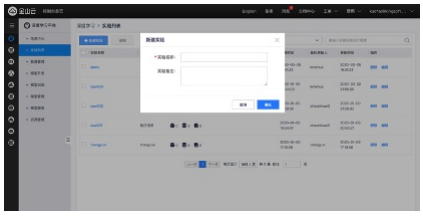
目录

实验管理

实验可理解为子项目，为了一个目的而进行深度学习的任务称为实验，一个实验可以使用多种不同的算法或者同一个算法的不同参数组合来实现。

实验创建

点击实验列表，进入实验管理页面。点击【新建实验】，填写相应的信息即可完成实验的创建。 ::: hl js-center



:::

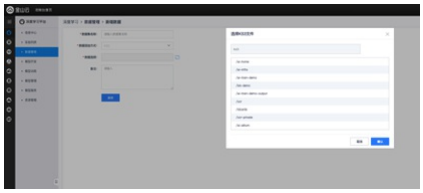
实验详细信息查看

点击实验名称，可查看实验的详细信息。 ::: hl js-center

::: 在实验中，可查看关联的模型训练、模型及模型服务。

数据管理

数据管理支持将KS3中的数据注册到数据列表中 点击数据管理，可进入数据管理页面。点击新增数据，可进行数据新增。 ::: hl js-center



:::

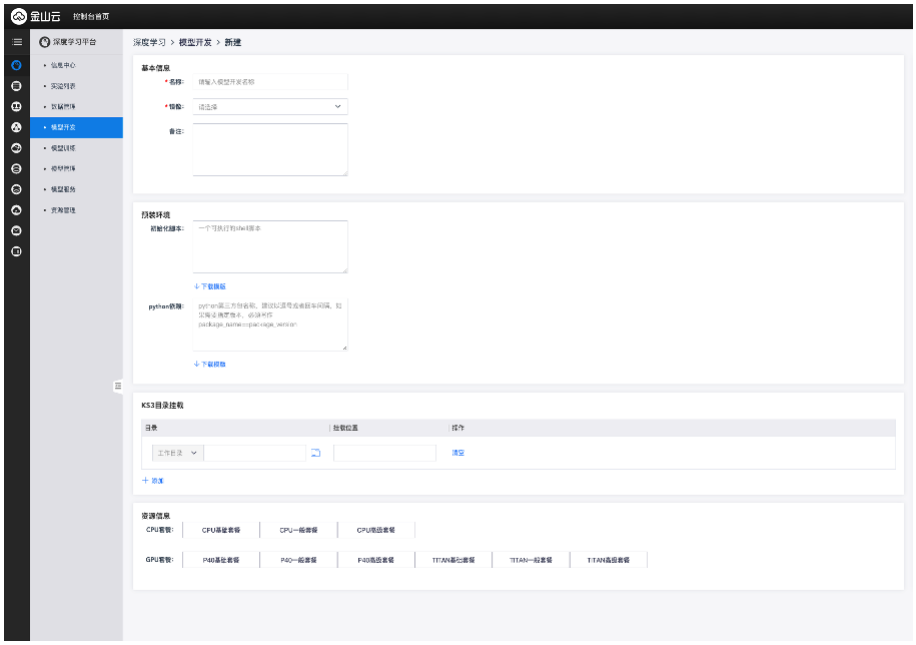
用户可将数据在项目内进行共享，点击共享后，同租户的其他子账户可在公开数据中查看分享的数据 ::: hl js-center

:::

模型开发

1、新建模型开发

用户既可以使用“一键示例模型开发”快速创建模型开发环境，也可以自定义模型开发环境。



用户需指定开发环境名称、选择基础镜像、预装环境、KS3挂载等信息

用户可以通过预装环境，对镜像环境进行自定义，KDL提供两种方式对环境进行自定义：

- 1、通过初始化脚本，用户可以提供一个可执行的shell脚本，对环境进行定制
- 2、Python依赖，用户可以提供一个pip requirements.txt文件，来安装自定义的pip包

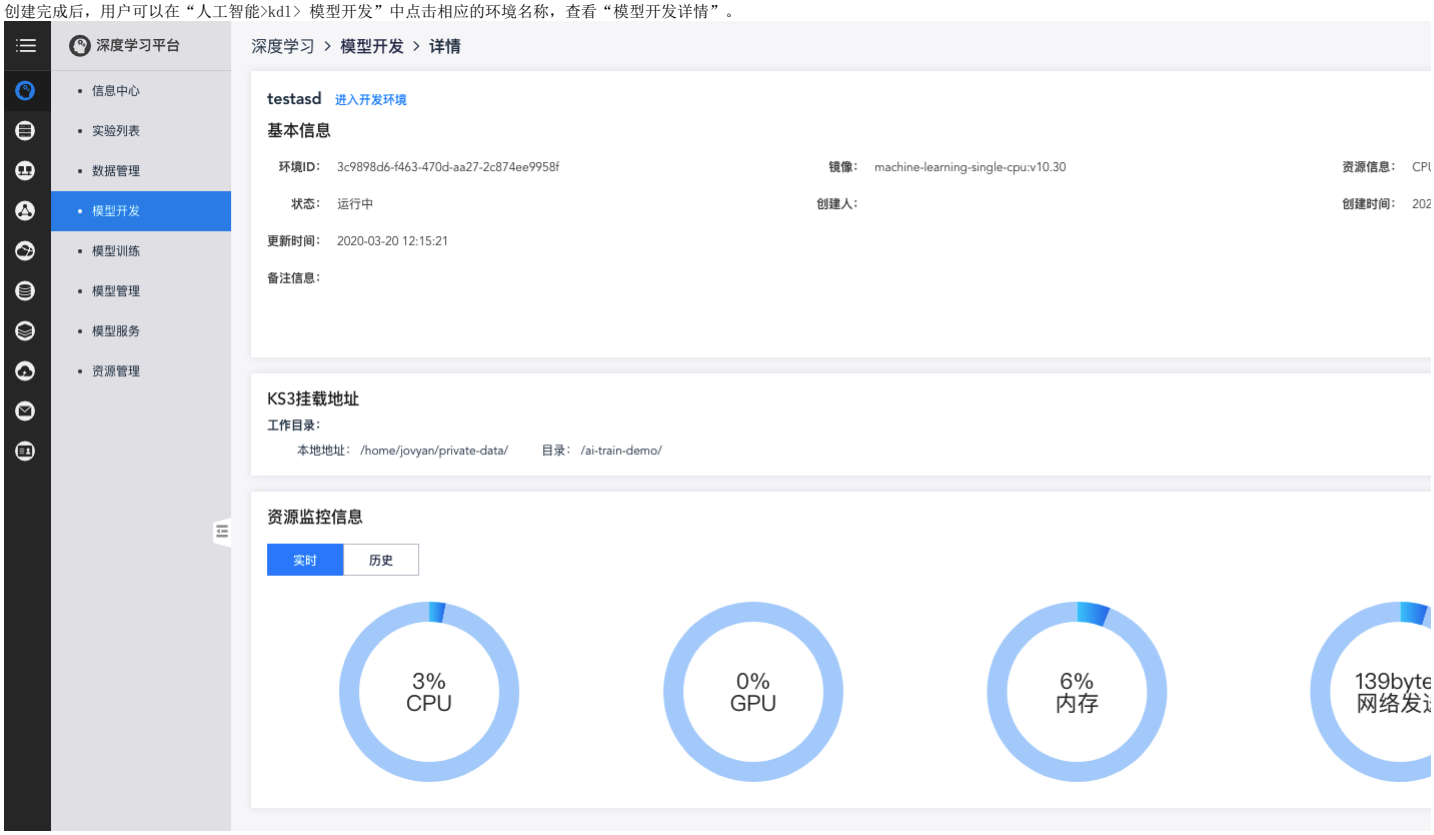
用户可以根据自身需求，给模型开发环境挂载 KS3 Buckets，一方面实现对 KS3 Buckets 操作像操作本地目录一样简单，另一方面也实现数据的共享，和安全。我们建议用户将自己的输入/输出数据都放在 Ks3 上，而 KDL 本身只提供提供计算能力。

