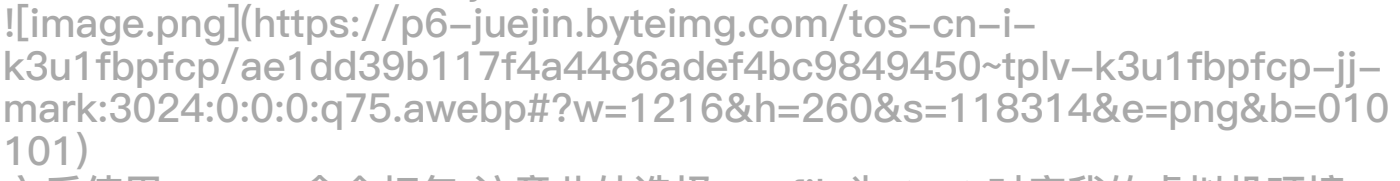
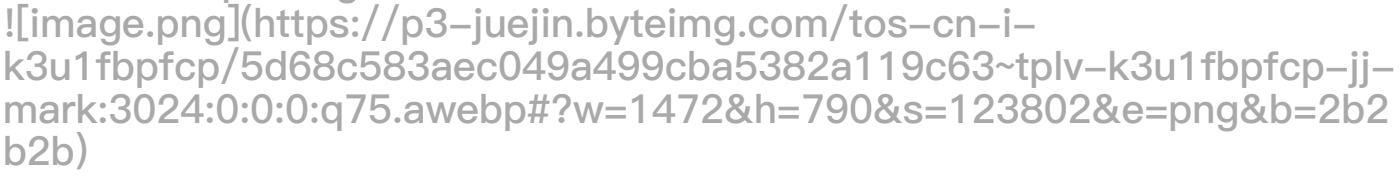
使用Dockerfile构建镜像 & 使用docker-compose 一键部署IM项目

本文讲解：使用Dockerfile构建镜像 & 使用docker-compose 一键部署IM项目。

im项目地址：[xzll-im](http://cxyroad.com/”https://github.com/598572/xzll-im”)，欢迎志同道合的开发者 一起 维护，学习，欢迎star

1、Dockerfile编写与镜像构建&容器运行

Dockerfile 是构建镜像的基础，首先创建两个空文件夹，用于存放im-connect和im-business的jar：

之后使用 maven命令打包 注意此处选择 profile为 test 对应我的虚拟机环境

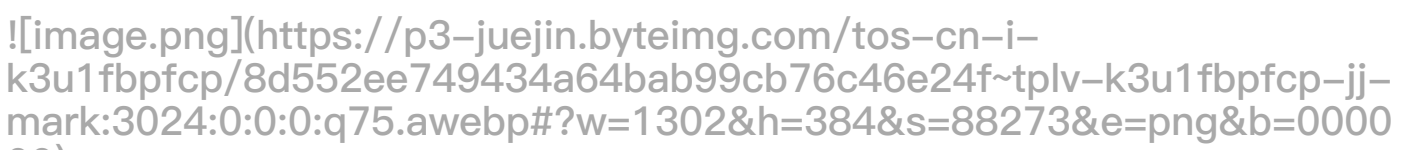
`mvn celan package -P test`:


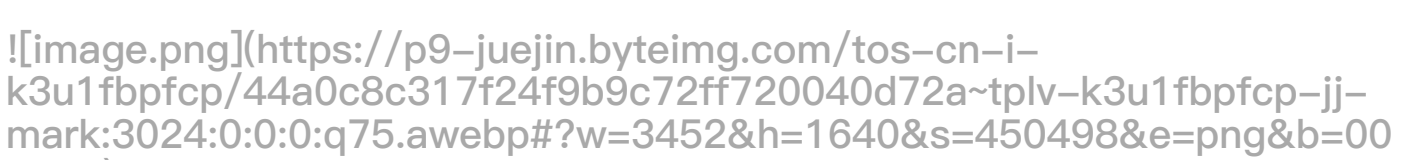
之后上传打好的jar到虚拟机:

```
sudo scp /Users/hzz/myself_project/im-开源04/xzll-im-parent/im-connect/im-connect-service/target/im-connect-service.jar root@172.30.128.65:/usr/local/soft_hzz/xzll-im/jar-file/docker-file-way/connect

sudo scp /Users/hzz/myself_project/im-开源04/xzll-im-parent/im-business/im-business-service/target/im-business-service.jar root@172.30.128.65:/usr/local/soft_hzz/xzll-im/jar-file/docker-file-
```

way/business

```

之后分别编写Dockerfile文件，内容如下：

```
business 的 Dockerfile 内容
FROM openjdk:11-jre-slim # jdk镜像
VOLUME /tmp # 挂载
COPY im-business-service.jar business.jar
ENTRYPOINT ["java", "-jar", "/business.jar"] # 启动命令
EXPOSE 8083 # 暴露端口 与服务端口保持一致

