基于MindX SDK的语音多分类实验指导手册

版本：1.0



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2021。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  C:\Users\jwx341670\Desktop\华为标志 Huawei Logo 2018\竖版标志Vertical Version\PNG\HW_POS_RBG_Vertical-150ppi.png 和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  注意  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | http://[e](http://e.huawei.com/).huawei.com |

目录

[1 实验总览 2](#_Toc119767116)

[1.1 实验背景 2](#_Toc119767117)

[1.2 实验目的 2](#_Toc119767118)

[1.3 实验清单 2](#_Toc119767119)

[2 基于MindX SDK的语音多分类实验 3](#_Toc119767120)

[2.1 实验简介 3](#_Toc119767121)

[2.2 实验环境 3](#_Toc119767122)

[2.3 实验内容 3](#_Toc119767123)

[2.3.1 数据集介绍 3](#_Toc119767124)

[2.3.2 模型介绍 4](#_Toc119767125)

[2.3.3 实验流程 4](#_Toc119767126)

[2.4 实验目的 5](#_Toc119767127)

[2.5 实验过程 5](#_Toc119767128)

[2.6 5](#_Toc119767129)

[2.7 5](#_Toc119767130)

[2.8 5](#_Toc119767131)

[2.6 部分代码介绍 7](#_Toc119767132)

[2.6.1 main脚本构建 7](#_Toc119767133)

[2.6.2 配置文件 10](#_Toc119767134)

[2.6.3 Pipeline任务流配置介绍 10](#_Toc119767135)

[2.6.4 音频数据预处理 11](#_Toc119767136)

[2.6.5 模型推理结果后处理 13](#_Toc119767137)

[2.6.6 运行脚本 14](#_Toc119767138)

[2.7 实验总结 14](#_Toc119767139)

[3 附录：环境准备 15](#_Toc119767140)

[3.1 获取镜像 15](#_Toc119767141)

[3.2 创建ECS 弹性服务器 16](#_Toc119767142)

[3.2.1 创建ECS弹性云服务器 16](#_Toc119767143)

[3.2.2 MobaXterm连接ECS 21](#_Toc119767144)

[3.3 关闭ECS弹性云服务器 23](#_Toc119767145)

# 实验总览

## 实验背景

语音分类任务（以本实验中的音乐分类任务为例），是指从音频文件的频谱信息中提取数据特征，然后通过统计学或者神经网络算法，对该音频可能属于的音乐流派进行判断的任务。

## 实验目的

本章实验的主要目的是掌握语音分类相关基础知识点，使用MindX SDK工具包实现音乐片段所属流派进行分类，加深对相关理论的理解。

## 实验清单

表格：实验、简述、难度、软件环境、硬件环境。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验 | 简述 | 难度 | 软件环境 | 开发环境 |
| 基于MindX SDK的语音多分类实验 | 通过MindX SDK来完成语音多分类任务 | 中级 | Python3.7.5、CANN3.3.0  FFmpeg3.4.8 | ECS |

# 基于MindX SDK的语音多分类实验

## 实验简介

语音分类任务（以本实验中的音乐分类任务为例），是指从音频文件的频谱信息中提取数据特征，然后通过统计学或者神经网络算法，对该音频可能属于的音乐流派进行判断的任务。

本实验通过配置镜像环境、购买ECS服务、部署项目代码、修改配置文件及执行推理脚本全流程展示如何使用MindX SDK完成音乐音频内容的分类任务。

## 实验环境

对象存储服务（OBS）、弹性云服务器（ECS）、MobaXterm。

对象存储服务（Object Storage Service，OBS）提供海量、安全、高可靠、低成本的数据存储能力，可供用户存储任意类型和大小的数据。适合企业备份/归档、视频点播、视频监控等多种数据存储场景。

弹性云服务器（Elastic Cloud Server,ECS） 是由CPU、内存、操作系统、云硬盘组成的基础的计算组件。弹性云服务器创建成功后，您可以像使用自己的本地PC或服务器一样，在云上使用弹性云服务器。

MobaXterm (官方网站：<https://mobaxterm.mobatek.net/>，可以下载“Home Edition”版本)是远程SSH连接工具。

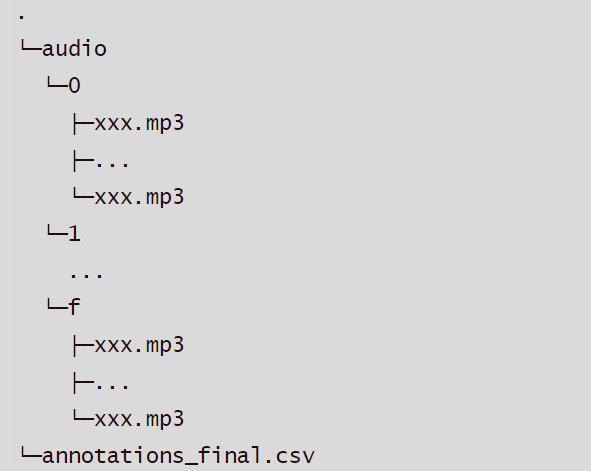
MindX SDK提供昇腾AI处理器加速的各类AI软件开发套件(SDK)，提供极简易用的API，加速高性能AI应用的开发，赋能千行百业。详细资料可参考：<https://www.hiascend.com/document/detail/zh/mind-sdk/30rc3/overview/index.html>

## 实验内容

### 数据集介绍

本实验使所用的数据为“MagnaTagATune Dataset” (数据集介绍：<https://mirg.city.ac.uk/codeapps/the-magnatagatune-dataset>) 。包含25,863个音乐片段。每个剪辑都是29秒长的摘录，属于5223首歌曲、445张专辑和230位艺术家。剪辑涵盖了广泛的流派，如古典、新时代、电子、摇滚、流行、世界、爵士、蓝调、金属、朋克等。每个音频剪辑都提供了一个包含188个标签的二进制注释向量。由于数据标签过多，且数据较不平衡，因此fcn4模型训练过程中，只采用top50标签数据集。在本实验中通过配置文件指定，输出的结果为测试数据最可能属于的5个音乐类别。

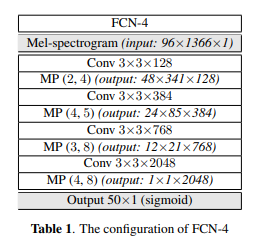
数据格式如下所示，音频文件audio下存放多个mp3文件。annotations\_final.csv存放数据的类别标注，1表示属于该类。



数据集路径结构

### 模型介绍

本实验用到的网络结构为FCN-4（详细结构参考原论文：<https://arxiv.org/pdf/1606.00298.pdf>；原始的ckpt模型下载链接为：[点此下载ckpt模型](https://download.mindspore.cn/model_zoo/r1.1/fcn4_ascend_v111_musictag_research_audio_bs32_acc90/fcn4_ascend_v111_musictag_research_audio_bs32_acc90.ckpt)）。由于cnn网络结构需要NCHW/NHWC矩阵结构，无法将音频数据直接传入网络结构中，因此在论文中，作者采用梅尔频谱作为模型输入。此外，论文作者采用了FCN-4, FCN-5, FCN-6, FCN-7进行试验，其中FCN-4在MagnaTagATune Dataset与Million Song Dataset 都得到较高的AUC，且参数较少。因此作者最终采用FCN-4模型结构。



FCN-4网络结构

### 实验流程

环境准备：使用分享镜像创建ECS服务器

项目部署：下载项目代码到ECS，并解压部署，安装依赖库

模型转换：将AIR格式的模型，转换为昇腾310支持的om模型

结果推理：运行sh脚本，调用mindx SDK工具包来执行推理，对data路径下的音频文件进行分类，识别出该音频文件属于哪一种音乐流派

## 实验目的

了解如何通过华为云私有镜像搭建弹性云服务器（ECS）；了解如何在ECS中部署基于MindX SDK的语音分类任务并运行，了解如何通过ATC工具将其他模型格式转换为昇腾310支持的om格式模型。

