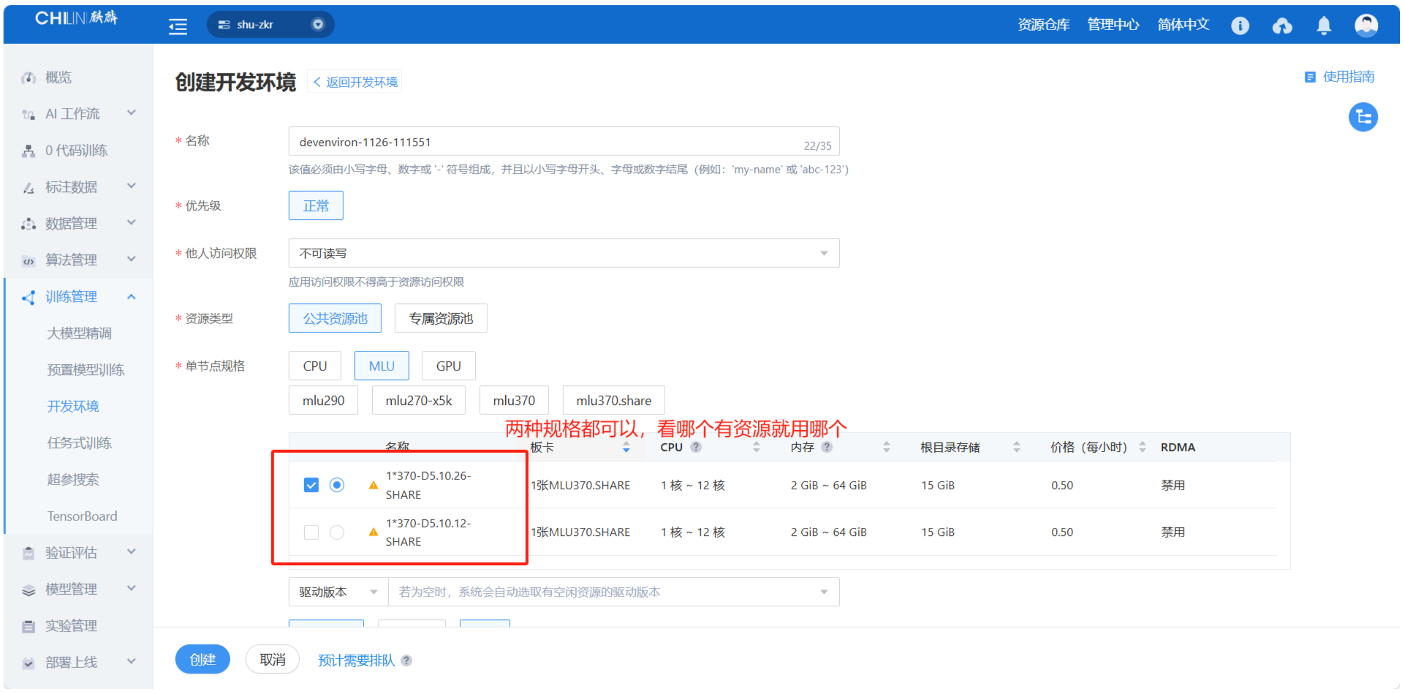
指导书中有关数据集、算法卷、模型卷可以不创建（安排的实验作业用不到），在**训练管理**中选择**开发环境**



创建开发环境 < 返回开发环境

* 名称: devenviron-1126-111551 (22/35)
该值必须由小写字母、数字或 '-' 符号组成，并且以小写字母开头、字母或数字结尾（例如：'my-name' 或 'abc-123'）

* 优先级: [✓ 正常]