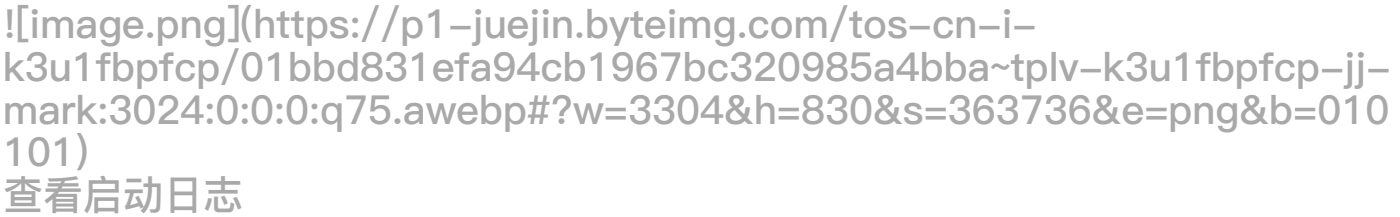
connect的 Dockerfile 内容
FROM openjdk:11-jre-slim # jdk镜像
VOLUME /tmp # 挂载
COPY im-connect-service.jar connect.jar
ENTRYPOINT ["java", "-jar", "/connect.jar"] # 启动命令
EXPOSE 10001 # 暴露端口 与服务端口保持一致
```



现在这俩服务的 Dockerfile就就绪了之后我们制作镜像：

```
. 代表当前目录 前边镜像名 后边版本号

docker build -t im-business:0.0.2 .
docker build -t im-connect:0.0.2 .
```


```

使用docker images看一下 有没有：

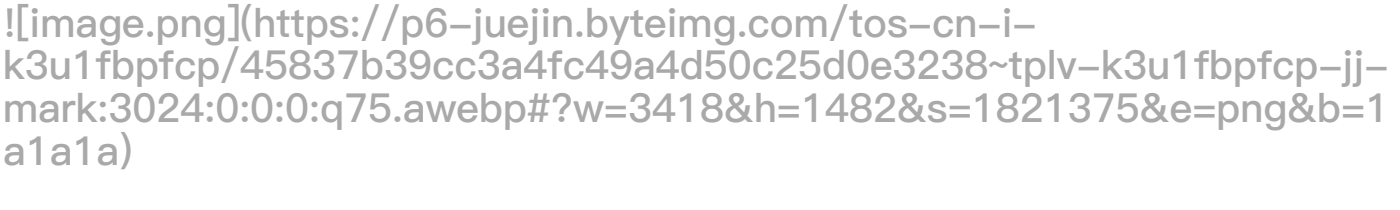
使用镜像启动容器：

```
docker run -d -p 8083:8083    --restart always --name im-business im-business:0.0.2
docker run -d -p 10001:10001   --restart always  --name im-connect im-connect:0.0.2
```



查看启动日志

```
docker logs -f --tail 50  im-business
docker logs -f --tail 50  im-connect
```



发消息试试能不能走通流程：

可以看到是没问题的。至此Dockerfile方式构建镜像 启动容器就说完了 但是这种方式比较繁琐，每一个服务都得搞个Dockerfile 服务多了头炸了。所以有了服务编排工具，常见的：

* docker-compose (姑且可以将其归纳为容器编排 但是论功能上 和k8s还是

差挺远的，k8s后续会用）
* k8s
* OpenShift
* Docker Swarm

下边就以docker-compose开刀 方便轻松管理多服务的情况！

2、docker-compose安装与容器编排
==========

为什么使用docker-compose

在上边我们介绍了使用 Dockerfile 构建 docker 镜像 然后 在镜像基础上启动应用程序，乍看起来已经能够满足我们的日常需求了，无论需要什么环境，在 Dockerfile 里逐步构建，然后 build、run，就 ok 了，也满足了我们docker 隔离性、快速部署的要求，为什么还需要docker-compose呢?

