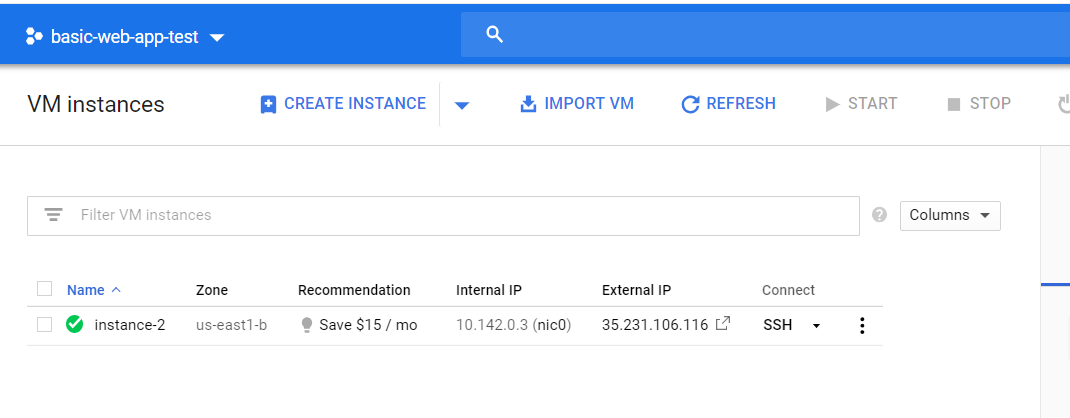
用Python/Keras/Flask/Docker在Kubernetes上部署深度学习模型

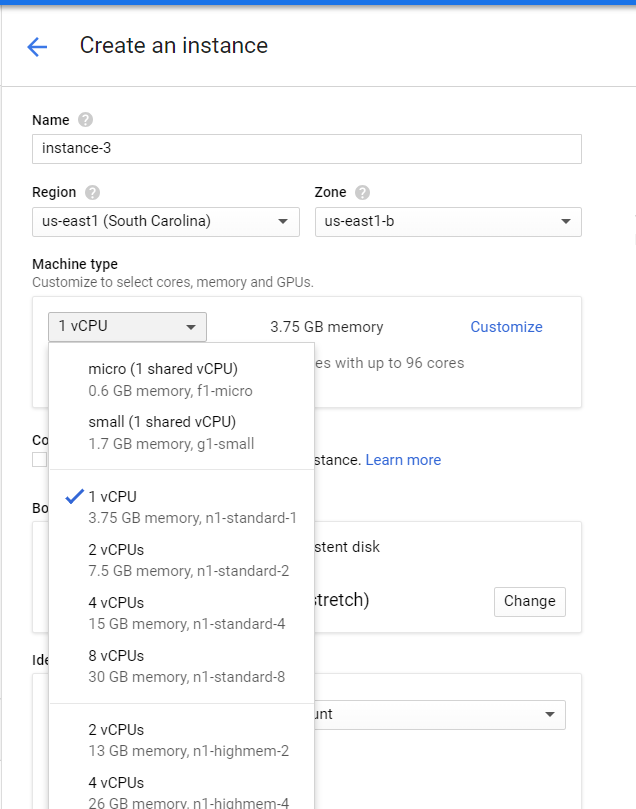
通过Docker和Kubernetes，用Keras部署深度学习模型，并且通过Flask提供REST API服务。