## 实验过程



下载并部署项目

根据实验流程，需要安装zip、tree命令，输入：

apt install tree

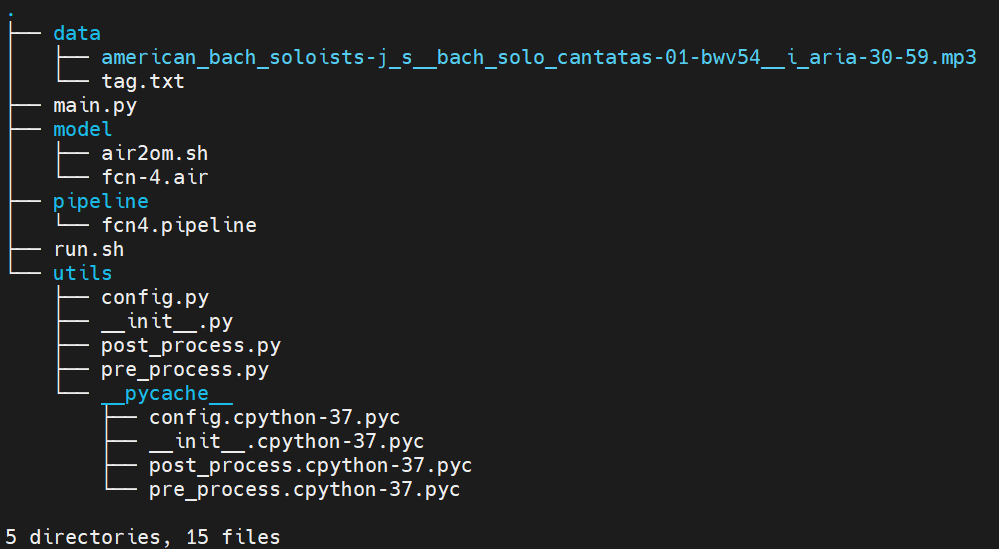
使用wget命令下载项目代码到ECS中（项目代码链接为：<https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/ASR/fcn4_sdk.zip>），使用zip命令进行解压，然后进入到项目文件夹中，输入如下命令：

wget <https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/ASR/fcn4_sdk.zip> --no-check-certificate

unzip fcn4\_sdk.zip

cd fcn4\_sdk

tree



项目代码树状目录

从上面的项目文件中我们可以完整看到整个项目中文件信息：

* data文件夹：待推理的音乐文件、分类类别的标签文件
* model文件夹：推理模型、模型转换脚本
* pipline文件夹：调用mindX SDK推理所需的配置文件
* main.py文件：推理任务执行业务代码
* utils文件夹：数据预处理和后处理文件、程序运行配置文件
* run.sh：整个项目运行的脚本文件。

模型转换

通过ATC工具（注：使用ATC工具，我们可以把Tensorflow、Caffe框架训练完成的模型转换为Ascend芯片支持的om模型，从而可以在Ascend环境下使用）。更多信息可参考：

<https://www.hiascend.com/document/detail/zh/CANNCommunityEdition/502alpha3infer/atctool/atlasatc_16_0006.html>

模型转换脚本内容如下（以下命令无需单独运行）

# air模型转om只支持静态batch，所以脚本中batchsize=1

atc --framework=1 \

--model=fcn-4.air \

--output=fcn-4\_bs1 \

--input\_format=NCHW \

--soc\_version=Ascend310 \

--log=error

其中参数的含义解释如下：

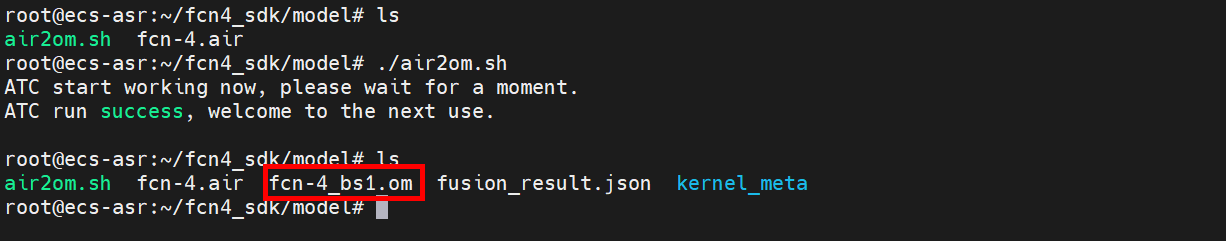
* --framework：原始框架类型。
* --model：原始模型文件路径与文件名。
* --output：如果是开源框架的网络模型，存放转换后的离线模型的路径以及文件名；如果是单算子描述文件（json格式），存放转换后的单算子模型的路径。
* --input\_format：输入数据格式。
* --soc\_version：模型转换时指定芯片版本。
* --log：设置ATC模型转换过程中日志的级别。

在model文件夹中先赋予SHELL脚本执行权限，运行脚本。输入以下命令来执行模型转换

cd /root/fcn4\_sdk/model

chmod 777 air2om.sh

./air2om.sh



转换完成的om模型

执行推理任务

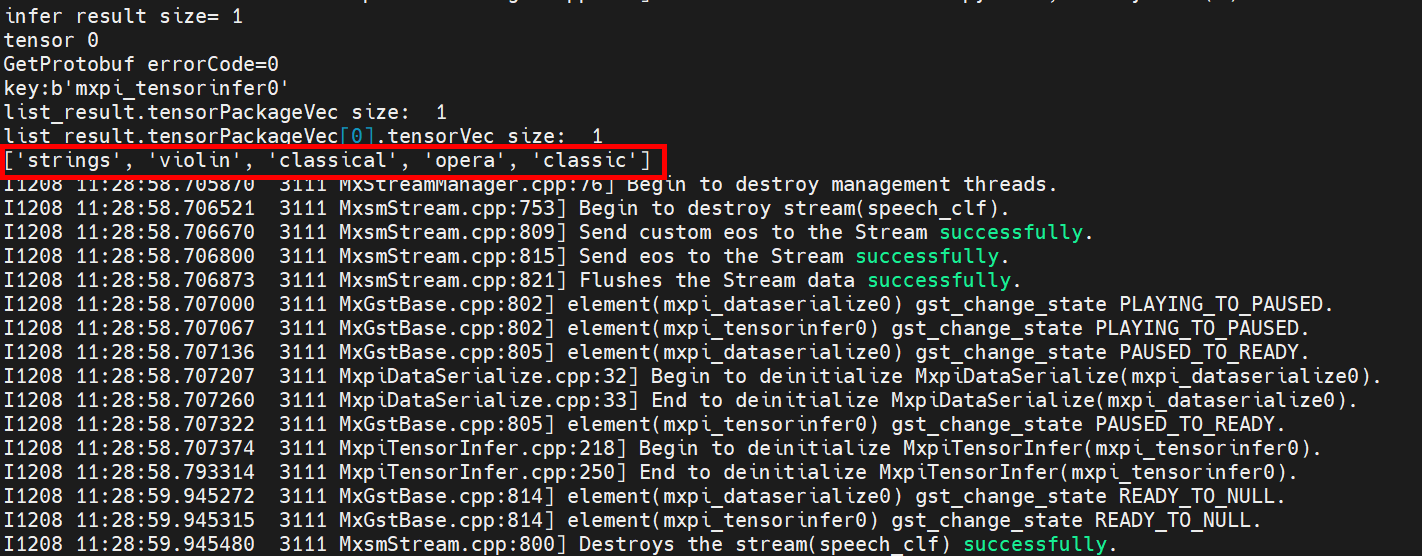
接下来，先赋予run.sh脚本执行权限，运行run.sh脚本完成音乐文件的识别，输入：

cd /root/fcn4\_sdk

chmod 777 run.sh

./run.sh

从下图的推理结果中可以，我们通过调用MindX SDK接口完成了语音分类任务，输出了wav音乐片段可能属于的top5类别（依次为strings：弦乐；violin:小提琴乐曲；classical：古典音乐；opera:歌剧；classic：经典音乐）。



语音分类结果

## 部分代码介绍

### main脚本构建

main.py脚本定义了语音识别任务的完整流程。主要由以下几个部分构成：

* 数据预处理：由于当前sdk并未提供音频插件，因此音频文件转梅尔频谱需要用到librosa库进行变化。详细的代码参考pre\_process.py。
* 数据发送与结构：利用sendProtobuf与getProtobuf，发送与接收tensor数据。详细可参考“sdk-中文语音理解样例4.2章节”（链接：<https://bbs.huaweicloud.com/forum/forum.php?mod=viewthread&tid=164495>）。
* 数据后处理：由于当前sdk并未提供音频插件，因此需要将模型输出转为结果类别。详细的后处理代码参考post\_process.py。

全部代码如下：

import os

import numpy as np

import MxpiDataType\_pb2 as MxpiDataType

from StreamManagerApi import StreamManagerApi, InProtobufVector, MxProtobufIn, StringVector

from utils.config import config as cfg

from utils.pre\_process import make\_model\_input, make\_model\_input\_single

from utils.post\_process import ClassFeaturizer

if \_\_name\_\_ == "\_\_main\_\_":

#创建语音识别任务流的管理接口，并初始化

stream\_manager = StreamManagerApi()

ret = stream\_manager.InitManager()

if ret != 0:

print("Failed to init stream manager, ret=%s" % str(ret))

exit()

# 读取配置好的pipeline文件，根据该配置内容设置任务流

with open(cfg.pipeline\_path, 'rb') as f:

pipeline\_str = f.read()

ret = stream\_manager.CreateMultipleStreams(pipeline\_str)

if ret != 0:

print("Failed to create Stream, ret=%s" % str(ret))

exit()