首先我们要知道docker 是轻量化的应用程序，docker 官方推荐每个 docker 容器中只运行一个进程，那么就是说，我们需要分别为我们的应用以及中间件创建单独的 docker 容器，然后分别启动它，想象一下，构建好 docker 之后，每次启动我们的网站，如果有n个服务 那么就得docker run n次，是不是很繁琐？而且此时这几个 docker 是分散独立的，很不方便管理，既然这几个 docker 都是为了同一个网站服务，是不是应该把它们放到一起？这就引出了 docker-compose 项目。

什么是docker-compose

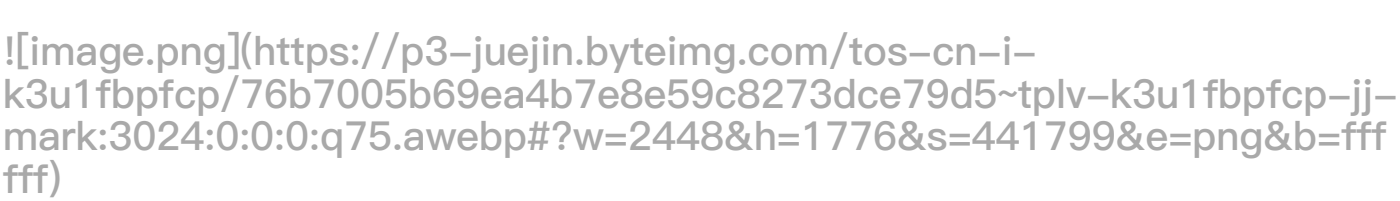
docker-compose是 docker 官方的开源项目，使用 python 编写，实现上调用了 Docker 服务的 API 进行容器管理，其官方定义为为：`定义和运行多个 Docker 容器的应用（Defining and running multi-container Docker applications））`，其实就是上面所讲的功能。

安装Docker-Compose

> Docker Compose是一个用来定义和运行复杂应用的Docker工具，一个使用Docker容器的应用，通常由多个容器组成。使用Docker Compose不再需要使用shell脚本来启动容器。

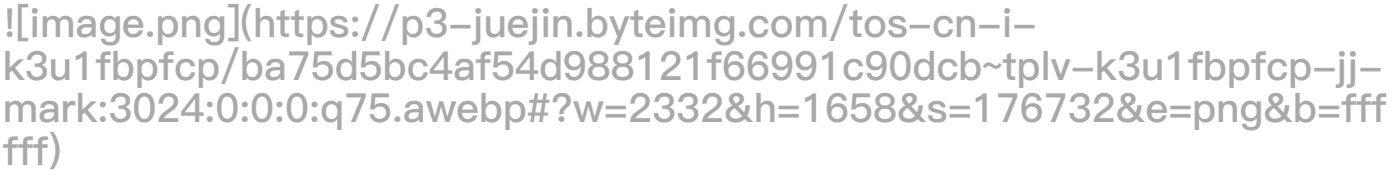
Compose 通过一个配置文件来管理多个Docker容器，在配置文件中，所有的容器通过services来定义，然后使用docker-compose脚本来启动，停止和重启应用，和应用中的服务以及所有依赖服务的容器，非常适合组合使用多个容器进行开发的场景

注意这个docker-compose和docker是有对应关系的，一定要在github找好关系，对应错了的话挺麻烦。
我用的docker版本是26.1.4 所以我要下载的docker-compose就是：v2.27.2



如果网络好的话 直接 下载：

```
sudo curl -L
"https://github.com/docker/compose/releases/download/v2.27.2/docke
r-compose-linux-x86_64" -o /usr/local/bin/docker-compose
```

但是我的虚拟机上网络很差 所以直接在宿主机下载好 然后上传到虚拟机。


上传到虚拟机的/usr/local/bin 目录：

```
sudo scp /Users/hzz/Downloads/docker-compose-linux-x86_64
root@172.30.128.65:/usr/local/bin
```

改个名叫docker-compose

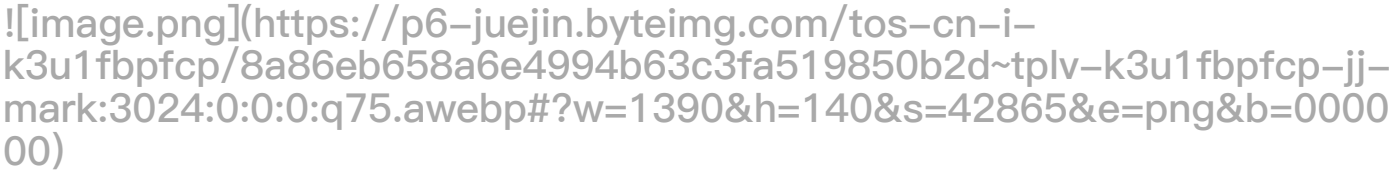
```
mv docker-compose-linux-x86_64 docker-compose
```