用户可根据实际需求，选择不同的资源进行开发环境的创建。

用户根据自身需求，填写/选择模型开发信息和资源信息；创建成功后，用户可以在“人工智能>kdl>模型开发”中查看相关信息。

2、查看模型开发信息

创建完成后，用户可以在“人工智能>kdl> 模型开发”中点击相应的环境名称，查看“模型开发详情”。



- **基本信息**

模型开发环境的基本信息。

- **套餐信息**

用户选择的套餐信息。

- **代码打包上传工具**

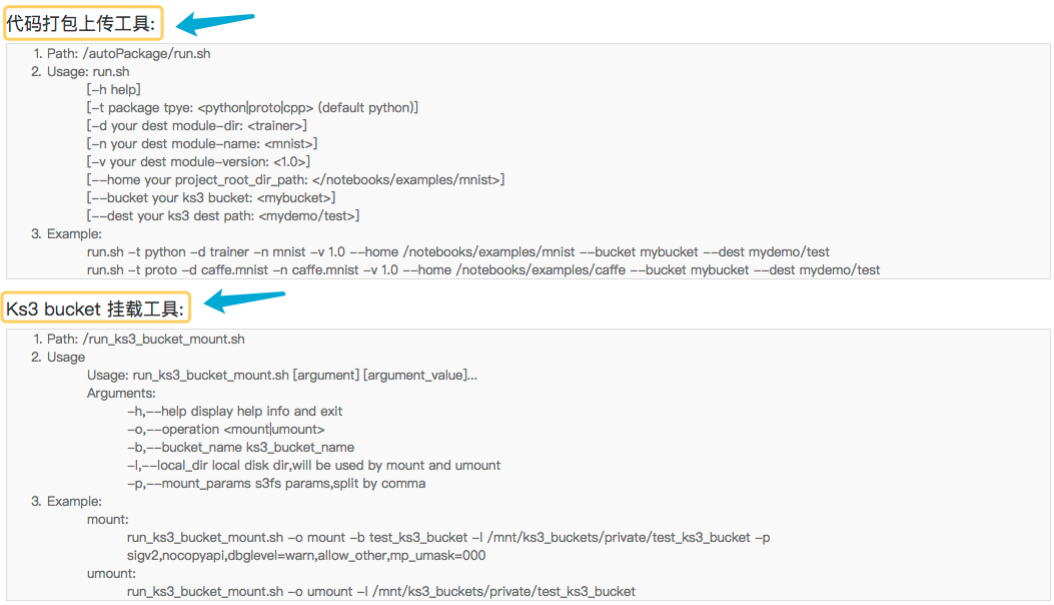
通过该上传工具，用户可以将开发/测试好的代码交由ks3托管。

3、登录模型开发环境

用户可以根据第一步中生成的模型开发环境的有关信息，通过 notebook 登录到开发环境。

在“kdl>模型开发>模型开发列表操作”中点击 打开，即可访问jupyter，进入 jupyter 页面，用户可以在这里查看示例代码，也可以选择 new>Terminal 打开 shell 窗口。输入 “bash” 即可进入我们熟悉的 shell 环境。

4、模型开发相关工具



用户在模型开发环境中调试的代码确认无误后，可以打包上传至 ks3 ，方便下一步模型训练。

- **待打包的代码结构（示例）**

```
notebooks
└── examples
    ├── mnist
        ├── setup.py（trainser 的打包脚本，通过打包工具会自动生成，并覆盖）
        ├── trainer（要打包的代码模块）
```