#读取待推理的语言文件

feature\_data = make\_model\_input\_single(cfg.wav\_file\_path) # feature\_data.shape=(n,1,96,1366)

print("input data shape = {}".format(feature\_data.shape))

# 实现sendProtobuf

# step1: 生成tensor pkg list，插入每一个张量数据

mxpi\_tensor\_package\_list = MxpiDataType.MxpiTensorPackageList()

for i in range(feature\_data.shape[0]):

feat\_data = feature\_data[i]

feat\_data = feat\_data[np.newaxis, :]

tensor\_package\_vec = mxpi\_tensor\_package\_list.tensorPackageVec.add()

tensorVec = tensor\_package\_vec.tensorVec.add()

tensorVec.memType = 1 # 0->host, 1->Device, 2->dvpp

tensorVec.deviceId = cfg.device\_id

# Compute the number of bytes of feature data.

tensorVec.tensorDataSize = int(feat\_data.shape[1] \* feat\_data.shape[2] \* 4)

tensorVec.tensorDataType = 0 # float32

for j in feat\_data.shape:

tensorVec.tensorShape.append(j)

tensorVec.dataStr = feat\_data.tobytes()

# step2：装入protobufIn

protobuf = MxProtobufIn()

protobuf.key = cfg.tensor\_key

protobuf.type = b'MxTools.MxpiTensorPackageList'

protobuf.protobuf = mxpi\_tensor\_package\_list.SerializeToString()

# step3: 装入protobufVec

protobuf\_vec = InProtobufVector()

protobuf\_vec.push\_back(protobuf)

# step4：sendProtobuf

unique\_id = stream\_manager.SendProtobuf(cfg.stream\_name, cfg.in\_plugin\_id, protobuf\_vec)

if unique\_id < 0:

print("Failed to send data to stream.")

exit()

# 实现getProtobuf

# step1: 定义StringVector，填入张量输出插件名

result = []

key\_vec = StringVector()

key\_vec.push\_back(cfg.key\_vec\_str)

infer\_result = stream\_manager.GetProtobuf(cfg.stream\_name, cfg.in\_plugin\_id, key\_vec)

if infer\_result.size() == 0:

print("infer\_result is null")

exit()

print("infer result size=", len(infer\_result)) # debug用，这里长度应该为feature\_data.shape[0]

for i in range(len(infer\_result)):

if infer\_result[i].errorCode != 0:

print("GetProtobuf error. errorCode=%d" % infer\_result[i].errorCode)

exit()

print(f"tensor {i}")

print("GetProtobuf errorCode=%d" % infer\_result[i].errorCode)

print("key:" + str(infer\_result[i].messageName))

# step 2：获取每一个张量数据，保存到result中

list\_result = MxpiDataType.MxpiTensorPackageList()

for i in range(len(infer\_result)):

list\_result.ParseFromString(infer\_result[i].messageBuf)

print("list\_result.tensorPackageVec size: ", len(list\_result.tensorPackageVec))

print("list\_result.tensorPackageVec[0].tensorVec size: ", len(list\_result.tensorPackageVec[0].tensorVec))

result.append(

np.frombuffer(list\_result.tensorPackageVec[0].tensorVec[0].dataStr, dtype=np.int32)

)

result = np.array(result)

# 输入n,50。返回歌曲的类别，top5

tag\_file\_path = os.path.join(os.getcwd(), "data/tag.txt")

class\_featurizer = ClassFeaturizer(tag\_file\_path)

print(class\_featurizer.output\_class(result, cfg.top\_num\_class))

### 配置文件

在utils文件夹中的config.py文件为实验执行过程中用到的部分参数信息，详细介绍如下：

pipeline\_path: pipeline路径

stream\_name: pipeline名字

tensor\_key: 接受数据的插件名

in\_plugin\_id: appsrc序号

device\_id: pipeline中设置的device id

key\_vec\_str: 获取tensorinfer后的值

wav\_file\_path: 推理音频文件路径

top\_num\_class: 预测topK类别

### Pipeline任务流配置介绍

本实验中用到的pipeline文件的代码及注释如下所示（使用时需要删除多余的注释和空格）

{

"speech\_clf": {

"stream\_config": {

"deviceId": "0" # 设置npu设备号

},

"appsrc0": {

"props": {

"blocksize": "409600"

},

"factory": "appsrc", # # 数据入口，从main.py中传送预处理后的数据

"next": "mxpi\_tensorinfer0"

},

"mxpi\_tensorinfer0": {

"props": {

"dataSource": "appsrc0",

"modelPath": "model/fcn-4\_bs1.om" #推理模型的路径

},

"factory": "mxpi\_tensorinfer",

"next": "mxpi\_dataserialize0"

},

"mxpi\_dataserialize0": {

"props": {

"outputDataKeys": "mxpi\_tensorinfer0"

},

"factory": "mxpi\_dataserialize",

"next": "appsink0"

},

"appsink0": {

"factory": "appsink"

}

}

}

此pipeline中用到的插件字段介绍如下（更多插件字段，可查看官方文档[<https://support.huawei.com/enterprise/zh/doc/EDOC1100221065/2b38b658>](https://support.huawei.com/enterprise/zh/doc/EDOC1100221065/2b38b658)）：

* stream\_config：该插件字段用于设定整个项目的基础配置，比如推理设备编号等。
* appsrc：Gstreamer原生插件。配合SendData接口与SendDataWithUniqueId接口使用，向Stream中发送数据，appsrc将数据发给下游元件，当且仅当Stream中存在appsrc插件时以上两个接口有效。
* mxpi\_tensorinfer：对输入的张量进行推理。此pipeline中有前后两部分推理，故定义了两个推理插件。
* mxpi\_dataserialize：将stream结果组装成json字符串输出。用户可通过在配置文件中填写元件名，来选择需要输出的结果。
* appsink：Gstreamer原生插件。配合GetResult接口与GetResultWithUniqueId接口使用，从stream中获取数据，当且仅当stream中存在appsink插件时，以上两个接口有效。

### 音频数据预处理

utils/pre\_process.py脚本实现了数据在进入推理模型之前的预处理任务，主要功能是将读取的音频文件按照指定的窗口和长度进行采样，并计算出音频数据的梅尔频谱。代码如下：

import numpy as np

import librosa

SR = 12000 #音频文件采样率

N\_FFT = 512 #窗口大小

N\_MELS = 96 #mel bands的数量

HOP\_LEN = 256 #连续帧之间的样本数

DURA = 29.12 #to make it 1366 frame..

# 根据训练脚本，使用相同预处理代码

#计算mel频谱

def compute\_melgram(src):

"""

extract melgram feature

"""

logam = librosa.core.amplitude\_to\_db

melgram = librosa.feature.melspectrogram

ret = logam(melgram(y=src, sr=SR, hop\_length=HOP\_LEN, n\_fft=N\_FFT, n\_mels=N\_MELS))

ret = ret[np.newaxis, np.newaxis, :] # fixed: [96,1366] -> [1,1,96,1366]

return ret

def make\_model\_input(wav\_file):

"""

:param wav\_file: 输入一个音频文件

:return: 返回ndarray数据，shape=[n, 1, 96, 1366]。音频长度不一，导致n也不一

"""

src, \_ = librosa.load(wav\_file, sr=SR) # whole signal

n\_sample = src.shape[0]

n\_sample\_fit = int(DURA \* SR)

if n\_sample < n\_sample\_fit: # if too short

n\_clip = 1

src = np.hstack((src, np.zeros((int(DURA \* SR) - n\_sample,))))

elif n\_sample > n\_sample\_fit: # if too long

n\_clip = n\_sample // n\_sample\_fit

src = src[(n\_sample - n\_sample\_fit\*n\_clip) // 2:(n\_sample + n\_sample\_fit\*n\_clip) // 2]

result = []

for i in range(n\_clip):

offset = i \* n\_sample\_fit

result.append(compute\_melgram(src[offset : offset+n\_sample\_fit]))

return np.concatenate(result, axis=0) # [(1,1,x,x), (1,1,x,x)] -> [n,1,x,x]

def make\_model\_input\_single(wav\_file):

src, \_ = librosa.load(wav\_file, sr=SR) # whole signal

n\_sample = src.shape[0]

n\_sample\_fit = int(DURA \* SR)

if n\_sample < n\_sample\_fit: # if too short

src = np.hstack((src, np.zeros((int(DURA \* SR) - n\_sample,))))

elif n\_sample > n\_sample\_fit: # if too long

src = src[:n\_sample\_fit]

return compute\_melgram(src) # [1,1,x,x]

if \_\_name\_\_ == "\_\_main\_\_":

wav\_path = "../data/sample-qingtian.mp3"

feature\_data = make\_model\_input(wav\_path)

print(feature\_data.shape)