之后授权：

```
sudo chmod +x /usr/local/bin/docker-compose
```

之后验证下：

```
docker-compose -v
```



`docker-compose` 安装成功 ， 接下来开始docker-compose之旅。

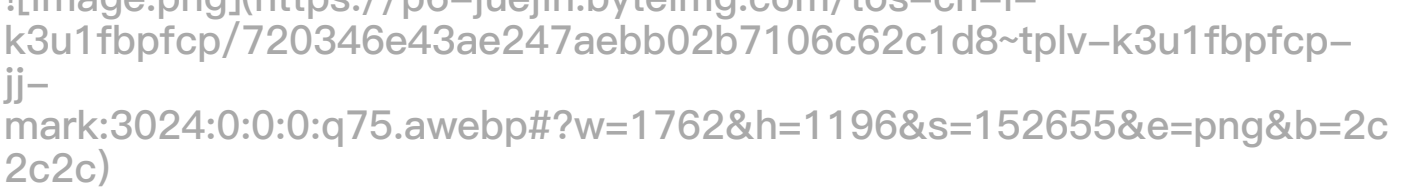
docker-compose.yml编写与批量上传

首先：
需要在项目根目录编写个docker-compose.yml文件 当然你vim方式在虚拟机上创建也行，但是为了直观我这里先在idea中创建，完事上传到虚拟机。

```
version: '3'
services:
  im-gateway:
    build: ./im-gateway
```

```
    hostname: im-gateway
    container_name: im-gateway
    restart: always
    ports:
      - "8081:8081"
    networks:
      - default_network
    volumes:
      - "/tmp/data/logs:/logs"
  im-auth:
    build: ./im-auth
    hostname: im-auth
    container_name: im-auth
    restart: always
    ports:
      - "8082:8082"
    networks:
      - default_network
    volumes:
      - "/tmp/data/logs:/logs"
  im-business:
    # 可以根据Dockerfile构建镜像（但是，Docker Compose 会在检测到上下文变化时重新构建镜像。也就是说如果你不修改Dockerfile docker-compose应该不是每次都构建镜像 实测确实如此）
    build: ./im-business
    # 也可以指定镜像名
    # image: im-business:0.0.2

    # 设置容器的主机名 即修改： /etc/hosts 中的内容
    hostname: im-business
    # 容器名称
    container_name: im-business
    # 重启策略， always 表示无论哪种状态退出，都会重启容器
    restart: always
    ports:
      # 设置主机与容器的端口映射
      - "8083:8083"
    networks:
      # 使用默认网络即：docker0 桥接
      - default_network
    volumes:
      # 将主机的 /tmp/data/logs 目录挂载到容器的 /logs 目录。这样可以实现数据的持久化，当容器重启时，数据不会丢失
      - "/tmp/data/logs:/logs"
  im-connect:
    build: ./im-connect
    hostname: im-connect
    container_name: im-connect
```

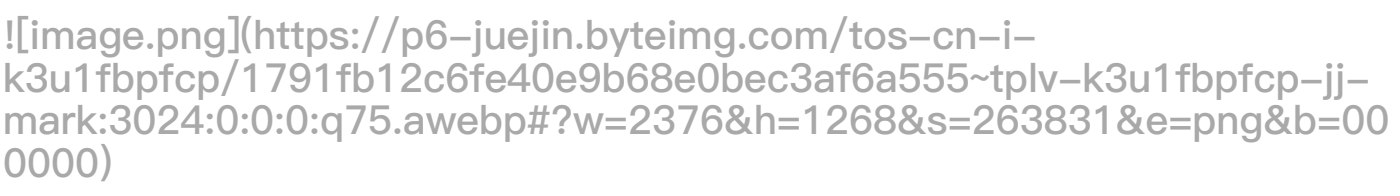
```
    restart: always
    ports:
      - "10001:10001"
    networks:
      - default_network
    volumes:
      - "/tmp/data/logs:/logs"
  im-console:
    build: ./im-console
    hostname: im-console
    container_name: im-console
    restart: always
    ports:
      - "8084:8084"
    networks:
      - default_network
    volumes:
      - "/tmp/data/logs:/logs"
networks:
  default_network:
    # 桥接
    driver: bridge
```

之后我们在每个项目的resource下创建对应的Dockerfile以便构建对应项目的镜像，如下是网关的：


接下来我们`mvn clean package -P test` 打包

现在jar包，Dockerfile docker-compose.yml都有了 剩下的工作就是将其上传到虚拟机然后构建并启动了。一共有5个springboot服务 5个Dockerfile 5个jar包 一个docker-compose.yml文件 一共11个文件 如果一个个执行scp上传的话太费劲，这里写个脚本，执行脚本就一次性上传完成，这就舒服了，但是我在脚本中使用的是sshpass工具上传，所以先在mac安装下 sshpass工具：

```
brew install hudochenkov/sshpass/sshpass
```

```