## 云平台创建用户

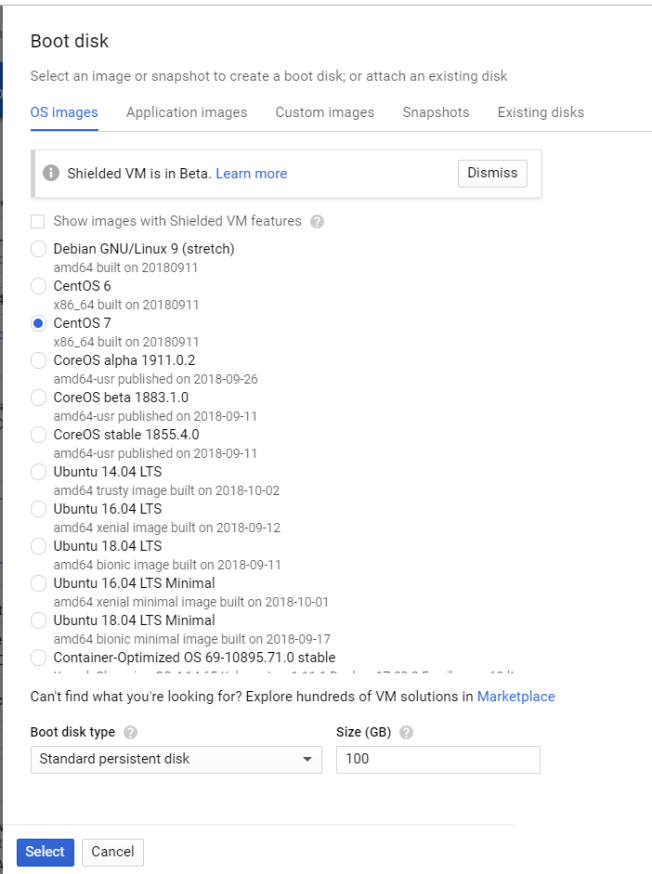
以Google云平台为例，创建了一个对外提供服务的容器化深度学习模型，当然Google平台并不是必须的，只要能够安装Docker，随便选择平台模式。

1. 进入Google云平台，点击左侧屏幕选择Compute Engine，启动Google Cloud VM。然后选择“Create Instance”，可以看到已经运行的实例。

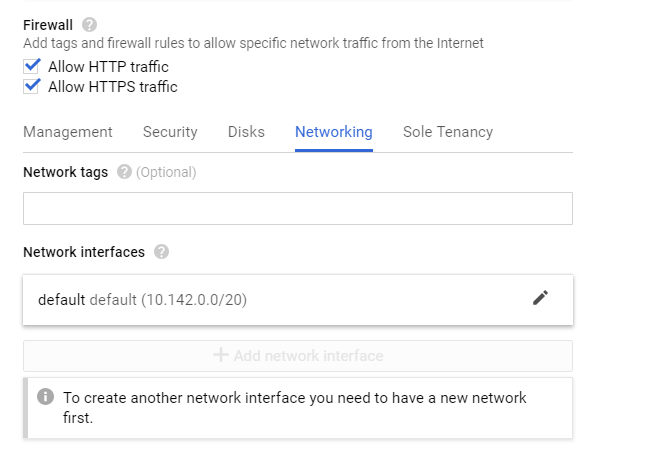
[](http://dockone.io/uploads/article/20181029/2b672c97b3862620a5409e97d676da1f.png)（2） 下一步选择计算资源。一般默认设置就足够，选择4vCPUs和15G内存。

[](http://dockone.io/uploads/article/20181029/333a14d64e15e31c34dfa5cb38763b81.png)

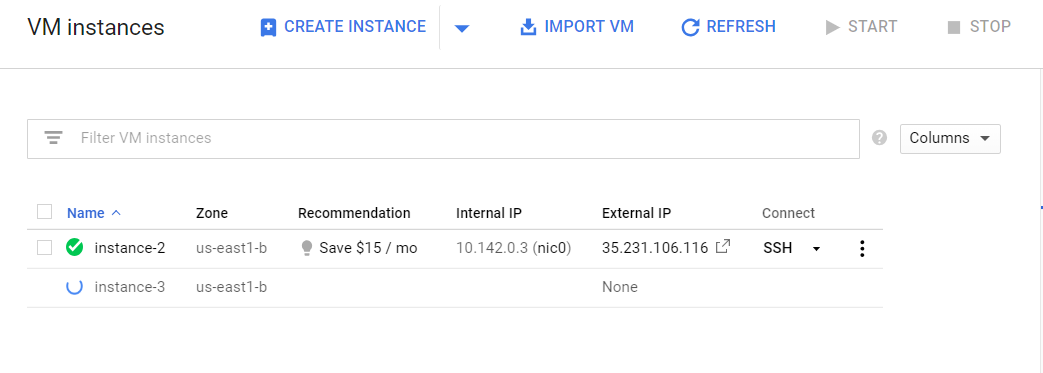
（3）选择操作系统和磁盘大小。选择CentOS 7，100G硬盘。建议磁盘大于10G，因为每个Docker容器有1G大小。