* 他人访问权限: 不可读写
应用访问权限不得高于资源访问权限

* 资源类型: [✓ 公共资源池] 专属资源池

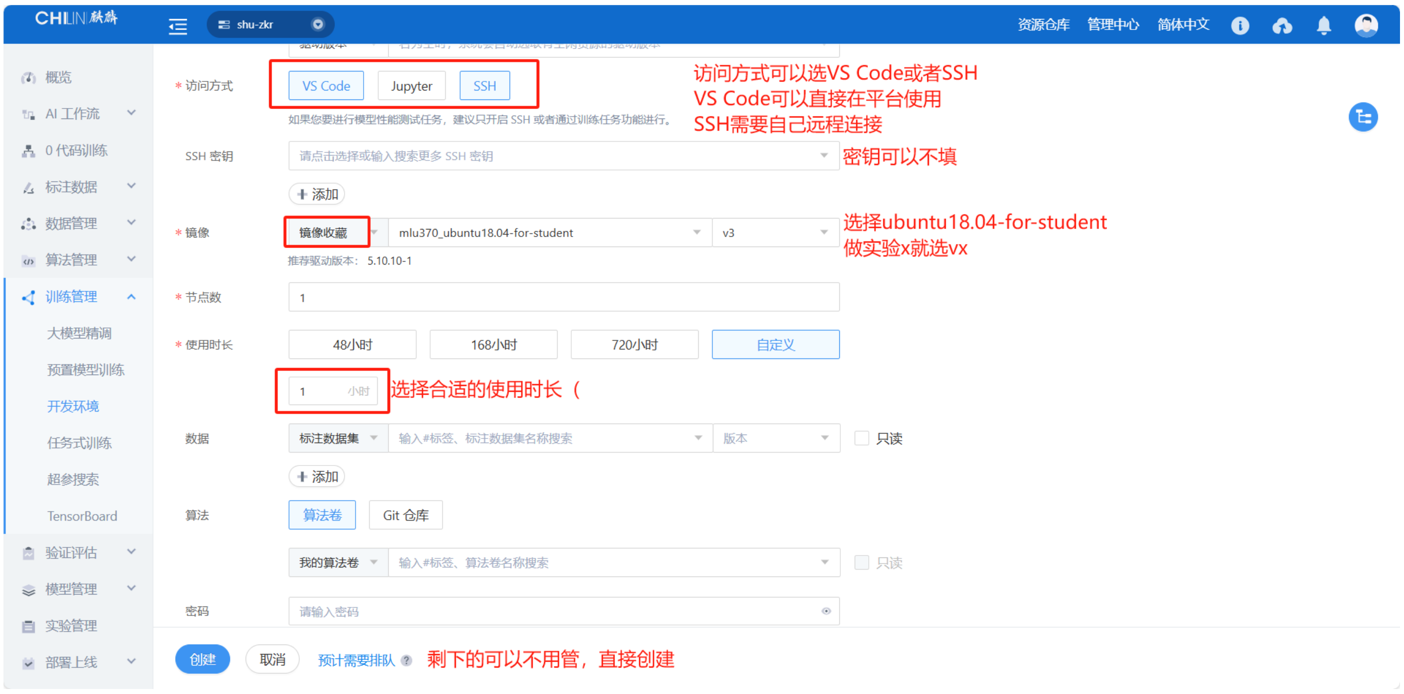
* 单节点规格: CPU [✓ MLU] GPU
mlu290 mlu270-x5k mlu370 mlu370.share

两种规格都可以，看哪个有资源就用哪个

名称	板卡	CPU	内存	根目录存储	价格（每小时）	RDMA
✓ ● 1*370-D5.10.26-SHARE	1张MLU370.SHARE	1 核 ~ 12 核	2 GiB ~ 64 GiB	15 GiB	0.50	禁用
1*370-D5.10.12-SHARE	1张MLU370.SHARE	1 核 ~ 12 核	2 GiB ~ 64 GiB	15 GiB	0.50	禁用

