1. 部署

1.1 制作docker镜像

笔者选用docker作为部署容器，同时为了提高系统的安全性和稳定性，笔者通过阿里云构建了一个私有镜像。首先要在去阿里云上的容器镜像服务中创建一个新的命名空间和仓库，然后通过阿里云的指导文档即可很轻易的进行上传或拉取镜像。制作私有镜像的过程大致如下图所示：

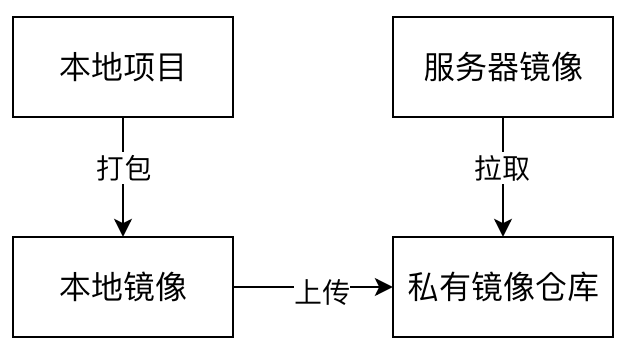


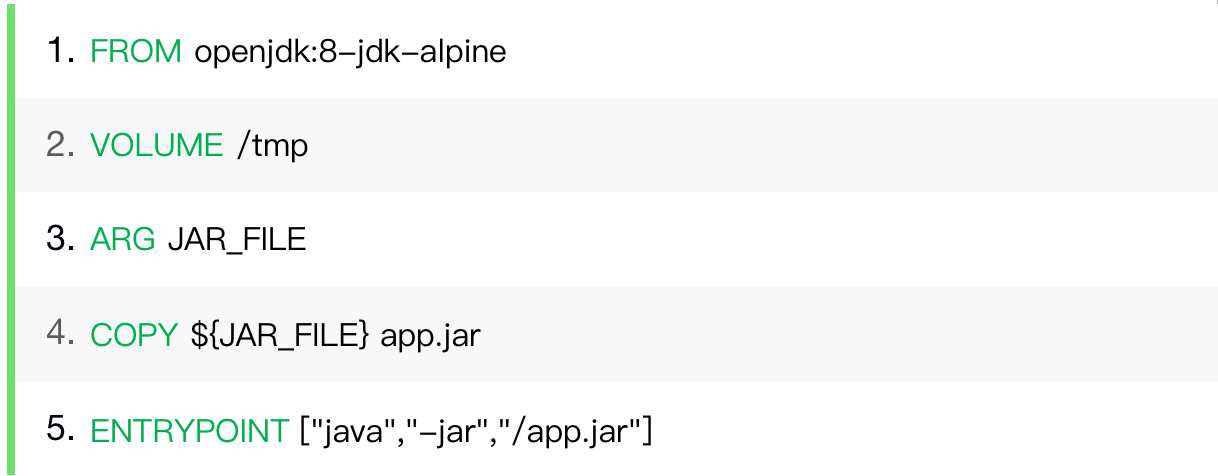
图1.1 私有镜像使用流程

Spring boot提供了特别好用的docker镜像制作插件，对此我们要引入以下依赖：



repository标签代表镜像名称，JAR\_FILE标签代表要打包进镜像的文件所在的位置，它的作用在下面会讲到。

maven 插件配置完成之后，就可以开始制作镜像了，对此笔者为了简化操作，制作了一个Dockerfile，如下面的代码所示：



由于运行jar包需要有jdk环境，所以笔者首先引入了openjdk的镜像源，这是第一行的作用；VOLUME的作用是配置一个具有持久化功能的目录，主机 /var/lib/docker 目录下创建了一个临时文件，并链接到容器的/tmp。此步骤可选，如果涉及到文件系统的应用就很有必要了。/tmp目录用来持久化到 Docker 数据文件夹，因为 Spring Boot 使用的内嵌 Tomcat 容器默认使用/tmp作为工作目录；ARG的作用是设置编译镜像时加入的参数，这里面的JAR\_FILE参数就是笔者在上述插件中的JAR\_FILE标签内的内容，且直接作用到COPY，它的作用就是把刚刚打包的jar包复制到镜像内；最后一句代码则是指定容器启动时运行的代码（而不是镜像制作的时候），它表示运行一个jar包。