```
 brew link sshpass
```

安装好sshpass后 就是编写批量上传脚本了，内容如下：

```
#!/bin/bash

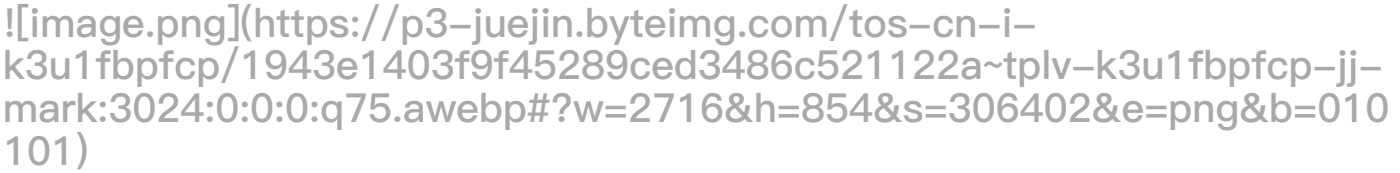
批量上传本地文件到 虚拟机 以便镜像构建以及 容器运行

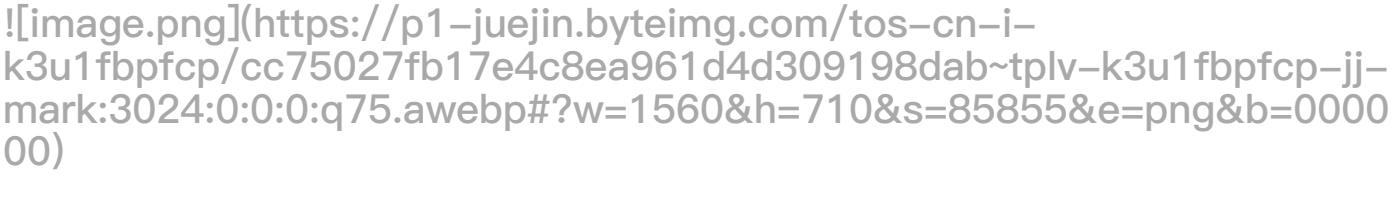
# 定义本地和远程目录前缀
local_base_dir="/Users/hzz/myself_project/im-开源04/xzll-im-parent"
remote_base_dir="/usr/local/soft_hzz/xzll-im/jar-file/docker-compose-way"

# 定义要上传的文件和目标目录
files=(
  "$local_base_dir/im-gateway/target/im-gateway.jar:$remote_base_dir/im-gateway/"
  "$local_base_dir/im-auth/target/im-auth.jar:$remote_base_dir/im-auth/"
  "$local_base_dir/im-business/im-business-service/target/im-business-service.jar:$remote_base_dir/im-business/"
  "$local_base_dir/im-connect/im-connect-service/target/im-connect-service.jar:$remote_base_dir/im-connect/"
  "$local_base_dir/im-console/im-console-service/target/im-console-service.jar:$remote_base_dir/im-console/"
  "$local_base_dir/im-gateway/src/main/resources/Dockerfile:$remote_base_dir/im-gateway"
  "$local_base_dir/im-
```
```

```
auth/src/main/resources/Dockerfile:$remote_base_dir/im-auth/"
  "$local_base_dir/im-business/im-business-
service/src/main/resources/Dockerfile:$remote_base_dir/im-business/"
  "$local_base_dir/im-connect/im-connect-
service/src/main/resources/Dockerfile:$remote_base_dir/im-connect/"
  "$local_base_dir/im-console/im-console-
service/src/main/resources/Dockerfile:$remote_base_dir/im-console/"
  "$local_base_dir/docker-compose.yml:$remote_base_dir/"
)

# 远程服务器的用户名和主机名
remote_user="root"
# 本机上的 环境变量 放到了： sudo vi ~/.zshrc 中
remote_host="$XUNIJI_ADDRESS"
remote_password="$REMOTE_PASSWORD"

# 日志文件
log_file="upload_log.txt"

# 遍历数组并执行上传
for file_pair in "${files[@]}"; do
  file="${file_pair%%:*}"
  remote_dir="${file_pair##*:}"

  echo "确保远程目录存在: $remote_dir" | tee -a "$log_file"
  sshpass -p "$remote_password" ssh ${remote_user}@${remote_host}
"mkdir -p $remote_dir"

  echo "上传文件 $file 到 $remote_dir" | tee -a "$log_file"
  if sshpass -p "$remote_password" scp -r "$file"
${remote_user}@${remote_host}:"$remote_dir"; then
    echo "上传成功: $file" | tee -a "$log_file"
  else
    echo "上传失败: $file" | tee -a "$log_file"
  fi
  echo "" | tee -a "$log_file"
done
```