驱动版本: 若为空时，系统会自动选取有空闲资源的驱动版本

创建 取消 预计需要排队



* 访问方式: [✓ VS Code] Jupyter [✓ SSH]
如果您要进行模型性能测试任务，建议只开启 SSH 或者通过训练任务功能进行。

访问方式可以选VS Code或者SSH
VS Code可以直接在平台使用
SSH需要自己远程连接

SSH 密钥: 请点击选择或输入搜索更多 SSH 密钥
+ 添加

密钥可以不填

* 镜像: 镜像收藏 | mlu370_ubuntu18.04-for-student | v3
推荐驱动版本： 5.10.10-1

选择ubuntu18.04-for-student
做实验x就选vx

* 节点数: 1

* 使用时长: 48小时 168小时 720小时 [✓ 自定义]
1 小时

选择合适的使用时长（

数据: 标注数据集 | 输入#标签、标注数据集名称搜索 | 版本 | 只读
+ 添加

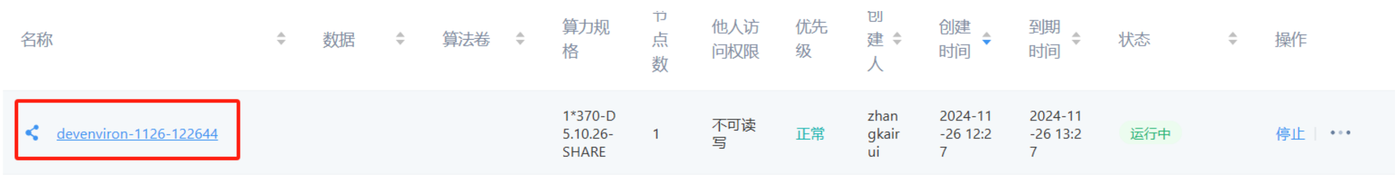
算法: [✓ 算法卷] Git 仓库
我的算法卷 | 输入#标签、算法卷名称搜索 | 只读