接下来就可以开始进行打包构建了，把上述Dockerfile放入项目根目录然后运行以下代码即可：

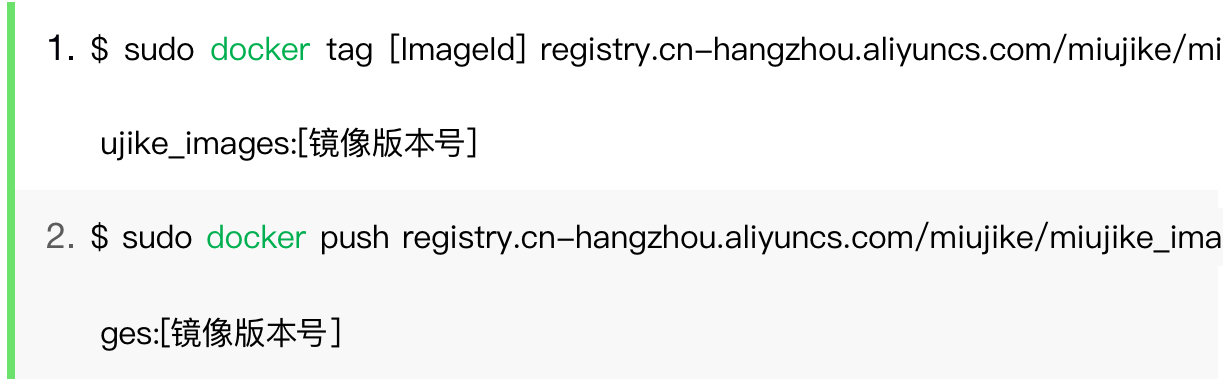


1.2 镜像发布

镜像制作完成之后，在本地机器docker查看镜像，即可发现新增的镜像：



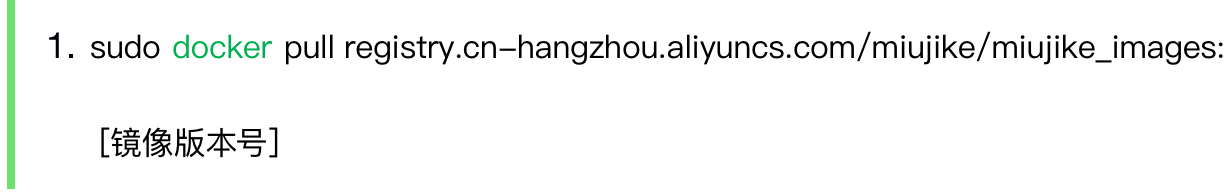
要发布镜像，我们要先在shell中登录到阿里云的Docker Registry，确认信息之后才能进行push操作：