[](http://dockone.io/uploads/article/20181029/e86ade654d2ce88724f10f68caeeac09.png)

（4） 最后一步是配置允许HTTP/S工作的防火墙策略。建议选择全部透明，以便减少麻烦。

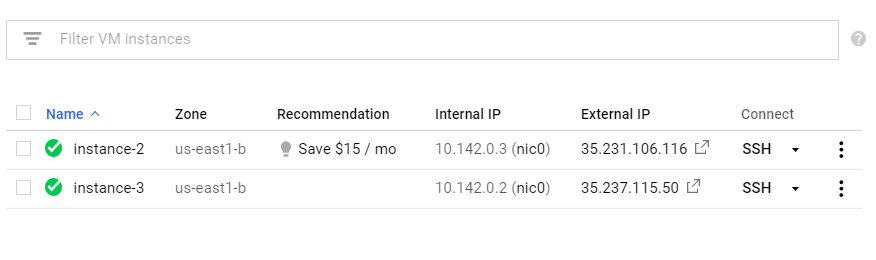
[](http://dockone.io/uploads/article/20181029/e8f370735490e120e32d11dfaea4787a.png)

（5） 选择“Create”

[](http://dockone.io/uploads/article/20181029/92fb03ec05e6a536905501f6bbe56c6e.png)

## 创建深度学习模型

SSH登录到虚机开始建立模型。最简单方式就是点击虚机下方的SSH图标，会在浏览器中打开一个终端。

[](http://dockone.io/uploads/article/20181029/9738fd2edf25b24f0c75b444ae0ef6f5.png)

1. 删除预装Docker

sudo yum remove docker docker-client docker-client-latest docker-common docker-latest docker-latest-logrotate docker-logrotate docker-selinux docker-engine-selinux docker-engine

1. 安装最新Docker版本

sudo yum install -y yum-utils device-mapper-persistent-data lvm2

sudo yum-config-manager — add-repo https://download.docker.com/linux/centos/docker-ce.repo

sudo yum install docker-ce

1. 启动容器运行测试脚本

sudo systemctl start docker

sudo docker run hello-world

以下是正确输出：

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

1. 创建深度学习模型  
   这里会借用Adrian Rosebrock的一个脚本，提供了使用Keras的深度学习模型并通过Flask提供服务的教程，可以从[这里](https://blog.keras.io/building-a-simple-keras-deep-learning-rest-api.html)访问。  
   这个模型可以直接执行。但是我修改了两个配置信息：  
   首先，改变了容器配置，默认flask使用127.0.0....作为默认服务地址，这会在容器内部运行时出现问题。我将它修改成0.0.0.0，这样就可以实现对外和对内都可以工作的IP地址。  
   第二是关于Tensorflow的配置，可以从GitHub中找到这个[问题描述](https://github.com/tensorflow/tensorflow/issues/14356)。

global graph

graph = tf.get\_default\_graph()

...

with graph.as\_default():

preds = model.predict(image)

运行脚本，首先创建专用目录：  
mkdir keras-app

cd keras-app

创建app.py文件：  
vim app.py

1. 创建requirements.txt文件  
   为了在容器内运行代码，需要创建requirements.txt文件，其中包括需要运行的包，例如keras、flask、一起其它相关包。这样无论在哪里运行代码，依赖包都保持一致。

keras

tensorflow

flask

gevent

pillow

requests

1. 创建Dockerfile

FROM python:3.6

WORKDIR /app

COPY requirements.txt /app

RUN pip install -r ./requirements.txt

COPY app.py /app

CMD ["python", "app.py"]~

首先让容器自行下载Python 3安装image，然后让Python调用pip安装requirements.txt中的依赖包，最后运行python app.py。

1. 创建容器

sudo docker build -t keras-app:latest .