- **代码打包工具**

打包脚本地址：/autoPackage/run.sh

打包命令：

可在登录模型开发环境后任意位置输入 run

```
[帮助信息]
run
[Focus] No value was specified, ignoring other command line parameters... You can input <-h>
Usage: run.sh
      [-h help]
      [-t package tpye: <python|proto|cpp> (default python)]
      [-d your dest module-dir: <trainer>]
      [-n your dest package-file-name: <mnist>]
      [-v your dest module-version: <1.0>]
      [--home your project_root_dir_path: </notebooks/examples/mnist>]
      [--bucket your ks3 bucket: <mybucket>]
      [--dest your ks3 dest path: <mydemo/test>]
 Quick Start1:[ run -t python -d trainer -n mnist -v 1.0 --home /notebooks/examples/mnist --bucket mybucket --dest mydemo/test ]
 Quick Start2:[ run -t proto -d caffe.mnist -n caffe.mnist -v 1.0 --home /notebooks/examples/caffe --bucket mybucket --dest mydemo/test ]

 参数解释:
 -h : [帮助信息]
 -t : [打包代码的类型，tensorflow 支持 python, caffe 支持 proto]
 -d : [待打包代码的**目录名**，如果是 python 的类型，它也作为打包后模块的 package 名，比如，在这个目录(假设此目录为 trainer)下用户自己创建了自定义 python 模块 a，此时要引用这个模块: import trainer.a ]
 -n : [打包后的 tar.gz file 的文件名]
 -v : [设置打包的版本]
 --home: [设置该项目的所属的根目录，如示例代码的 mnist ,必须是绝对路径]
 --bucket:[对应 ks3 bucket]
 --dest: [对应 ks3 bucket 上具体的目录]
```

- **查看打包后的代码**

根据在打包命令中设置的输出地址，在金山云对象存储中可以找到刚刚打包后的代码。

KDL 虽然在模型开发界面提供了 ks3 的挂载，考虑到用户使用的方便，保留挂载脚本供用户使用

```
root@de-3574-8936-021705-3634728127-79ppp:/# ./run_ks3_bucket_mount.sh
Usage: run_ks3_bucket_mount.sh [argument] [argument_value]...
Arguments:
    -h,--help       display help info and exit
    -o,--operation      <mount|umount>
    -b,--bucket_name    ks3_bucket_name
    -l,--local_dir      local disk dir,will be used by mount and umount
    -p,--mount_params   s3fs params,split by comma
Example:
    mount:
        /run_ks3_bucket_mount.sh -o mount -b your_bucket_name -l /mnt/ks3_buckets/private/your_local_dir -p sigv2,nocopyapi,dbglevel=warn,allow_other,mp_umask=000
    umount:
        /run_ks3_bucket_mount.sh -o umount -l /mnt/ks3_buckets/private/your_local_dir
参数解释:
-h : [帮助信息]
-o : [挂载/卸载]
-b : [ks3 bucket]
-l : [本地挂载目录]
-p : [s3fs 的优化参数，多个参数使用逗号隔开]
```

模型训练

1、本地编写训练代码

编写训练任务代码，代码规范与社区版TensorFlow一致并符合Python模块标准，不依赖额外的KDL API。我们创建第一个示例linear：

- **创建模块**

```
mkdir trainer
touch trainer/__init__.py
touch trainer/task.py
```

- **编写代码**

```
vim trainer/task.py
```

- **引入依赖**

```
#!/usr/bin/env python

import datetime
import json
import numpy as np
import os
import sys
import tensorflow as tf
from tensorflow.contrib.session_bundle import exporter
from tensorflow.python.saved_model import builder as saved_model_builder
from tensorflow.python.saved_model import signature_constants
from tensorflow.python.saved_model import signature_def_utils
from tensorflow.python.saved_model import tag_constants
from tensorflow.python.saved_model import utils
from tensorflow.python.util import compact
```

- **用户自定义参数**

```
flags = tf.app.flags
flags.DEFINE_integer('max_epochs', 100, 'Number of steps to run trainer.')
flags.DEFINE_string('checkpoint_path', './checkpoint/',
'The checkpoint directory')
flags.DEFINE_string("output_path", "./tensorboard/",
"indicates training output")
flags.DEFINE_integer('checkpoint_period', 100,
'Number of epochs to save checkpoint.')
flags.DEFINE_integer('model_version', 1, 'Version number of the model.')
flags.DEFINE_string('model_path', './model/', 'The model directory')
flags.DEFINE_float('learning_rate', 0.01, 'Initial learning rate.')
flags.DEFINE_string("optimizer", "sgd", "Optimizer to train")
FLAGS = flags.FLAGS
```

我们定义的参数将会在KDL控制台创建训练任务时传入，并应用到我们的训练任务代码中。如checkpointpath，指定了我们的训练任务在运行过程中checkpoint时使用的文件目录；outputpath参数用以输出summary信息；model_path以及model_version指定了训练完成后输出模型数据的文件目录及模型版本。我们的训练任务在kdl中作为标准Python模块被引入与执行，用户自定义参数在执行时被设置：

```
...
os.chdir(module_dir)
subprocess.call("python ./setup.py install", shell=True)
...
runpy.run_module(module_name, run_name="__main__")
...
```

当然，也可以在本地执行：

```
python -m trainer.task
```

- **输出Summary信息**

我们可以通过TensorFlow标准方式将summary信息输出至本地磁盘或者ks3文件系统：

```
tf.summary.scalar('loss', loss)
tf.summary.scalar('training/hptuning/metric', loss)
summary_op = tf.summary.merge_all()
init_op = tf.global_variables_initializer()
with tf.Session() as sess:
sess.run(init_op)
print("Save tensorboard files into: {}".format(FLAGS.output_path))
writer = tf.summary.FileWriter(FLAGS.output_path, sess.graph)
summary_value, loss_value, step = sess.run(
[summary_op, loss, global_step],
feed_dict={X: x,Y: y})
writer.add_summary(summary_value, step)
```

- **Checkpoint**

长时间运行的训练任务需要有失败处理策略，可以使用TensorFlow提供的checkpoint功能在合适的时机对作业进行checkpoint，以确保重新启动作业时可以继续运行而不是重新运行，KDL支持用户将checkpoint数据写入ks3文件系统：

```
saver = tf.train.Saver()
with tf.Session() as sess:
sess.run(init_op)
...
saver.save(sess, FLAGS.checkpoint_path + '/model.ckpt', global_step=i)
```

- **Export Model**

训练作业运行完成后我们需要将模型数据导出至ks3文件系统，并启动模型服务队外提供模型预测。具体请参考“模型服务”具体章节。

- **linear及mnist示例完整代码**

  https://ks3-cn-beijing.ksyun.com/ai-train-demo/tf-1.0/linear/trainer-1.0.tar.gz https://ks3-cn-beijing.ksyun.com/ai-train-demo/tf-1.0/mnist/mnist-1.0.tar.gz