其中docker tag的命令是用来重命名镜像的，可以进行省略；docker push则是把镜像推送到阿里云。经过以上操作，笔者的镜像就已经发布完成了。

1.3 拉取镜像

在镜像拉取之前，在目标服务器上同样要先登录到Docker Registry，然后通过docker pull则可以拉取镜像到目标服务器：



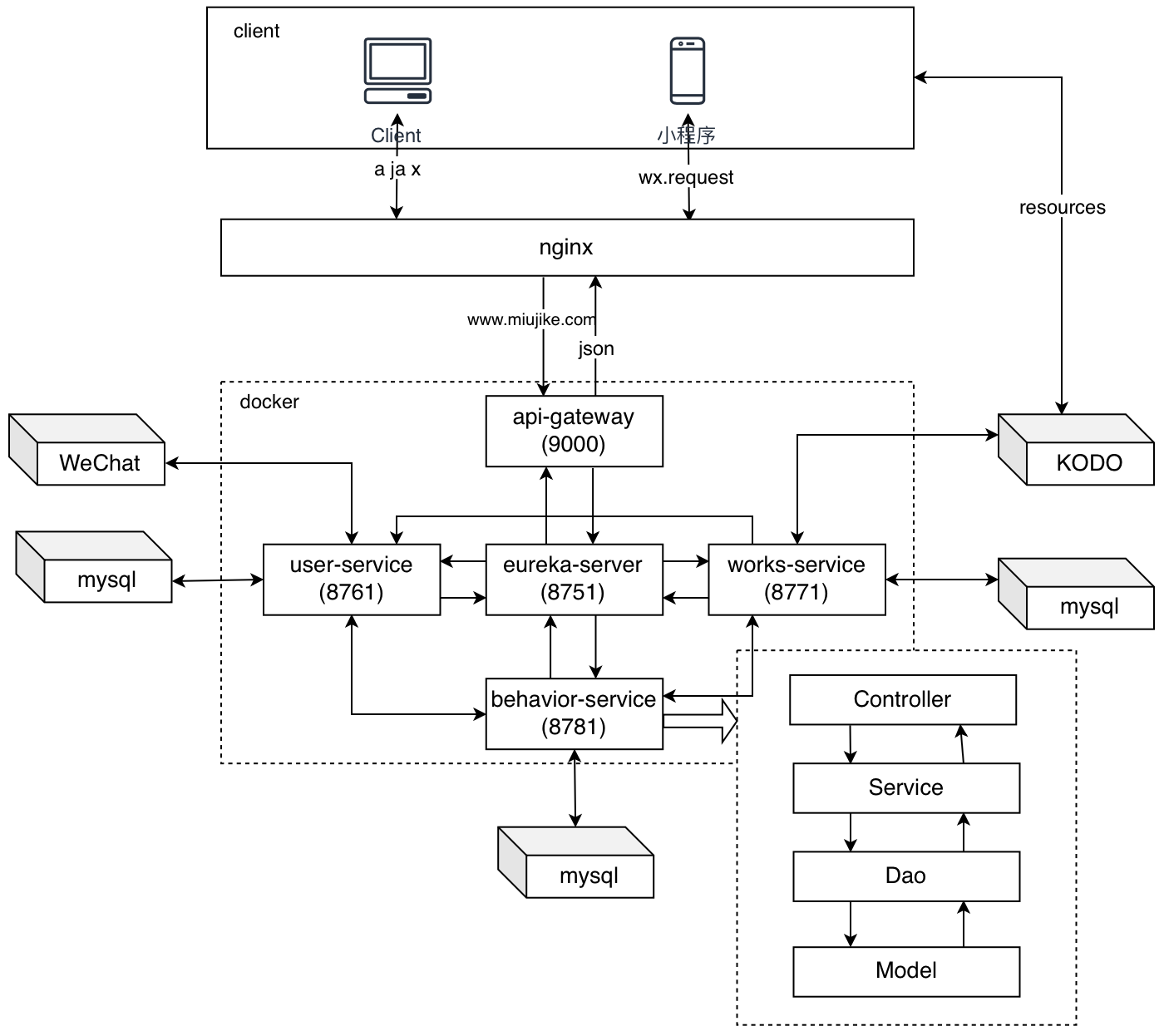
然后启动一个容器：



其中，--name代表对容器进行命名，-p代表将容器中的端口映射到本机的真正端口，最后面的一串字符代表镜像的id。

1. 系统架构设计

本系统采用小程序加微服务相结合的方式，服务端采用spring cloud的方式进行构建，以微服务的方式进行项目治理，如下图所示：



笔者选用了eureka作为注册中心，微服务体系中的其他服务都会向注册中心进行注册。除了网关服务以外，其他几个微服务各自包含四层业务结构：

1. Controller层：负责对外提供接口调用，在本系统中主要是交个网关进行映射以及作为模块之间的相互调用接口；
2. Service层：负责具体的业务处理，然后交个Controller层进行调用；
3. Dao层负责处理一些单元性的增删改查等操作，由于笔者选用的是mybatis，这里的Dao层会与XML 文件相映射，具体的数据库处理交给SQL语句，因此本层都是一些对Service层提供的接口；
4. Model层：存放数据库实体，与数据库相映射。

各服务之间的相互调用，笔者选用了fegin进行http调用，并存放在Service层中，这里可以当成本地bean一样进行使用。网关层采用zuul组件，负责对外统一暴露接口地址，所有用户对系统的调用都是通过网关接口进行了，而直接访问每个服务的调用都会被网关拦截掉，这样可以极大提高系统的安全性。相关的配置方法，笔者会在下面进行讨论。

笔者的数据库选用了mysql，通过jdbc进行连接；音视频文件的存储放在了KODO对象存储服务器上，通过CDN进行加速。另外，在user-service中还会对接到微信服务器进行用户授权登录验证。代理服务器笔者选用nginx，数据传输都是通过json数据格式进行传输。请求方面，小程序客户端采用的是wx.request({})方法，web端采用ajax进行，两者都是异步请求的方式，可以进行并发访问。