注意：此脚本会判断 如果目标文件夹不存在则创建

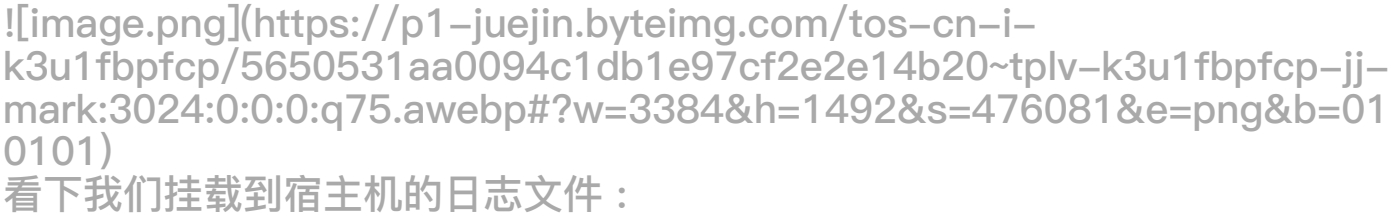
上传日志如下：


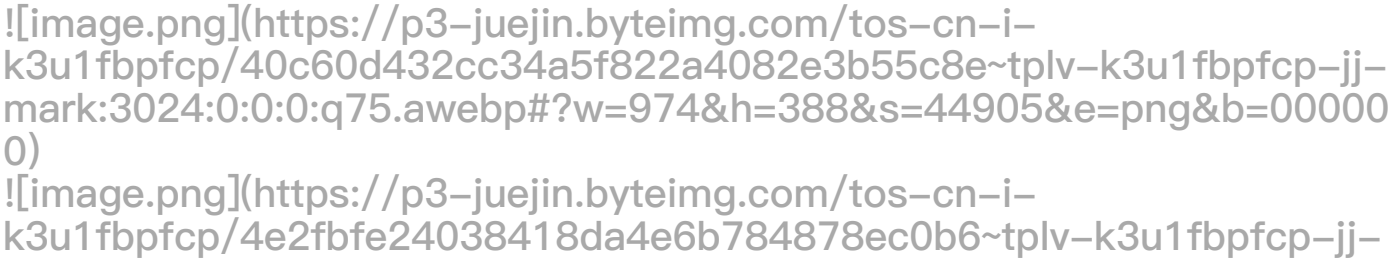
可以看到虚拟机中已有了：


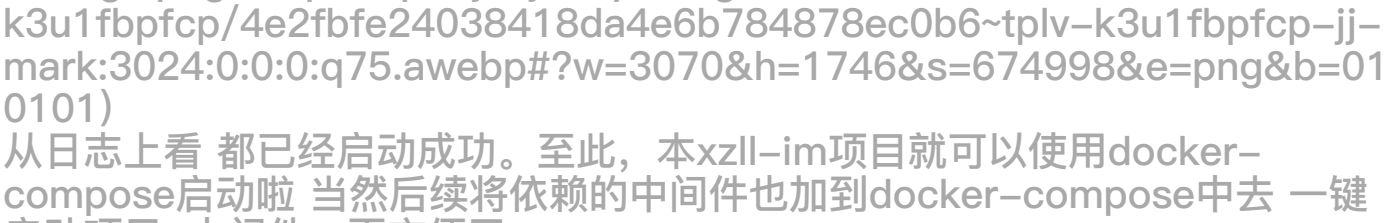
构建镜像并一键启动（包含服务和中间件）

之后我们直接 使用下边命令构建&启动 docker-compose 中定义的服务：

```
docker-compose up -d --build
```

执行效果如下：


看下我们挂载到宿主机的日志文件：




从日志上看 都已经启动成功。至此，本xzll-im项目就可以使用docker-compose启动啦 当然后续将依赖的中间件也加到docker-compose中去 一键启动项目+中间件。更方便了。