2、打包训练任务代码

目前KDL只支持tar.gz格式的代码包，我们可以用setuptools进行打包：

```
cat << EOF > setup.py
import setuptools
setuptools.setup(name='trainer', version='1.0', packages=['trainer'])
EOF
```

打包：

```
python setup.py sdist --format=gztar
```

dist目录中生成我们的代码包：

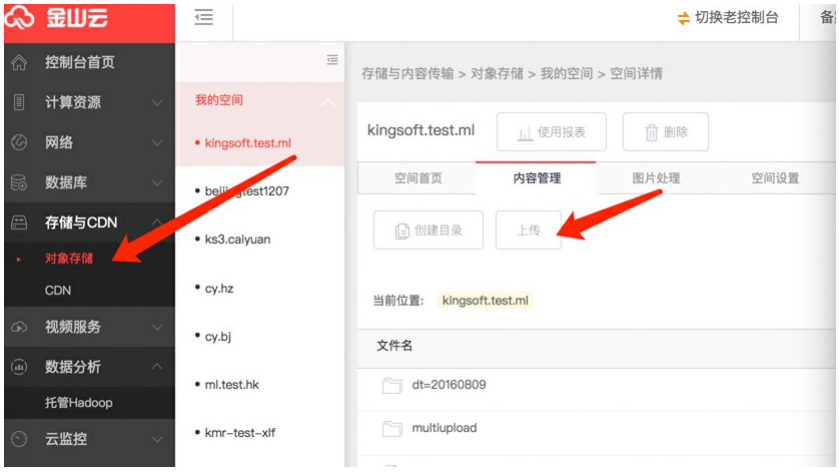
```
├── dist
│   └── trainer-1.0.tar.gz
├── setup.py
├── trainer
│   ├── __init__.py
│   └── task.py
└── trainer.egg-info
    ├── dependency_links.txt
    ├── PKG-INFO
    ├── SOURCES.txt
    └── top_level.txt
```

3、上传代码至ks3

将我们的代码上传至ks3中，如果代码是在“模型开发”中生成，则可以通过打包工具自动导出至ks3。

- **上传代码包**

将我们的代码包上传至ks3中，如图所示：



- **上传数据包**

KDL支持训练作业使用ks3上的训练&测试数据。支持加密数据读取；对压缩文件支持不足，需要用户在作业代码中自行解压。以mnist示例的训练&测试数据为例：

（1）控制台上传数据，上传至某个目录并将访问控制设置为公开；

（2）使用ks3 python sdk上传数据，安装ks3 python sdk，请参考KS3 Python SDK文档https://docs.ksyun.com/documents/965。

上传非加密数据：

```
[]# ls input_data/t10k-images-idx3-ubyte.gz t10k-labels-idx1-ubyte.gz train-images-idx3-ubyte.gz train-labels-idx1-ubyte.gz

vim upload_mnist_data.py

from ks3.connection import Connection as Ks3Connection
import base64

ak = "..."
sk = "..."
host = "ks3-cn-beijing.ksyun.com"
bucket_name = "..."

def test_put_mnist_data():
filenames = ["t10k-images-idx3-ubyte.gz","t10k-labels-idx1-ubyte.gz",
"train-images-idx3-ubyte.gz","train-labels-idx1-ubyte.gz"]
obj_prefix = "tf-1.0/mnist/input_data/"
local_file_prefix = "./input_data/"
try:
connection = Ks3Connection(ak,sk,host=host)
bucket = connection.get_bucket(bucket_name)
for filename in filenames:
key = bucket.new_key(obj_prefix+filename)
response = key.set_contents_from_filename(local_file_prefix+filename)
print response.status,response.msg,response.read()
except Exception as e:
print(e)
if __name__ == '__main__':
test_put_mnist_data()

python upload_minist_data.py
```

上传加密数据：

ks3对上传加密数据提供秘钥托管、自定义秘钥两种方式，需要在上传时提供相应header。

秘钥托管：

```
vim upload_mnist_data_with_hosting_encryption.py

from ks3.connection import Connection as Ks3Connection
import base64

ak = "..."
sk = "..."
host = "ks3-cn-beijing.ksyun.com"
bucket_name = "..."
filenames = ["t10k-images-idx3-ubyte.gz","t10k-labels-idx1-ubyte.gz",
"train-images-idx3-ubyte.gz","train-labels-idx1-ubyte.gz"]
obj_prefix = "tf-1.0/mnist/hosting_encryption_input_data/"
local_file_prefix = "./input_data/"

headers = {"x-kss-server-side-encryption":"AES256"}

def test_post_with_hosting_encryption():
try:
connection = Ks3Connection(ak,sk,host=host)
bucket = connection.get_bucket(bucket_name)
for filename in filenames:
key = bucket.new_key(obj_prefix+filename)
response = key.set_contents_from_filename(local_file_prefix+filename,
headers=headers)
print response.status,response.msg,response.read()
except Exception as e:
print e

def test_get_with_header():

if __name__ == '__main__':
test_
```

自定义秘钥：

```
vim upload_mnist_data_with_custom_encryption.py

from ks3.connection import Connection as Ks3Connection
import base64
import hashlib
import os

ak = "your ak"
sk = "your sk"
host = "ks3-cn-beijing.ksyun.com"
bucket_name = "your bucket"
filenames = ["t10k-images-idx3-ubyte.gz","t10k-labels-idx1-ubyte.gz",
"train-images-idx3-ubyte.gz","train-labels-idx1-ubyte.gz"]
raw_key = "12345678901234567890123"

def make_header():
base64_key = base64.b64encode(raw_key)
md5_base64_key = base64.b64encode(hashlib.md5(raw_key).digest())
```

```
headers = {
"x-kss-server-side-encryption-customer-algorithm":"AES256",
"x-kss-server-side-encryption-customer-key":base64_key,
"x-kss-server-side-encryption-customer-key-MD5":md5_base64_key
}
return headers

def test_post_mnist_data_with_custom_encryption():
try:
obj_prefix = "tf-1.0/mnist/custom_encryption_input_data/"
local_file_prefix = "./input_data/"
connection = Ks3Connection(ak,sk,host=host)
bucket = connection.get_bucket(bucket_name)
for filename in filenames:
key = bucket.new_key(obj_prefix+filename)
response = key.set_contents_from_filename(local_file_prefix+filename,
headers=make_header())
print response.status,response.msg,response.read()
except Exception as e:
print e
def test_get_mnist_data_with_custom_encryption():
try:
local_save_path = "./custom_encryption_download/"
if not os.path.exists(local_save_path):
os.makedirs(local_save_path)
remote_base_path = "tf-1.0/mnist/custom_encryption_input_data/"
connection = Ks3Connection(ak,sk,host=host)
bucket = connection.get_bucket(bucket_name)
headers = make_header()
for filename in filenames:
key = bucket.get_key(remote_base_path+filename,headers=headers)
key.get_contents_to_filename(local_save_path+filename,headers=headers)
except Exception as e:
print(e)
if __name__ == '__main__':
test_post_mnist_data_with_custom_encryption()
test_get_mnist_data_with_custom_encryption()
```