在keras-app目录下创建容器，后台开始安装Python 3 image等在步骤6中定义的操作。

1. 运行容器  
   sudo docker run -d -p 5000:5000 keras-app

用sudo docker ps -a检查容器状态，应该看到如下输出：  
[gustafcavanaugh@instance-3 ~]$ sudo docker ps -a

CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS                    PORTS                    NAMES

d82f65802166        keras-app           "python app.py"     About an hour ago   Up About an hour          0.0.0.0:5000->5000/tcp   nervous\_northcutt

1. 测试模型  
   现在可以测试此模型。用狗的照片作为输入，可以返回狗的品种。

执行命令：

curl -X POST -F image=@dog.jpg 'http://localhost:5000/predict'

应该得到如下输出：

{"predictions":[{"label":"beagle","probability":0.987775444984436},{"label":"pot","probability":0.0020967808086425066},{"label":"Cardigan","probability":0.001351703773252666},{"label":"Walker\_hound","probability":0.0012711131712421775},{"label":"Brittany\_spaniel","probability":0.0010085132671520114}],"success":true}

## Kubernetes部署模型

1. 创建Docker Hub账号  
   第一步需要在[Docker hub](https://hub.docker.com/)上传模型，以便使用Kubernetes集中管理。
2. 登录到Docker Hub  
   sudo docker login， 登录到Docker Hub，应该看到如下输出：  
   Login Succeeded
3. 给容器打标签  
   给模型容器命名，上传前先给它打标签。  
   sudo docker images，应该得到容器的id，输出如下：

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE keras-app           latest              ddb507b8a017        About an hour ago   1.61GB

打标签命令如下：

#Format

sudo docker tag <your image id> <your docker hub id>/<app name>

My Exact Command - Make Sure To Use Your Inputs

sudo docker tag ddb507b8a017 gcav66/keras-app

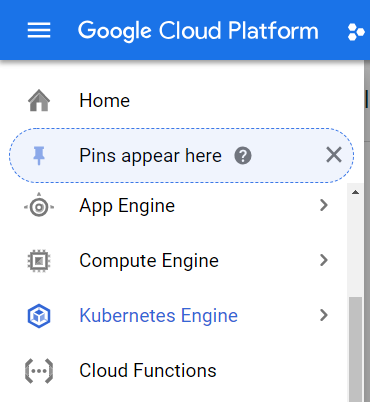
（4） 将模型容器上传到Docker Hub  
运行命令如下：  
 #Format

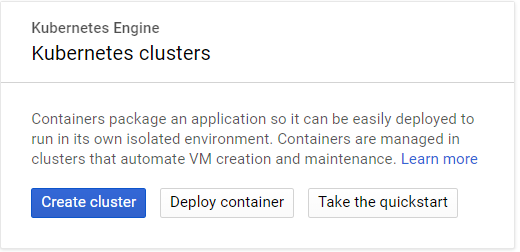
sudo docker push <your docker hub name>/<app-name>