后续补充：

将依赖的中间件也写到docker-compose.yml中 这样的话 真正做到一键启动。

改后的docker-compose.yml文件：

```
version: '3.9'
services:
  im-gateway:
    build:
      context: ./im-gateway
      dockerfile: Dockerfile
    image: im-gateway:latest

    hostname: im-gateway
    container_name: im-gateway
    restart: always
    ports:
      - "8081:8081"
    networks:
      - default_network
    volumes:
      - "/tmp/data/logs:/logs"
    depends_on:
      - nacos
      - zookeeper
      - redis
      - rmq_broker
      - rmq_namesrv
  im-auth:
    build:
      context: ./im-auth
      dockerfile: Dockerfile
    image: im-auth:latest

    hostname: im-auth
    container_name: im-auth
    restart: always
    ports:
      - "8082:8082"
    networks:
      - default_network
    volumes:
      - "/tmp/data/logs:/logs"
    depends_on:
      - nacos
      - zookeeper
      - redis
      - rmq_broker
      - rmq_namesrv
  im-business:
    # 可以根据Dockerfile构建镜像（但是，Docker Compose 会在检测到上下文变化时重新构建镜像。也就是说如果你不修改Dockerfile docker-
```

```
compose应该不是每次都构建镜像 实测确实如此）
    build:
      context: ./im-business # 指定Dockerfile文件位置
      dockerfile: Dockerfile # 指定名称
    image: im-business:latest # 指定生成镜像的 名称

    # 也可以直接指定镜像名 但是要确保镜像存在 （如果在docker仓库, 则不需要再本地存在镜像 会自动pull）
    # image: im-business:0.0.2

    # 设置容器的主机名 即修改： /etc/hosts 中的内容
    hostname: im-business
    # 容器名称
    container_name: im-business
    # 重启策略， always 表示无论哪种状态退出，都会重启容器
    restart: always
    ports:
      # 设置主机与容器的端口映射
      - "8083:8083"
    networks:
      # 使用默认网络即：docker0 桥接
      - default_network
    volumes:
      # 将主机的 /tmp/data/logs 目录挂载到容器的 /logs 目录。这样可以实现数据的持久化，当容器重启时，数据不会丢失
      - "/tmp/data/logs:/logs"
    depends_on:
      - nacos
      - zookeeper
      - redis
      - rmq_broker
      - rmq_namesrv
  im-connect:
    build:
      context: ./im-connect
      dockerfile: Dockerfile
    image: im-connect:latest

    hostname: im-connect
    container_name: im-connect
    restart: always
    ports:
      - "10001:10001"
    networks:
      - default_network
    volumes:
      - "/tmp/data/logs:/logs"
    depends_on:
```

```
      - nacos
      - zookeeper
      - redis
      - rmq_broker
      - rmq_namesrv
  im-console:
    build:
      context: ./im-console
      dockerfile: Dockerfile
    image: im-console:latest

    hostname: im-console
    container_name: im-console
    restart: always
    ports:
      - "8084:8084"
    networks:
      - default_network
    volumes:
      - "/tmp/data/logs:/logs"
    depends_on:
      - nacos
      - zookeeper
      - redis
      - rmq_broker
      - rmq_namesrv

  # ########################################## 以下是此im项目依赖的中间件 ##########################################

  # rocketMq nameServer
  rmq_namesrv:
    image: apache/rocketmq:4.8.0
    container_name: rmq_namesrv
    restart: always
    ports:
      - "9876:9876"
    volumes:
      - /usr/local/soft_hzz/docker/rocketmq_namesrv/store:/root/store
      - /usr/local/soft_hzz/docker/rocketmq_namesrv/logs:/root/logs
    command: sh mqnamesrv
  # rocketMq broker
  rmq_broker:
    image: apache/rocketmq:4.8.0
    container_name: rmq_broker
    restart: always
    ports:
      - "10911:10911"
```

```
      - "10909:10909"
    volumes:
      - /usr/local/soft_hzz/docker/rocketmq_broker/store:/root/store
      - /usr/local/soft_hzz/docker/rocketmq_broker/logs:/root/logs
      -
/usr/local/soft_hzz/docker/rocketmq_broker/conf/broker.conf:/opt/rock
etmq-4.8.0/conf/broker.conf
    environment:
      - LOCAL_IP=${LOCAL_IP}
    command: sh mqbroker -c /opt/rocketmq-4.8.0/conf/broker.conf

  rocketmq-console:
    image: styletang/rocketmq-console-ng
    container_name: rocketmq-console
    restart: always
    ports:
      - "8080:8080"
    environment:
      JAVA_OPTS: "-Drocketmq.namesrv.addr=${LOCAL_IP}:9876 -
Dcom.rocketmq.sendMessageWithVIPChannel=false"
    depends_on:
      - rmq_namesrv
      - rmq_broker
  # nacos
  nacos:
    image: nacos/nacos-server:2.0.3
    container_name: nacos
    restart: always
    ports:
      - "8848:8848"
    volumes:
      - /usr/local/soft_hzz/docker/nacos/data:/home/nacos/data
      - /usr/local/soft_hzz/docker/nacos/logs:/home/nacos/logs
    environment:
      MODE: standalone
  #redis
  redis:
    image: redis
    container_name: redis
    restart: always
    ports:
      - "6379:6379"
    volumes:
      - /usr/local/soft_hzz/docker/redis/data:/data
      -
/usr/local/soft_hzz/docker/redis/conf/redis.conf:/usr/local/etc/redis/re
dis.conf
    command: redis-server /usr/local/etc/redis/redis.conf
```

```
  # zk
  zookeeper:
    image: zookeeper
    container_name: zookeeper
    restart: always
    ports:
      - "2181:2181"
    volumes:
      - /usr/local/soft_hzz/docker/zk/data:/data
      - /usr/local/soft_hzz/docker/zk/datalog:/datalog
      - /usr/local/soft_hzz/docker/zk/conf/zoo.cfg:/conf/zoo.cfg

networks:
  default_network:
    # 桥接
    driver: bridge
```