密码: 请输入密码

创建 取消 预计需要排队

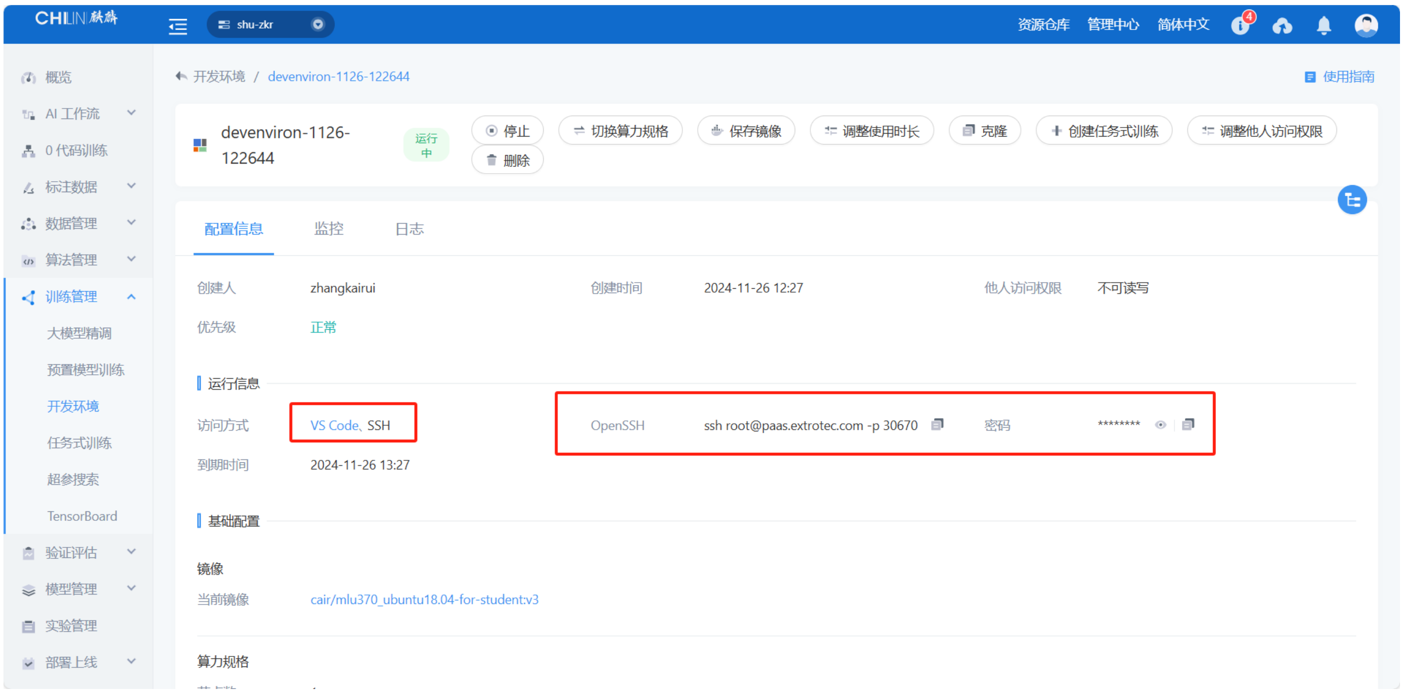
剩下的可以不用管，直接创建

创建好后需要等待一段时间，看到开发容器进入**运行中**状态即可使用（如果一直是排队中的话可以换另一个规格试试）



名称	数据	算法卷	算力规格	节点数	他人访问权限	优先级	创建人	创建时间	到期时间	状态	操作
devenviron-1126-122644			1*370-D5.10.26-SHARE	1	不可读写	正常	zhangkairui	2024-11-26 12:27	2024-11-26 13:27	运行中	停止 ···

点击容器名称查看详情，访问方式里有两种选择：VS Code 和 SSH



开发环境 / devenviron-1126-122644

devenviron-1126-122644 运行中

创建人	zhangkairui	创建时间	2024-11-26 12:27	他人访问权限	不可读写
优先级	正常				

运行信息

访问方式	VS Code、SSH	OpenSSH	ssh root@paas.extrotec.com -p 30670	密码	********
到期时间	2024-11-26 13:27				