My exact command

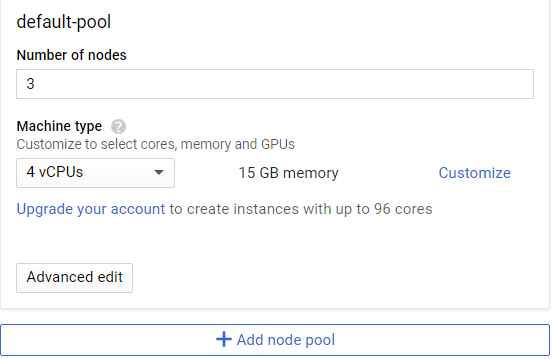
sudo docker push gcav66/keras-app

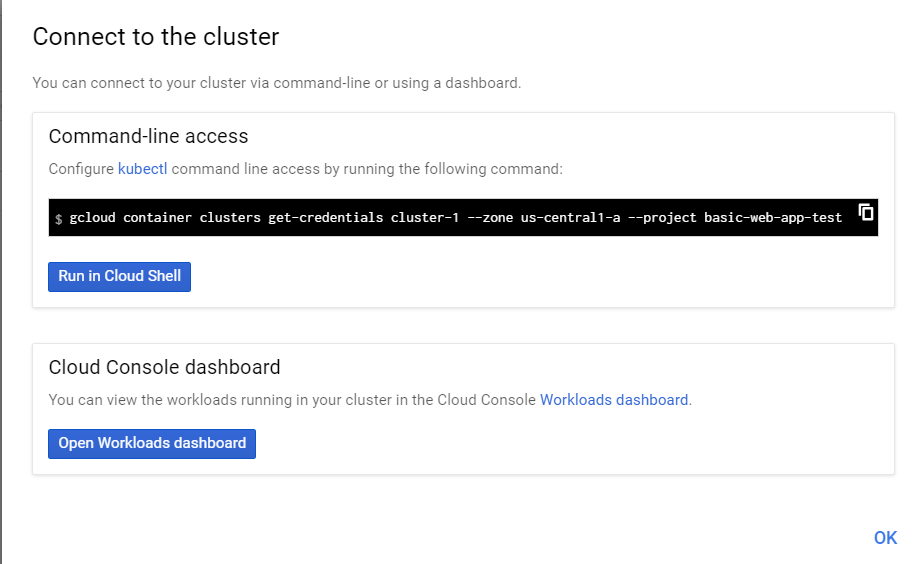
（5） 创建Kubernetes集群  
在Google Cloud Home界面，选择Kubernetes Engine。

[](http://dockone.io/uploads/article/20181029/d9092f358ebfe5116cdfd4090164605b.png)  
创建新集群：

[](http://dockone.io/uploads/article/20181029/b73b4a57040eea9256ae47368c7ad1f0.png)

选择集群内节点资源，因为要启动三个节点（每个节点4vCPU和15G内存），至少需要12vCPU和45G内存。

[](http://dockone.io/uploads/article/20181029/a9a9c23c05ec3381711f8377e51d7cf8.png)连接集群，Google’s Kubernetes自动会在VM上安装Kubernetes。

[](http://dockone.io/uploads/article/20181029/2f4b787188dfe297bdd9263d51399573.png)

在Kubernetes中运行容器：  
kubectl run keras-app --image=gcav66/keras-app --port 5000

确认是否Pod正确运行kubectl get pods，输出如下：

gustafcavanaugh@cloudshell:~ (basic-web-app-test)$ kubectl get pods

NAME                         READY     STATUS    RESTARTS   AGE

keras-app-79568b5f57-5qxqk   1/1       Running   0          1m

为了安全起见，将服务端口暴露与80端口：

kubectl expose deployment keras-app --type=LoadBalancer --port 80 --target-port 5000

确认服务正常启动：kubectl get service，正常输出如下：

gustafcavanaugh@cloudshell:~ (basic-web-app-test)$ kubectl get service

NAME         TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE

keras-app    LoadBalancer   10.11.250.71   35.225.226.94   80:30271/TCP   4m

kubernetes   ClusterIP      10.11.240.1    <none>          443/TCP        18m

提取cluster-IP，并将其合并于服务提交命令：curl -X POST -F image=@dog.jpg 'http://&lt;your service IP>/predict'，得到正常输入如下：

$ curl -X POST -F image=@dog.jpg 'http://35.225.226.94/predict'

{"predictions":[{"label":"beagle","probability":0.987775444984436},{"label":"pot","probability":0.0020967808086425066},{"label":"Cardigan","probability":0.001351703773252666},{"label":"Walker\_hound","probability":0.0012711131712421775},{"label":"Brittany\_spaniel","probability":0.0010085132671520114}],"success":true}

参考链接：<http://dockone.io/article/8316>

利用docker部署深度学习模型的一个最佳实践

深度学习模型部署的痛点：

* 部分神经网络框架比如caffe依赖过重，安装困难。
* 各种网络模型未做工程化优化，部署困难。
* tensorflow等框架对GPU等硬件的占用难以灵活控制。

对于做应用来说，这些问题诸如对GPU的硬件的管理，对复杂依赖的部署，而这些正好就是docker的强项。而python本身表达能力强，可以以很短的代码量达成我们的目的。

## 部署流程

具体的部署步骤涉及这几个工具链：

1. [Dockerfile](https://link.zhihu.com/?target=https%3A//docs.docker.com/develop/develop-images/dockerfile_best-practices/)进行模型的镜像部署。
2. [docker-py](https://link.zhihu.com/?target=https%3A//docker-py.readthedocs.io/en/latest/containers.html)进行container的启动和关闭。
3. [grpc](https://link.zhihu.com/?target=https%3A//grpc.io/docs/quickstart/python.html)和进行模型的外部通信。
4. python的[with语句](https://link.zhihu.com/?target=http%3A//python.jobbole.com/87317/)表达模型的加载和资源的释放。
5. [gitlab](https://link.zhihu.com/?target=https%3A//about.gitlab.com/getting-help/)进行内网的代码分发和版本控制。