### 模型推理结果后处理

utils/post\_process.py脚本实现了数据在结果模型推理之后的数据信息，与50种音乐类别进行一一映射（50种音乐类型的标签路径为：data/tag.txt），然后将推理结果中概率最大的前n个类别的信息返回。代码如下：

import numpy as np

class ClassFeaturizer(object):

def \_\_init\_\_(self, tag\_file\_path):

self.tag\_file\_path = tag\_file\_path

self.class\_array = []

self.token\_to\_index = {}

self.index\_to\_token = {}

#读取标签文件并生成token\_to\_index，index\_to\_token两个字典文件

f = open(self.tag\_file\_path, "r")

lines = f.readlines()

index = 0

for line in lines:

line = line.strip()

self.token\_to\_index[line] = index

self.index\_to\_token[index] = line

self.class\_array.append(line)

index += 1

f.close()

def encode(self, token):

return self.token\_to\_index[token]

def decode(self, index):

return self.index\_to\_token[index]

#将推理结果转换为输出信息的函数

def output\_class(self, infer\_result, top\_num\_class):

"""

:param infer\_result: ndarray, shape=[n,50]

top\_num\_class: num of class to output

:return: top n class

"""

sums = np.sum(infer\_result, axis=0)

argsort = np.argsort(sums)[::-1][:top\_num\_class] # top5

top\_class = []

for idx in argsort:

if sums[idx] < 1:

break

top\_class.append(self.index\_to\_token[int(idx)])

return top\_class

if \_\_name\_\_ == "\_\_main\_\_":

np.random.seed(1)

infer\_result = np.random.rand(5, 50)

infer\_result[infer\_result > 0.98] = 1

infer\_result[infer\_result < 0.98] = 0

tag\_file\_path = "../data/tag.txt"

class\_featurizer = ClassFeaturizer(tag\_file\_path)

top\_class = class\_featurizer.output\_class(infer\_result, 5)

print(top\_class)

### 运行脚本

需要按照用户安装软件路径来设置，如果部分变量已在用户目录下的“.bashrc”中设置，可以注释该脚本中的设置，详情可参考sdk软件包中的sample脚本。

set -e

CUR\_PATH=$(cd "$(dirname "$0")" || { warn "Failed to check path/to/run.sh" ; exit ; } ; pwd)

# Simple log helper functions

info() { echo -e "\033[1;34m[INFO ][MxStream] $1\033[1;37m" ; }

warn() { echo >&2 -e "\033[1;31m[WARN ][MxStream] $1\033[1;37m" ; }

# export MX\_SDK\_HOME=${CUR\_PATH}/../../..

export LD\_LIBRARY\_PATH=${MX\_SDK\_HOME}/lib:${MX\_SDK\_HOME}/opensource/lib:${MX\_SDK\_HOME}/opensource/lib64:/usr/local/Ascend/ascend-toolkit/latest/acllib/lib64:${LD\_LIBRARY\_PATH}

export GST\_PLUGIN\_SCANNER=${MX\_SDK\_HOME}/opensource/libexec/gstreamer-1.0/gst-plugin-scanner

export GST\_PLUGIN\_PATH=${MX\_SDK\_HOME}/opensource/lib/gstreamer-1.0:${MX\_SDK\_HOME}/lib/plugins

#to set PYTHONPATH, import the StreamManagerApi.py

export PYTHONPATH=$PYTHONPATH:${MX\_SDK\_HOME}/python

python3 main.py

exit 0

## 实验总结

本章实验在华为云ECS中上，调用MindX SDK中已实现的FCN-4模型来完成了音乐片段所属的流派分类任务，从中可以了解到如何使用镜像文件搭建ECS服务器；了解FCN-4模型的结果及其在语音分类任务中的应用；了解如何通过MindX SDK通过简单配置快速完成一个语音分类接口的开发。

# 附录：环境准备

## 获取镜像

本节指导如何获取华为云账号下项目ID，以便方便导入实验所需镜像。

获取项目ID

进入华为云网址：<https://www.huaweicloud.com/>

点击 “账号中心”



账号中心

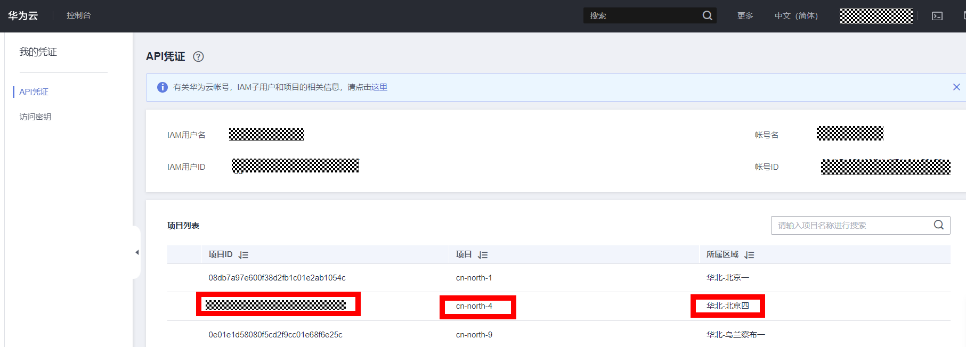
点击“我的凭证”



我的凭证

获取项目ID

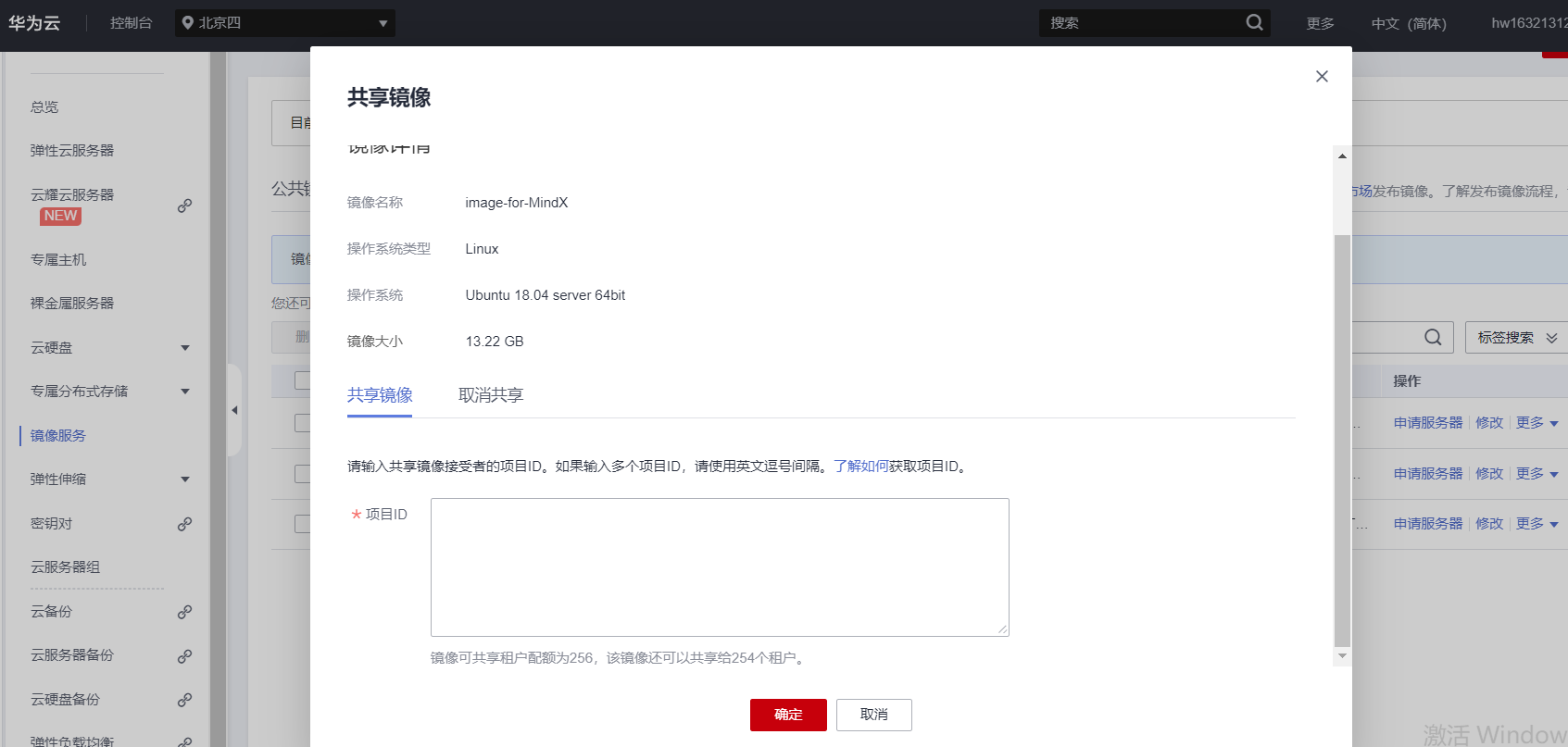
选择 项目为“cn-north-4”的项目ID,[高校教师通过邮件发送至yanxukun1@huawei.com](mailto:高校教师通过邮件发送至chenxiuhong@huawei.com) ,华为技术支持给高校老师分发镜像“image-for-Speech\_multi\_classification”；若学生则需要自行发送给任课老师，由老师发放。



