root
[root@B2B-TST07 aaecab8c1a03ef2c2838c896df8c1a71b95d258b2a1ae65a3a0aad48eababbf6]# cd root/
[root@B2B-TST07 root]# ll
总用量 20
-rw-r--r-- 16 root root 15836 9月  11 2017 anaconda-post.log
lrwxrwxrwx 1 root root  7 12月 13 2018 bin -> usr/bin
dr-xr-xr-x 2 root root  6 4月  11 2018 boot
drwxr-xr-x 2 root root  6 9月  11 2017 dev
drwxr-xr-x 50 root root 4096 2月  20 21:11 etc
drwxr-xr-x 2 root root  6 11月  5 2016 home
lrwxrwxrwx 1 root root  7 12月 13 2018 lib -> usr/lib
lrwxrwxrwx 1 root root  9 12月 13 2018 lib64 -> usr/lib64
drwx----- 2 root root  6 9月  11 2017 lost+found
drwxr-xr-x 2 root root  6 11月  5 2016 media
drwxr-xr-x 2 root root  6 11月  5 2016 mnt
drwxr-xr-x 2 root root  6 11月  5 2016 opt
drwxr-xr-x 2 root root  6 9月  11 2017 proc
dr-xr-xr-x 3 root root 162 2月  13 16:54 root
drwxr-xr-x 12 root root 161 2月  20 21:11 run
lrwxrwxrwx 1 root root  8 12月 13 2018/sbin -> usr/sbin
drwxr-xr-x 2 root root  6 11月  5 2016 srv
drwxr-xr-x 2 root root  6 9月  11 2017 sys
drwxrwxrwt 8 root root 155 2月  13 16:54 tmp
drwxr-xr-x 13 root root 155 12月 13 2018 usr
drwxr-xr-x 18 root root 238 2月  20 21:11 var
[root@B2B-TST07 root]#

```