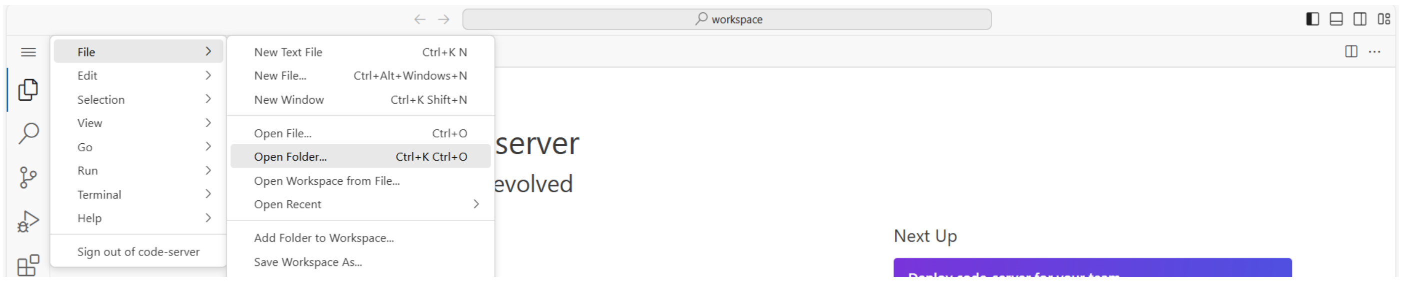
基础配置

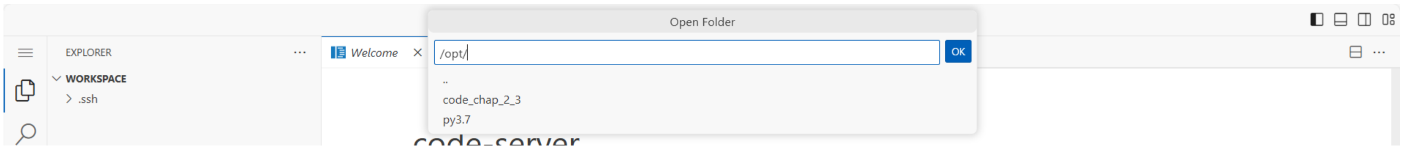
镜像

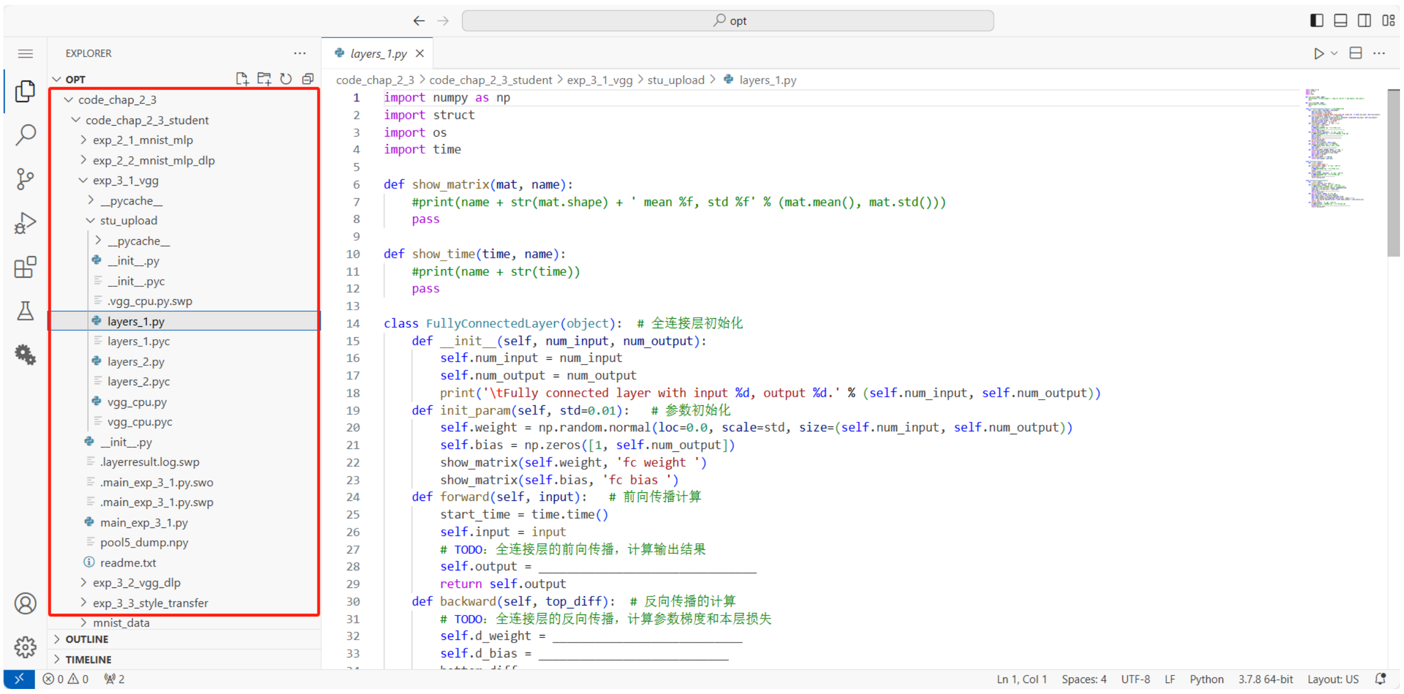
当前镜像 cair/mlu370_ubuntu18.04-for-student:v3

算力规格

可以点击 VS code 进入 code-server，默认路径是/workspace，需要切换到/opt







code_chap_2_3 > code_chap_2_3_student > exp_3_1_vgg > stu_upload > layers_1.py

```python
import numpy as np
import struct
import os
import time

def show_matrix(mat, name):
    #print(name + str(mat.shape) + ' mean %f, std %f' % (mat.mean(), mat.std()))
    pass

def show_time(time, name):
    #print(name + str(time))
    pass

class FullyConnectedLayer(object):  # 全连接层初始化
    def __init__(self, num_input, num_output):
        self.num_input = num_input
        self.num_output = num_output
        print('\tFully connected layer with input %d, output %d.' % (self.num_input, self.num_output))
    def init_param(self, std=0.01):   # 参数初始化
        self.weight = np.random.normal(loc=0.0, scale=std, size=(self.num_input, self.num_output))
        self.bias = np.zeros([1, self.num_output])
        show_matrix(self.weight, 'fc weight ')
        show_matrix(self.bias, 'fc bias ')
    def forward(self, input):   # 前向传播计算
        start_time = time.time()
        self.input = input
        # TODO: 全连接层的前向传播，计算输出结果
        self.output = ______________________________
        return self.output
    def backward(self, top_diff):  # 反向传播的计算
        # TODO: 全连接层的反向传播，计算参数梯度和本层损失
        self.d_weight = __________________________
        self.d_bias = __________________________
```