配置文件：
首先需要设置环境变量，方便我们在docker-compose.yml或者配置文件中引用：

```
# 编辑bash文件
sudo vim ~/.bashrc

# 设置环境变量
export LOCAL_IP=$(ip addr show enp0s3 | grep 'inet ' | awk '{print $2}' | cut -d/ -f1)

# 使其生效
source ~/.bashrc

# 打印看看  将输出enp0s3网卡对应的ip
echo $LOCAL_IP
```

```
# zoo.cfg配置文件内容：

tickTime=2000
dataDir=/data
dataLogDir=/datalog
```

```
clientPort=2181
initLimit=5
syncLimit=2
server.1=zookeeper:2888:3888

# redis.conf文件内容：

appendonly yes
requirepass 123456

broker.conf文件  内容：

brokerClusterName = DefaultCluster
brokerName = broker-a
brokerId = 0
deleteWhen = 04
fileReservedTime = 48
brokerRole = ASYNC_MASTER
flushDiskType = ASYNC_FLUSH
namesrvAddr=${LOCAL_IP}:9876
autoCreateTopicEnable=true
brokerIP1=${LOCAL_IP}

```

由于我将中间件和服务全部搞到docker-compose中 所以以后不需要docker run方式启动了 这里把之前启动的容器和下载的镜像 全部清理掉。
执行

```
# 将停止并删除所有容器
docker rm -f $(sudo docker ps -a -q)
# 删除所有镜像
docker rmi $(docker images -q)

```

之后再执行命令 构建 或者 pull镜像并启动容器：

* (自己写的服务是构建 即根据docker-compose.yml中的buil指令 来使用Dockerfile文件build)
* (中间件是直接指定了镜像名称 所以直接去docker仓库 pull)

```
docker-compose up -d --build
```

可以看到 项目服务和相关中间件 一个命令全部启动成功了，**挺爽的：**


有个点要注意，就是日志输出框架的配置，需要指定目录到容器的 /logs， 这样才能和docker-compose中的日志挂载配置相吻合 ，我使用的是log4j2 ，我是这么定义的(以im-console为例)：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xml>
<Configuration status="off" monitorInterval="10">

  <properties>
    <property name="FILE_NAME">im-console</property>
    <property name="LOG_HOME">/logs</property> <!-- 设置日志输出目录为 /logs -->
  </properties>

  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%date{yyyy-MM-dd HH:mm:ss.SSS} %level [%thread][%file:%M:%line] [%X{im_trace_id}] - %msg%n" />
      <ThresholdFilter level="info" onMatch="ACCEPT" onMismatch="DENY" />
    </Console>
    <RollingRandomAccessFile name="File"
                 fileName="${LOG_HOME}/${FILE_NAME}.log"
                 filePattern="${LOG_HOME}/$${date:yyyy-MM}/${FILE_NAME}-%d{yyyy-MM-dd}-%i.log">
      <PatternLayout pattern="%date{yyyy-MM-dd HH:mm:ss.SSS} %level [%thread][%file:%M:%line] [%X{trace_id}] - %msg%n" />
      <Policies>
        <TimeBasedTriggeringPolicy />
        <SizeBasedTriggeringPolicy size="2000MB" />
      </Policies>

      <DefaultRolloverStrategy max="40" />
```

```
        <ThresholdFilter level="info" onMatch="ACCEPT"
onMismatch="DENY" />
      </RollingRandomAccessFile>
    </Appenders>

    <Loggers>
      <Root level="info">
        <AppenderRef ref="Console" />
        <AppenderRef ref="File" />
      </Root>
      <logger level="info" name="jdbc.sqlonly" additivity="false">
        <AppenderRef ref="Console" />
        <AppenderRef ref="File" />
      </logger>

      <logger level="OFF" name="jdbc.sqltiming" additivity="false">
        <AppenderRef ref="Console" />
        <AppenderRef ref="File" />
      </logger>

      <logger level="OFF" name="jdbc.companyAudit" additivity="false">
        <AppenderRef ref="Console" />
        <AppenderRef ref="File" />
      </logger>

      <logger level="OFF" name="jdbc.resultset" additivity="false">
        <AppenderRef ref="Console" />
        <AppenderRef ref="File" />
      </logger>

      <logger level="OFF" name="jdbc.connection" additivity="false">
        <AppenderRef ref="Console" />
        <AppenderRef ref="File" />
      </logger>

      <logger level="OFF" name="jdbc.audit" additivity="false">
        <AppenderRef ref="Console" />
        <AppenderRef ref="File" />
      </logger>

      <logger level="OFF" name="serviceStatsLog" additivity="false">
        <AppenderRef ref="Console" />
        <AppenderRef ref="File" />
      </logger>
    </Loggers>
</Configuration>

```