4、提交作业

KDL为用户提供了一键示例任务，帮助用户快速创建训练任务，用户也可以根据自己的具体需求自定义任务。

- **创建一键示例任务**

KDL提供了两个一键示例任务，用户可以选择示例任务进行任务创建，无需修改参数即可提交作业。

- **创建训练任务**

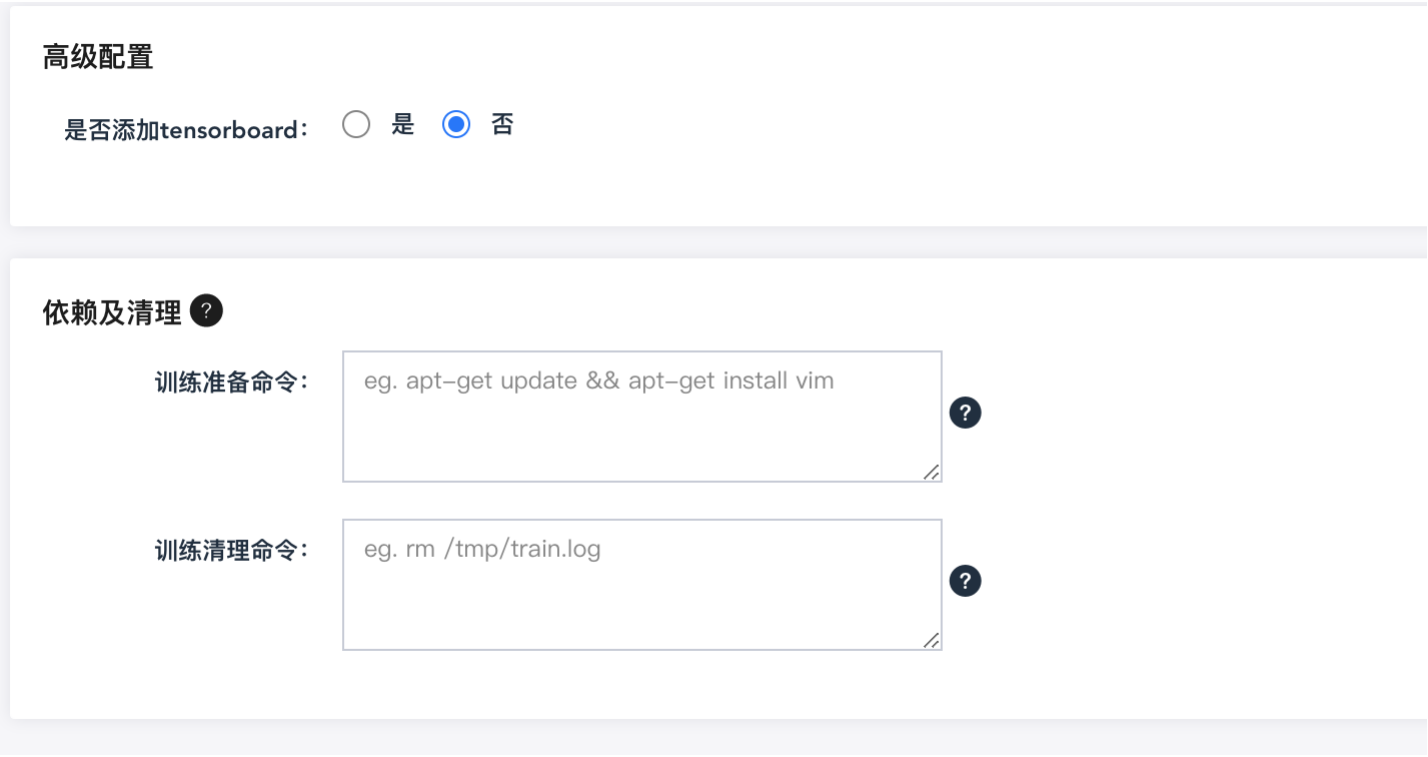
（1）填写任务基本信息

（2）填写训练数据挂载地址

（3）选择资源套餐：KDL提供了多种资源套餐供用户选择

（4）使用训练准备命令与训练清理命令：训练准备命令主要提供训练前安装相关依赖的能力；训练清理命令可以在训练任务结束后进行清理工作。

（5）绑定TensorBoard：创建TensorBoard有两种方式：1、训练任务结束后创建TensorBoard并绑定至训练任务。2、创建训练任务时同时创建TensorBoard并绑定。第2种方式如下图所示：



- **任务信息填写完毕即可提交作业**

5、查看作业

用户可以点击相应的“任务名称”查看作业详情。

- **基本信息**

作业基本信息包含作业的名称、状态、参数等信息:

- **作业资源使用情况**

处于“运行中”的作业，用户可以查看资源利用情况:

- **查看作业日志**

KDL提供作业日志查看功能，用户可以通过日志定位作业异常:

- **绑定TensorBoard**

在训练任务运行的过程中，用户可以通过TensorBoard查看作业的更多信息。KDL中为作业绑定TensorBoard有两种方式:

1、创建训练任务时同时创建TensorBoard并绑定。

2、单独创建TensorBoard,并在训练任务完成后进行绑定。

6、使用命令行工具

KDL提供的命令行工具，可以满足用户在命令行进行训练任务、模型服务、tensorboard的创建及查询功能等功能。注意：由于Kubernetes命名的限制，所有models、TensorBoards资源名称必须符合[a-z0-9]([-a-z0-9]*[a-z0-9]正则表达式，不可以用下划线，不可以超过25个字符。