整个接口的调用精简成面向对象的调用方式，with语句进入时启动模型，占用GPU，打开rpc调用端口，之后在调用结束后退出模型，释放资源，整个调用过程就简化成如下样子：

**with** Model\_Docker() **as** sess:

img **=** cv2**.**imread('demo.jpg')

r **=** sess**.**run(img)

print('result:',r,'type',type(r))

其中Model\_Docker是这样的：

**class** **CTPN\_Docker**(object):

**def** **\_\_init\_\_**(self):

self**.**client **=** docker**.**from\_env()

**def** **get\_container**(self,client):

container **=** client**.**containers**.**run(image **=** DEMO\_IMAGE\_NAME:TAG,

command **=** "python server.py",

runtime**=**'nvidia',

environment **=** ["CUDA\_VISIBLE\_DEVICES=0"],

ports **=** {'8888/tcp':'8888'},

detach**=True**,

auto\_remove **=** **True**)

**return** container

**def** **\_\_enter\_\_**(self):

self**.**container **=** self**.**get\_container(self**.**client)

**for** line **in** self**.**container**.**logs(stream**=True**):

**if** line**.**strip()**.**find(b'grpc\_server\_start') **>=** 0:

**break**

**return** self

**def** **\_\_exit\_\_**(self, exc\_type, exc\_val, exc\_tb):

self**.**container**.**stop()

print('container has stopped.')

**def** **run**(self,img):

**assert** isinstance(img,np**.**ndarray), 'img must be a numpy array.'

imgstr **=** img**.**tobytes()

shape **=** json**.**dumps(img**.**shape)

stub **=** ctpn\_pb2\_grpc**.**ModelStub(grpc**.**insecure\_channel('localhost:50051'))

response **=** stub**.**predict(ctpn\_pb2**.**rect\_request(img**=**imgstr, shape**=**shape))

**return** json**.**loads(response**.**message)

整个流程是这么个步骤：

1. \_*init*\_方法获得docker client。
2. get\_container方法实例化一个container。
3. with语句进入接口的\_*enter*\_方法，负责获取container实例和实例内模型启动结 束的flag。
4. with语句清理接口的\_*exit*\_方法，负责实例的关闭。
5. run方法通过grpc调用docker内模型和返回结果。

docker-py是一个docker的python接口，docker除了cmdline的操作方式，还提供了REST的调用接口，docker-py就是其中一个很人性化的封装，[具体使用可见官方文档](https://link.zhihu.com/?target=https%3A//docker-py.readthedocs.io/en/latest/containers.html)。

container的实例化中有这几个地方需要注意：

1. runtime需要用nvidia，与使用nvidia-docker效果一样。
2. detach是后台模式，与-d效果一样。
3. auto\_remove是自动删除，与--rm效果类似。
4. environment 来设定CUDA\_VISIBLE\_DEVICES。
5. ports 来指定导出端口映射。

除了docker-py调用中的这些技巧，还有如下几个指令在构建过程中值得注意。

1、grpc的编译，这里没啥好说的，和grpc的官方说明文档里一样。

RUN python -m grpc\_tools.protoc --python\_out=. --grpc\_python\_out=. -I. mode.proto

2、docker image的构建，有时候构建需要添加--no-cache，避免远程资源更新了，docker构建却没重新。

docker build --no-cache -t name .

3、pip安装的时候需要添加几个参数，-r指定.txt安装，-i指定清华镜像为安装源，--no-cache-dir压缩docker镜像。

RUN pip install -r requirements\_docker.txt -i https://pypi.tuna.tsinghua.edu.cn/simple --no-cache-dir

参考链接：

<https://zhuanlan.zhihu.com/p/42809515>**利用docker部署深度学习模型的一个最佳实践**

<https://blog.csdn.net/xingoo_/article/details/86164566> **深度学习Tensorflow生产环境部署（下·模型部署篇）**