项目ID

共享镜像

（镜像分享者需要操作）把项目ID复制到项目ID 里,点击确定。



共享镜像

## 创建ECS 弹性服务器

### 创建ECS弹性云服务器

进入华为云ECS控制台

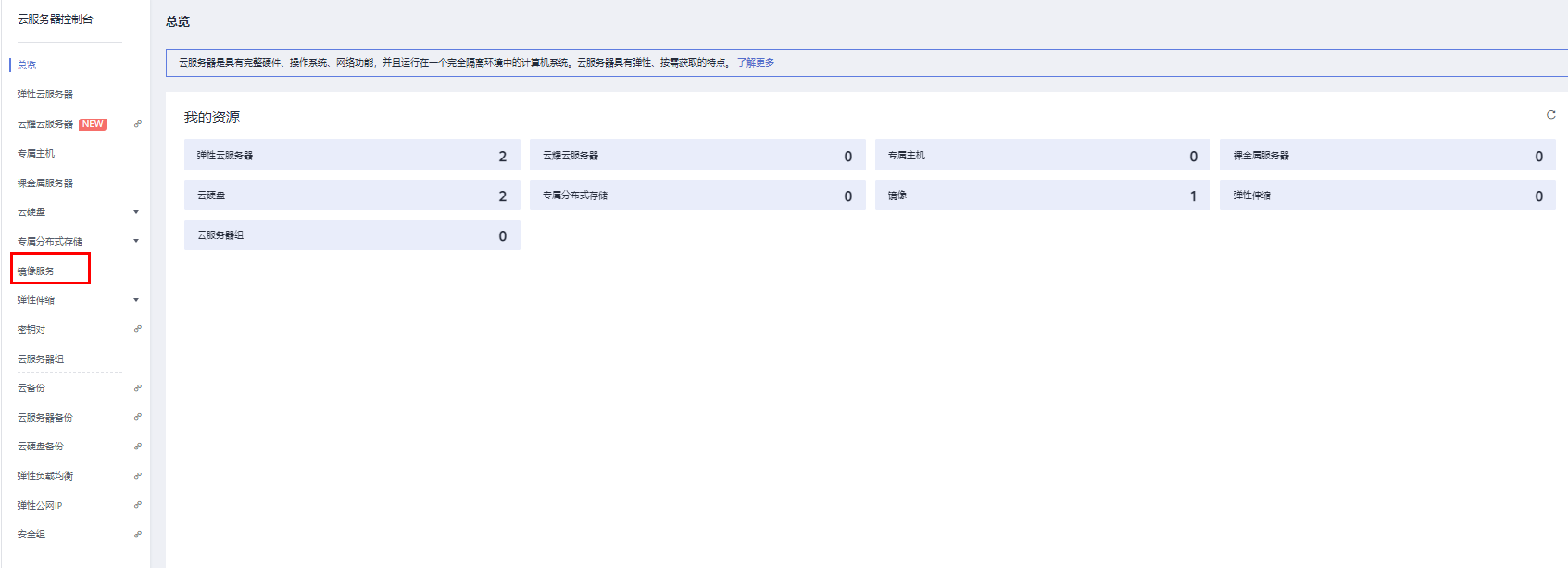
在[华为云ECS主页](https://www.huaweicloud.com/product/ecs.html)，点击“管理控制台”进入ECS的管理页面。



华为云ECS主页

接受共享镜像

在左侧菜单栏中选择“镜像服务”



镜像服务

选择“共享镜像”，“全部接受”。

共享镜像

创建弹性云服务器

控制台区域选择“华北－北京四”，在左侧菜单栏中选择“弹性云服务器”，在右上角“购买弹性云服务器”。



ECS 控制台

在“基础配置”里，选择如下配置：

计费模式：按需计费。

区域：华北-北京四。

可用区：随机分配。

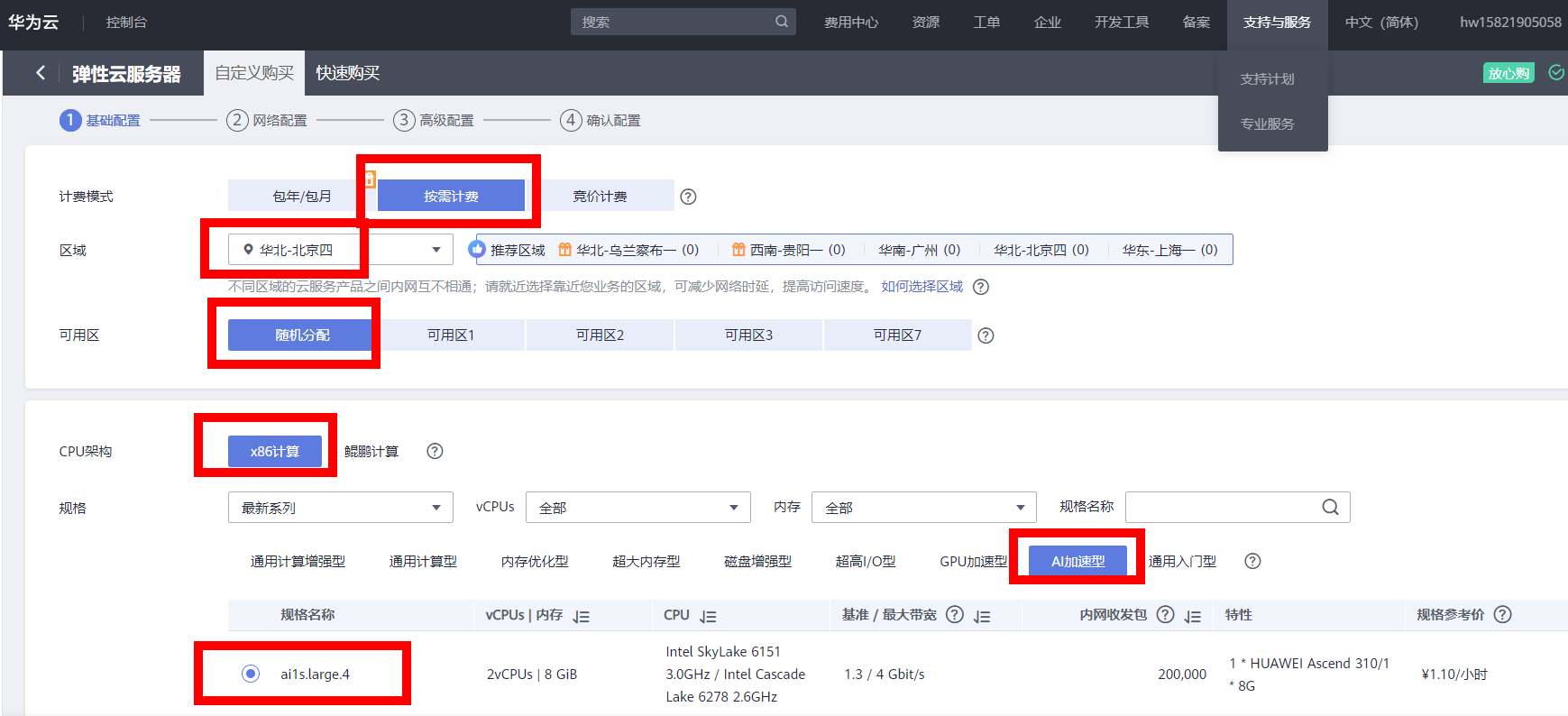
CPU架构：x86计算。

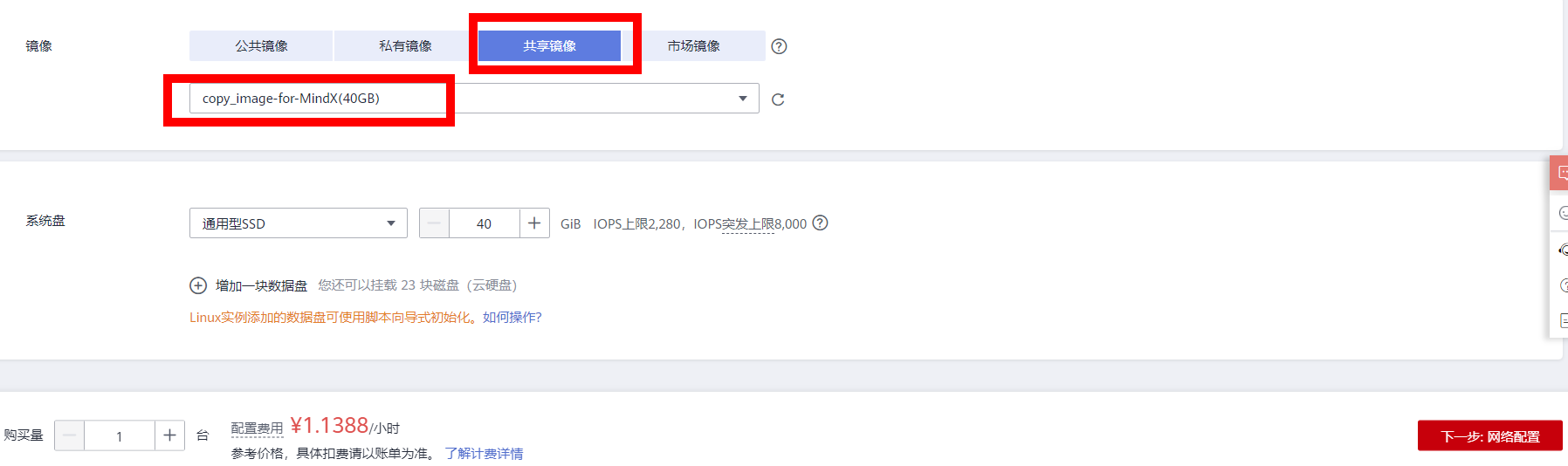
规格： AI加速型 | ai1s.large.4 | 2vCPUs | 8GiB | 1 \* HUAWEI Ascend 310/1 \* 8G。