- **安装命令行工具**

```
wget https://ks3-cn-beijing.ksyun.com/ai-infra/kdl-cli/cloud_ml_common.tar.gz
wget https://ks3-cn-beijing.ksyun.com/ai-infra/kdl-cli/cloud_ml_sdk.tar.gz
tar -zxvf cloud_ml_common.tar.gz
tar -zxvf cloud_ml_sdk.tar.gz
```

需要先安装cloud_ml_common，再安装cloud_ml_sdk:

```
cd cloud_ml_common
sudo pip install -r requirements.txt
sudo python setup.py install

cd ../cloud_ml_sdk
sudo pip install -r requirements.txt
sudo python setup.py install
```

- **初始化环境**

用户需要获取金山云的access key和secret key，以及kdl的endpoint:

```
$cloudml init
Please input access key: your_access_key
Please input secret key (will not be echoed):
Please input cloudml endpoint[default: https://ai-beta:8000]: http://ai.beta.ksyun.com

Test access with supplied credentials? [y/N]: N

Save settings? [y/N]: y
Successfully initialize config file in path: /home/user/.config/ksc/config
```

- **cloudml命令**

可以通过cloudml -h查看支持的命令:

命令名	描述	示例
init	初始化环境变量	cloudml init
org_id	查询当前用户的组织结构id	cloudml org_id
flavors	查询kdl支持的资源套餐	cloudml flavors
jobs	创建、查询模型训练任务	cloudml jobs
models	创建、查询模型服务	cloudml models

tensorboard 创建、查询tensorboard cloudml tensorboard
quoto 查询当前用户的quota信息 cloudml quoto

- **训练任务相关命令**

训练任务相关命令可以通过cloudml jobs -h查看。

（1）查看训练任务列表

```
cloudml jobs list
```

支持分页及按状态过滤：

```
cloudml jobs list -cp 1 -pg 5 -st error
```

（2）提交训练任务

通过参数提交：

```
cloudml jobs submit -n cli_linear_test -m trainer.task -u ks3://ai-train-demo/tf-1.0/linear/trainer-1.0.tar.gz -fi cpu.basic
```

通过JSON文件提交：训练任务json描述文件cli_linear_demo.json：

```
{
"raw_job_name": "cli_linear_demo",
"module_name": "trainer.task",
"trainer_uri": "ks3://ai-train-demo/tf-1.0/linear/trainer-1.0.tar.gz",
"job_args": "--max_epochs 100 --model_path ks3://ai-train-demo/tf-1.0/linear/model",
"flavor_id": "cpu.basic"
}
```

提交任务：

```
cloudml jobs submit -f cli_linear_demo.json
```

提交训练任务命令参数说明：

参数名	参数全名	任务名称	数据类型	是否必填	举例
-n	--raw_job_name	任务名称	string	yes	linear_test
-m	--module_name	任务代码的python模块名称	string	yes	trainer.task
-u	--trainer_uri	任务代码包ks3路径	string	yes	ks3://ai-train-demo/tf-1.0/linear/trainer-1.0.tar.gz
-fi	--flavor_id	资源套餐id	string	yes	cpu.basic
-f	--filename	任务描述文件路径	string	no	../linear_test.json
-a	--job_args	任务参数	string	no	--max_epochs 100 --optimizer sgd
-pc	--prepare_command	训练准备命令，训练任务运行前执行	string	no	pip isntall six && apt-get install libssl-dev
-fc	--finish_command	训练清理命令，训练任务运行后执行	string	no	rm run.log
-ct	--creat_tensorboard	是否为当前任务创建tensorboard	string	no	true或false
-tn	--tensorboard_name	tensorboard名称	string	no	linear_tensorboard
-ld	--logdir	tensorboard目录	string	no	ks3://ai-train-demo/tf-1.0/linear/tensorboard/
-ek	--ks3_encryption_key	ks3上的加密训练数据的加密秘钥,必须是长度为16/24/32的字符串	string	no	12345678901234562

（3）查看训练任务详细信息

```
cloudml jobs describe 1495610454570
```

（4）删除训练任务

```
cloudml jobs describe 1495610454570
```

任务被删除后状态变为terminated。

（5）查看训练任务日志

```
cloudml jobs logs 1495611270639
```

（6）查看训练任务使用情况

```
cloudml jobs metrics 1495611270639
```

（7）查看超参调优任务结果对比

```
cloudml jobs hp 1495594907257-hp-2
```

非超参数调优任务不能使用该命令。

（8）为训练任务绑定tensorboard

将tensorboard绑定至训练任务：

```
cloudml jobs bind_ts 1495611270639 linear-tensorboard
```

- **模型服务相关命令**
- **TensorBoard相关命令**

7、TensorBoard

为了更方便对 TensorFlow 程序的理解、调试与优化，用户可以利用 TensorBoard 可视化工具，展现TensorFlow 图像，绘制图像生成的定量指标图以及附加数据。

- **创建TensorBoard**

创建TensorBoard有两种方式：

（1）创建训练任务时同时创建TensorBoard。具体请参考"模型训练->提交作业"。

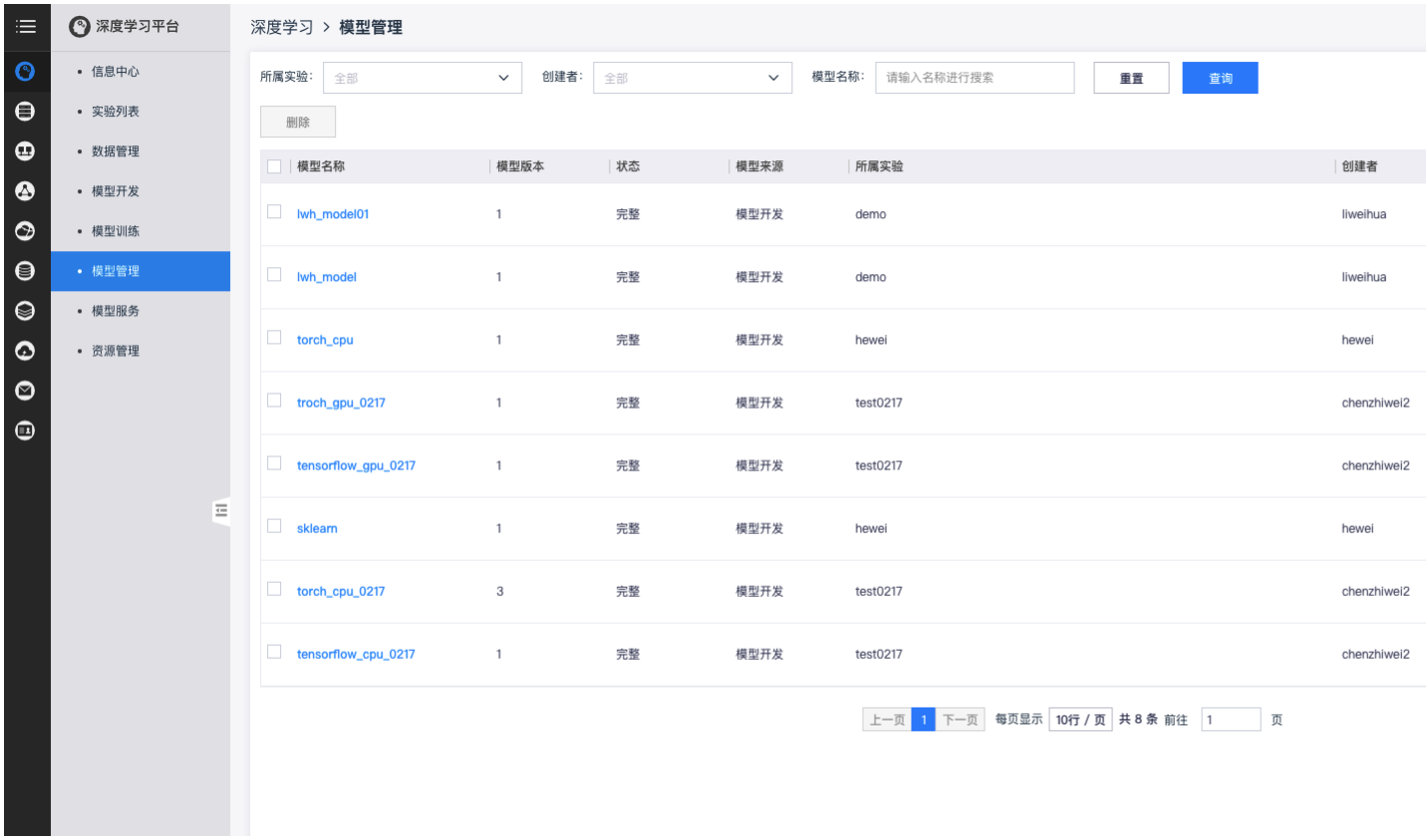
（2）在KDL控制台创建TensorBoard,如下图所示：

- **使用TensorBoard**

KDL为用户托管TensorBoard，只需要访问KDL提供的地址即可访问TensorBoard,如下图所示：

模型管理

模型训练完成后，模型将自动更新到模型管理列表中。



- **模型下载** 模型训练完成后，可点击【模型下载】进行模型下载，默认打包为zip包。
- **模型服务发布** 点击【发布为模型服务】可将模型发布为服务，提供标准的restful接口服务。
- **模型完整性校验** 模型管理提供完整性校验，会校验模型中是否存在模型启动文件，若无启动文件，则会显示模型不完整,完整文件如下：

```
(base) jone@bzd24924-xx tensorflow-cpu % tree
.
├── Model.py
├── mnist_gpu.ipynb
├── models
│   ├── saved_model.pb
│   └── variables
│       ├── variables.data-00000-of-00001
│       └── variables.index
├── requirements.txt
└── setup.sh
```