镜像：公享镜像，image-for-MindX。

系统盘：通用型SSD，40GB。

如图所示：





ECS基础配置

基础配置完成之后，选择“下一步:网络配置”。　　　　　　　　　　　　　　　　　　　在“网络配置”里，选择如下配置：

网络：可以前往控制台创建新的虚拟私有云。

拓展网卡：无。

安全组：可以新建安全组。

弹性公网IP：现在购买。

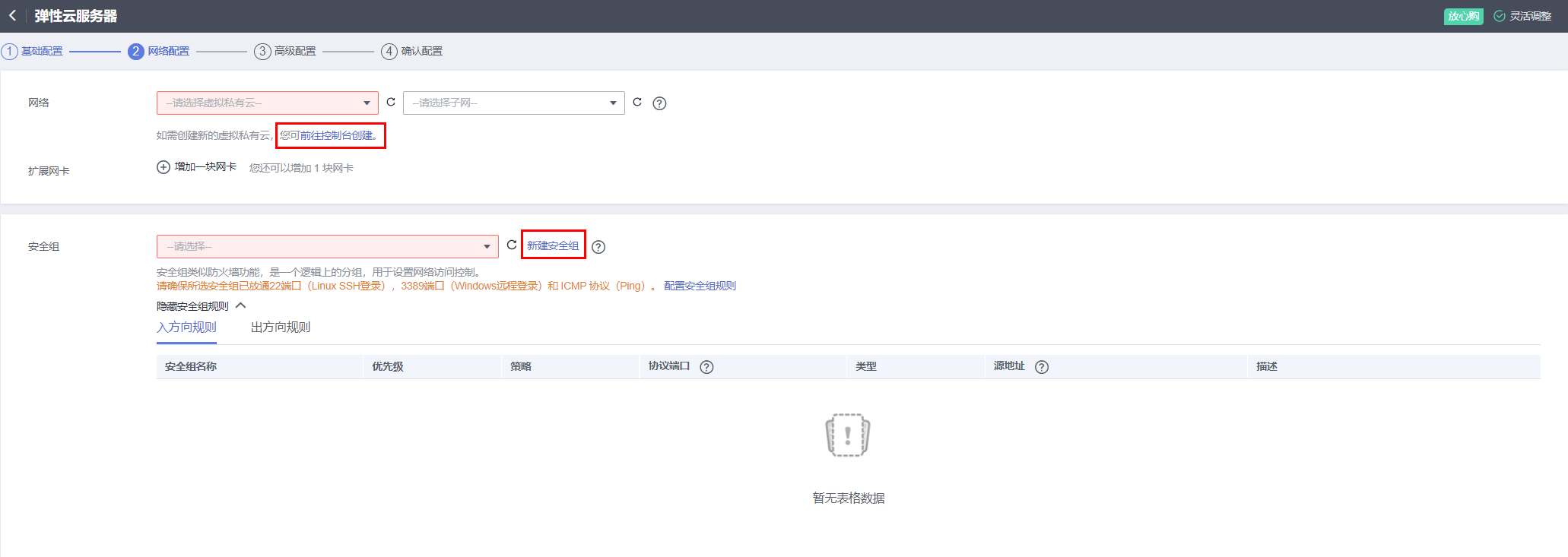
线路：全动态BGP。

公网带宽：按流量计费。

宽带大小：自定义，200Mbit/s。

释放行为：勾选随实例释放。

如图所示：





网络配置完成之后，选择“下一步”，进行高级配置。

在“高级配置”里，选择如下配置：

云服务器名称：可以自定义。

登录凭证：密码。

用户名：root。

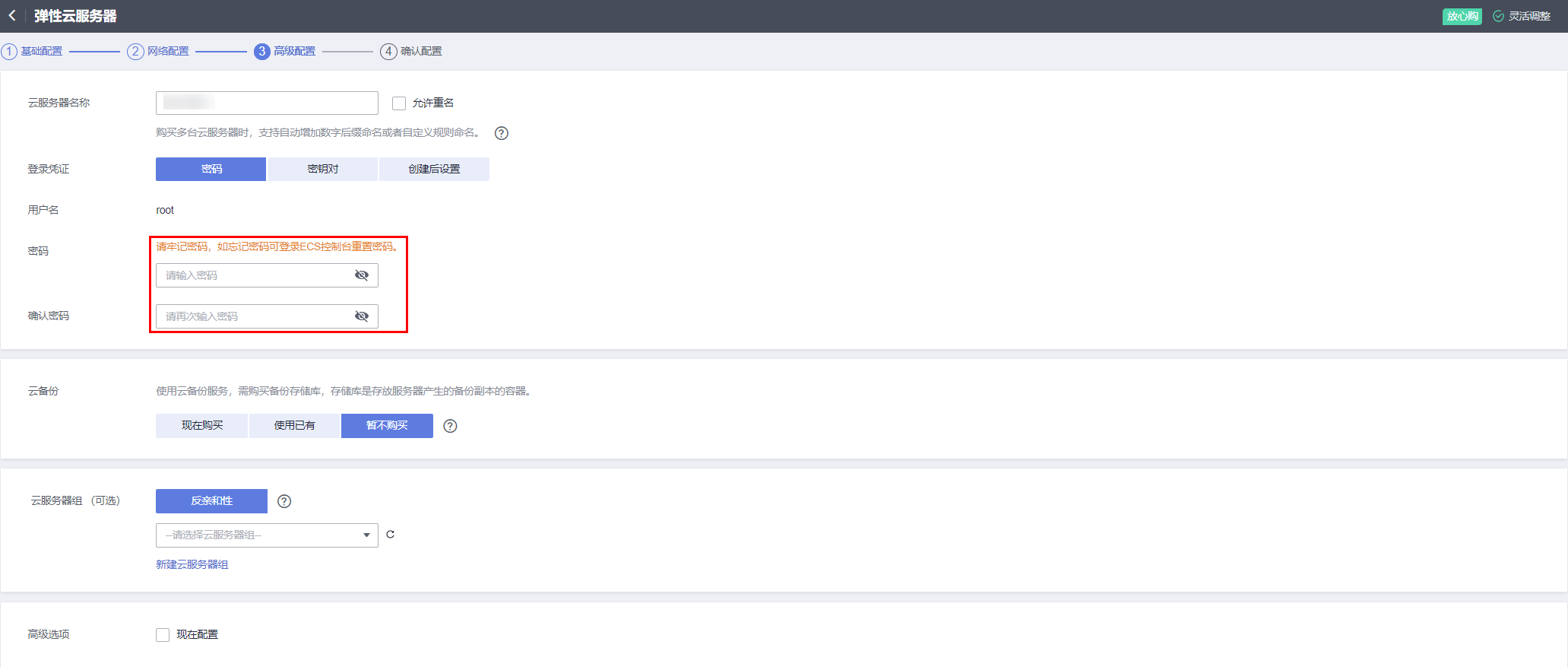
密码：自定义（后续登录使用，需谨记）。

云备份：暂不购买。

云服务器组：无。

高级选项：无。

如图所示：

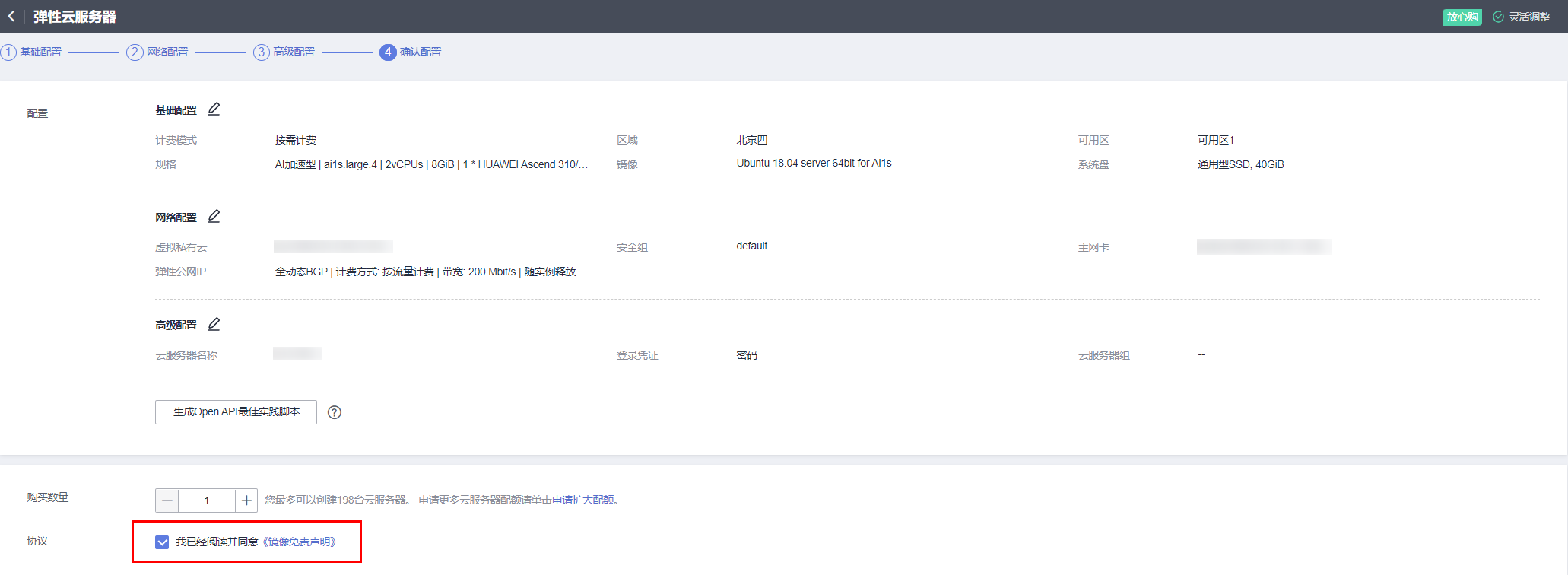


ECS高级配置

高级配置完成之后，选择“下一步”，确认配置。