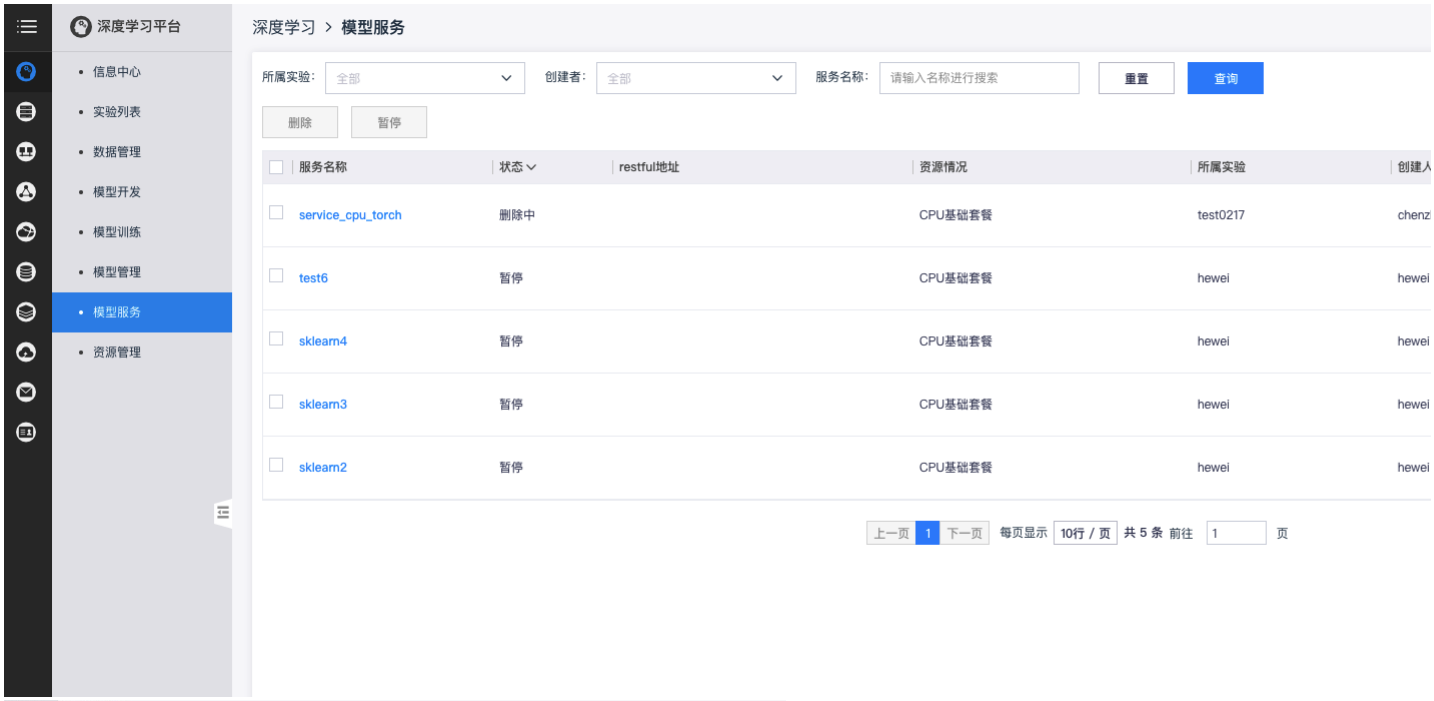
需要包含如下信息：

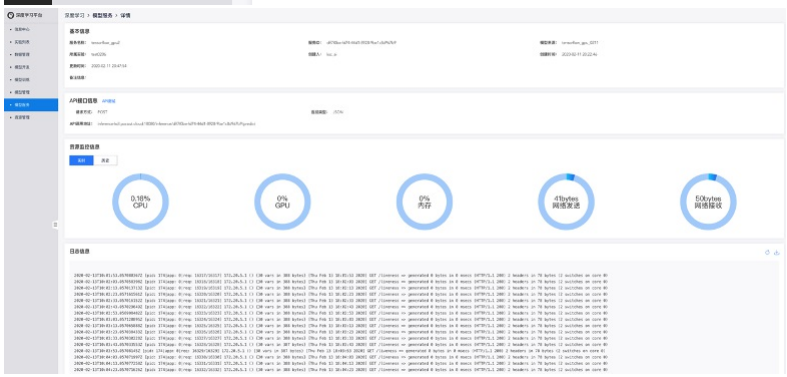
模型启动脚本 > model.py 模型依赖文件 > requirements.txt 环境执行脚本 > setup.sh

模型服务

用户点击模型服务，可查看模型服务列表。

KDL为用户提供了模型导出的功能，支持一键部署、按需部署，支持restful接口访问。





将训练好的模型通过控制台创建模型服务，实现模型部署，提供服务。

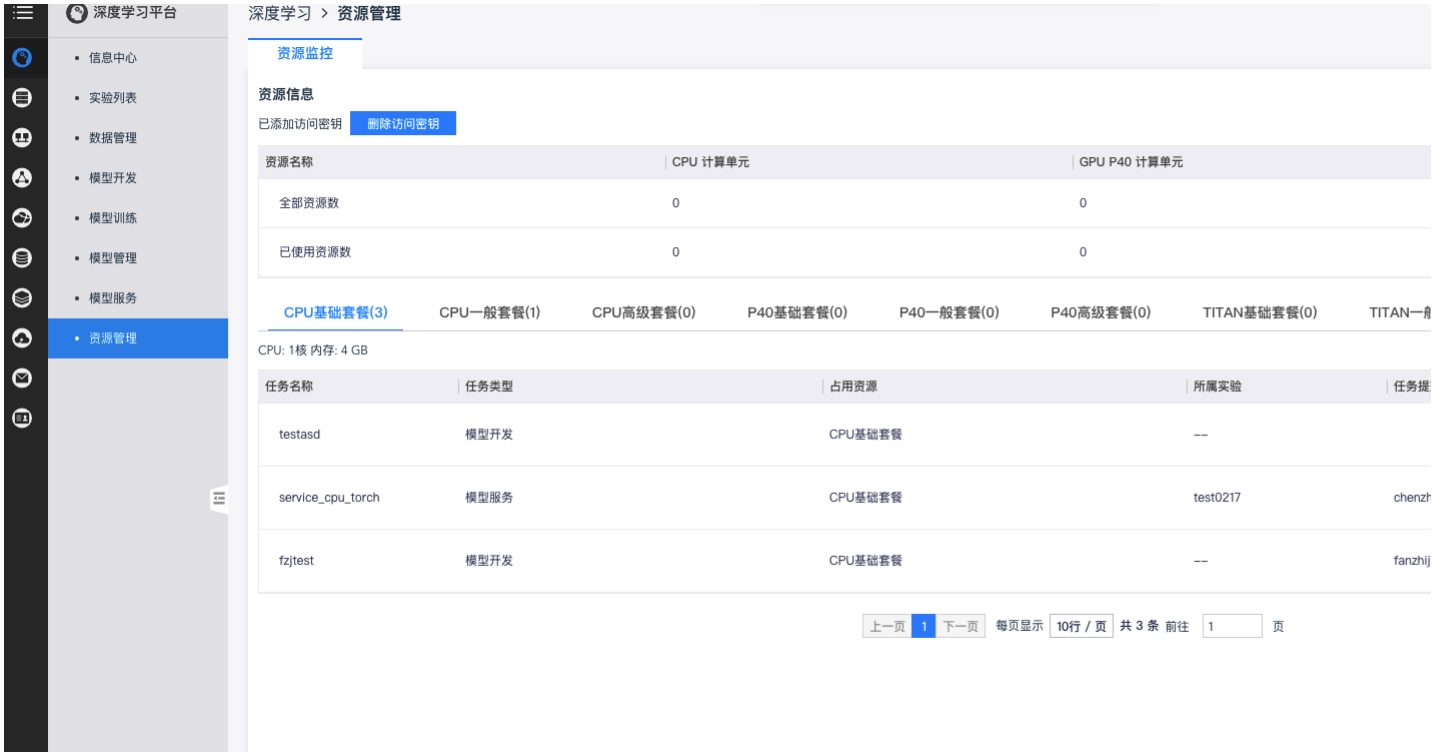
创建模型服务

- 在KDL控制台，点击“模型列表”—>“发布为模型服务”
- 填写服务名称
- 选择服务所占用的资源及副本数量

资源概览

1、用户资源概览

通过资源概览,用户可以看到资源使用情况和作业运行情况



2、资源变更

通过资源变更,用户可以方便的购买需要的资源,并且立即生效

3、AK、SK绑定

KDL依赖KS3进行存储，用户需要将AK、SK 手动填写到系统中，已保证系统正常运行。