在“确认配置”里，选择如下配置：

协议：勾选我已阅读并同意《镜像免责声明》。

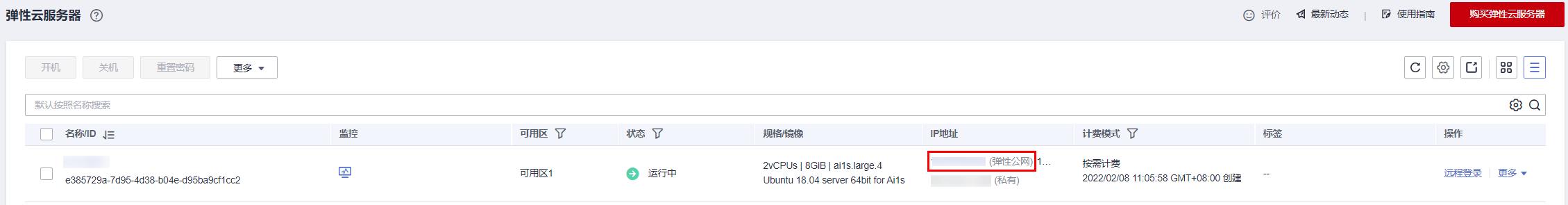


ECS 确认配置

确认配置完成之后，选择“立即购买”。

“任务提交成功”之后，选择“返回服务器列表”即可回到弹性云服务器的管理控制台，看到已创建的ECS弹性云服务器正在运行中。

\* 注意在“IP地址”显示的弹性公网IP地址，后续会用到。



ECS弹性云服务器创建成功

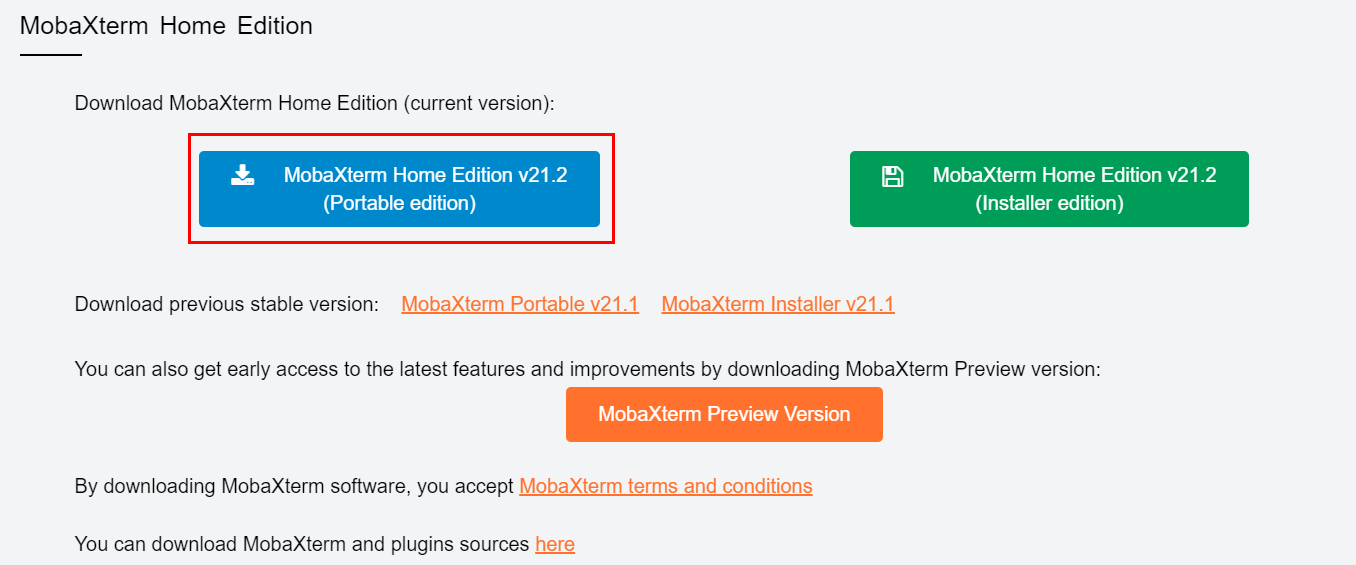
### MobaXterm连接ECS

下载MobaXterm （21.2及以上版本皆可）

进入MobaXterm 的官网主页：[**https://mobaxterm.mobatek.net/**](https://mobaxterm.mobatek.net/)

选择“Home Edition”，下载“MobaXterm Home Edition v21.x （Portable edition）”。

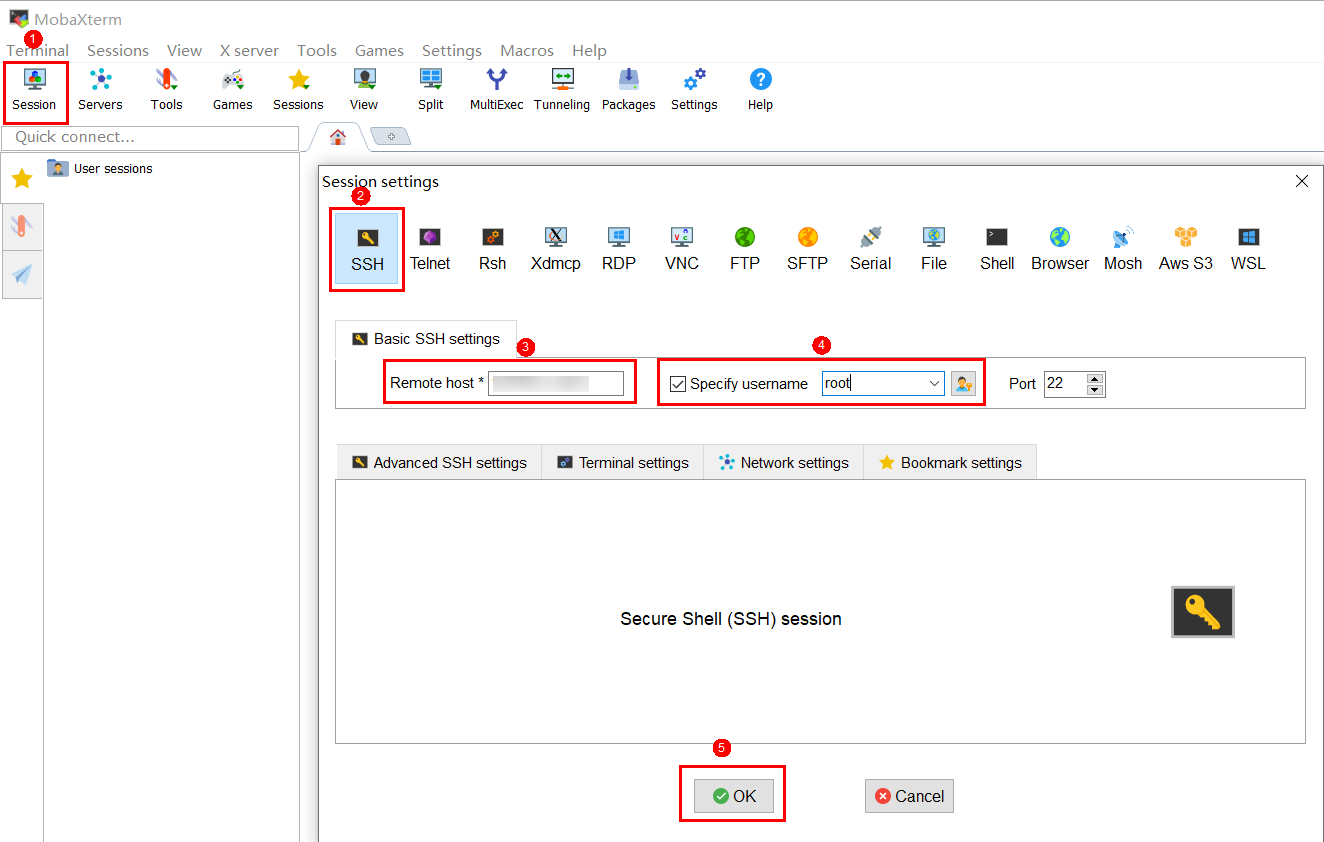
下载完成之后解压MobaXterm\_Portable\_v21.x.zip文件。



下载界面

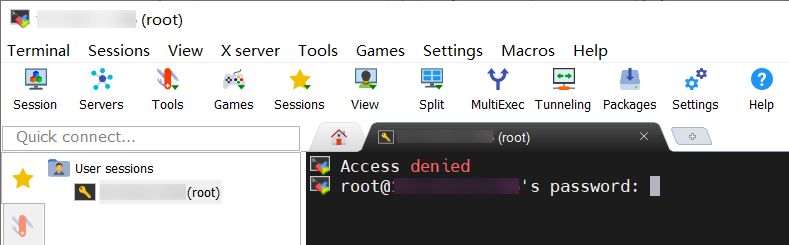
使用MobaXterm远程连接弹性云服务器

进入解压后的MobaXterm\_Portable\_v21.x文件夹，打开MobaXterm\_Personal\_21.x.exe文件，选择菜单栏的“Session”，之后进入“Session settings”页面，远程链接选择“SSH”协议，输入图3-12 ECS弹性云服务器创建成功时显示的弹性公网IP地址，选择指定用户名“Specify username”，用户名为“root”，配置完成之后选择“OK”提交。



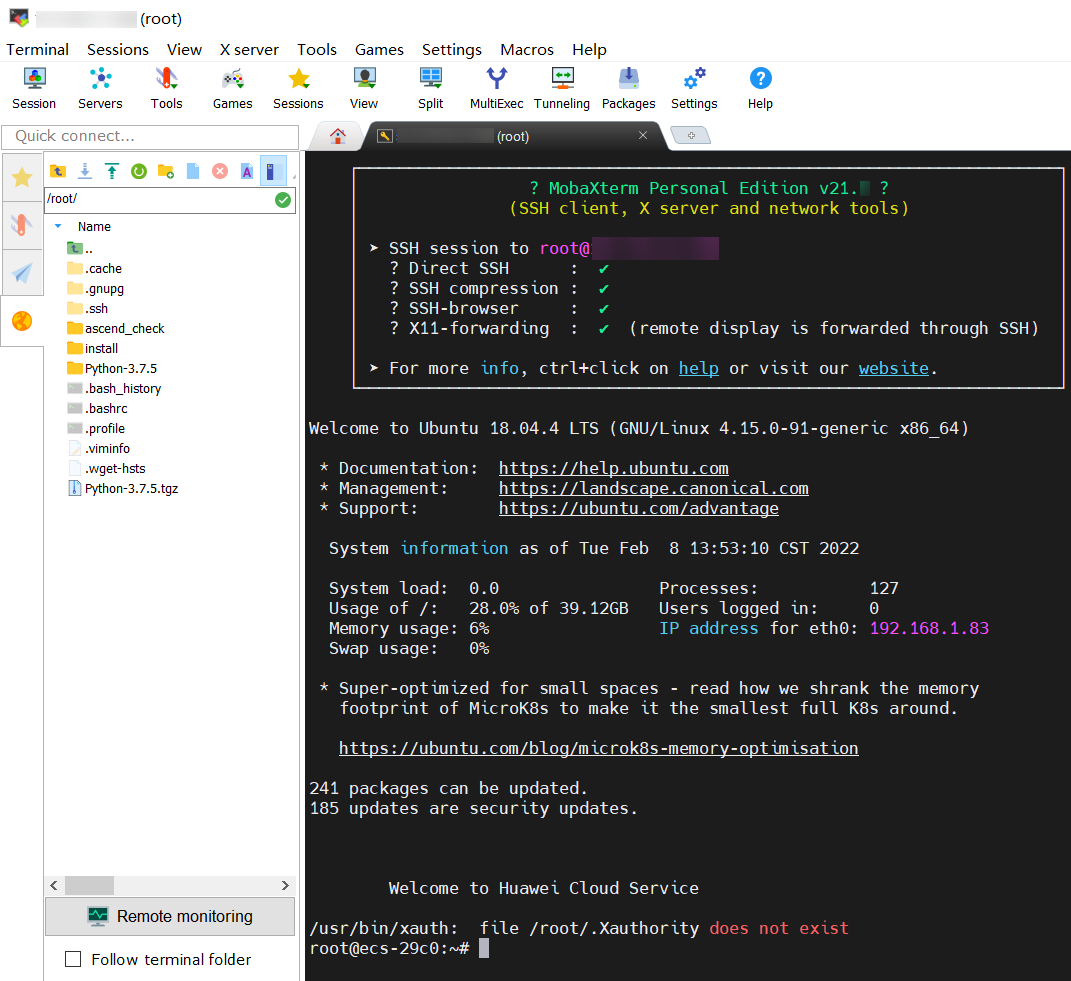
MobaXterm远程连接

MobaXterm 登录ECS需输入密码，在[3.2.1 创建ECS弹性云服务器](#_创建ECS弹性云服务器)的步骤3中，高级配置里已自定义了弹性云服务器root用户密码，在此输入即可。



MobaXterm 登录ECS需输入密码

MobaXterm远程链接弹性云服务器成功，后续还需进一步配置弹性云服务器的云上环境。



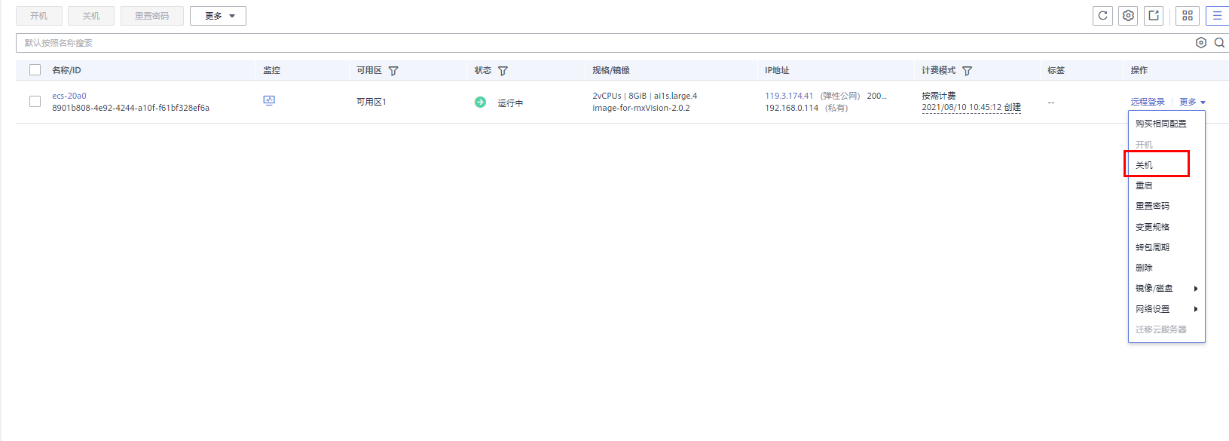
MobaXterm 登录成功界面

## 关闭ECS弹性云服务器

实验完成后，请及时关闭服务器，防止资源浪费。

关闭服务器

点击“更多”，选择“关机”。



关机界面

删除服务器

完成实验后，选择删除服务器，释放资源。



删除服务器



删除资源