# 1．定制分析

**（1）功能介绍**

编写python程序调用平台中的数据、运行、结果可视化；

**（2）前端逻辑**

现有在线编程逻辑不变：点击之后，每个用户自己创建的Project列表和描述->点击每个project后，进入code editor界面；

Code editor界面中的页面界面，打开有默认代码（如山东省的NDVI数据计算：先输入检索条件、划格网、后台分布式计算，计算过程log、计算结果可视化）；

Code editor目前为单py文件，改成可以多个py文件的创建和编辑

大的tif的可视化：需要转COG；

Code editor中要有一个封装好的库，叫做ARDService。类似于GEE的ee库，封装好常用的数据检索、数据保存、COG切片等。GEE中有Image、ImageCollection对象。。ARDService库提供Cube对象。ARDService有个init函数，初始化相关的认证、链接等操作。ARDService有个get\_cubes函数，根据输入的时间、空间、载荷名称（可缺省）返回Cube对象的list。Cube对象有get\_rasters函数，返回该Cube内的栅格数据list。Cube对象有get\_rasters函数，返回该Cube内的栅格数据list，注意返回的一个栅格数据并不是tiff，应该是一个raster对象（包含必要的元数据）。Cube对象有get\_features函数，返回矢量相关。ARDService有个DistributeComputing，封装ray相关的调用。默认提供的代码应包含DistributeComputing的使用case。ARDService还需要有个可视化对象，使用titler对minio中的COG进行可视化

示例代码可能如下：

import ARDService

@ARDServices.DistributeComputing.DistributedFunction

def hello\_world\_cal\_NDVI(cube, other prarams):

        gf2\_list = cube.get\_rasters(sensor\_name=GF2)

ndvi\_list = []

for gf2 in gf2\_list

        img = gf2.get\_pixels() #执行真的相交计算使用riotiler，返回np数组

ndvi\_list.apeend(NDVI(img)) # NDVI用户自定义函数

return ndvi\_list

if \_\_name\_\_ == '\_\_main\_\_':

ARDService.init()

       # 获取Cubes 的list

cubes = get\_cubes(spatial\_dimension = geojson/regionID, time\_dimension = timedataspan, geodata\_dimension = All/一个sensor name/多个sensorName)

ndvs = [hello\_world\_cal\_NDVI (cube) for cube in cubes]

results = ARDServices.DistributeComputing.get(ndvs) #获取所有cube的结果

ARDService.save(results) #保存到用户自己的minio空间，输入可以是np数组

**（3）后端逻辑**

用户容器Image制作：以python-slim镜像为基础，采用dockerfile的形式打包ARDService的依赖，注意所有执行压缩为一行run；

用户生成的中间数据和结果数据必须上传到用户的MINIO空间；

用户生成tiff时，转成COG，可以默认在ARDService.save中写死，或者给个option。

Code editor界面中的页面界面，打开有默认代码（如山东省的NDVI数据计算：先输入检索条件、划格网、后台分布式计算，计算过程log、计算结果可视化）；

用户资源限制：ray可调用cpu、内存; minIO最大存储空间

用户写的代码保存在哪 ？容器中、MINIO？

用户的一个project的container：运行多久杀死，杀死后下次怎么RUN？

用户可视化时，通过tiller调用minio用户空间中的cog

# 2．工具发布

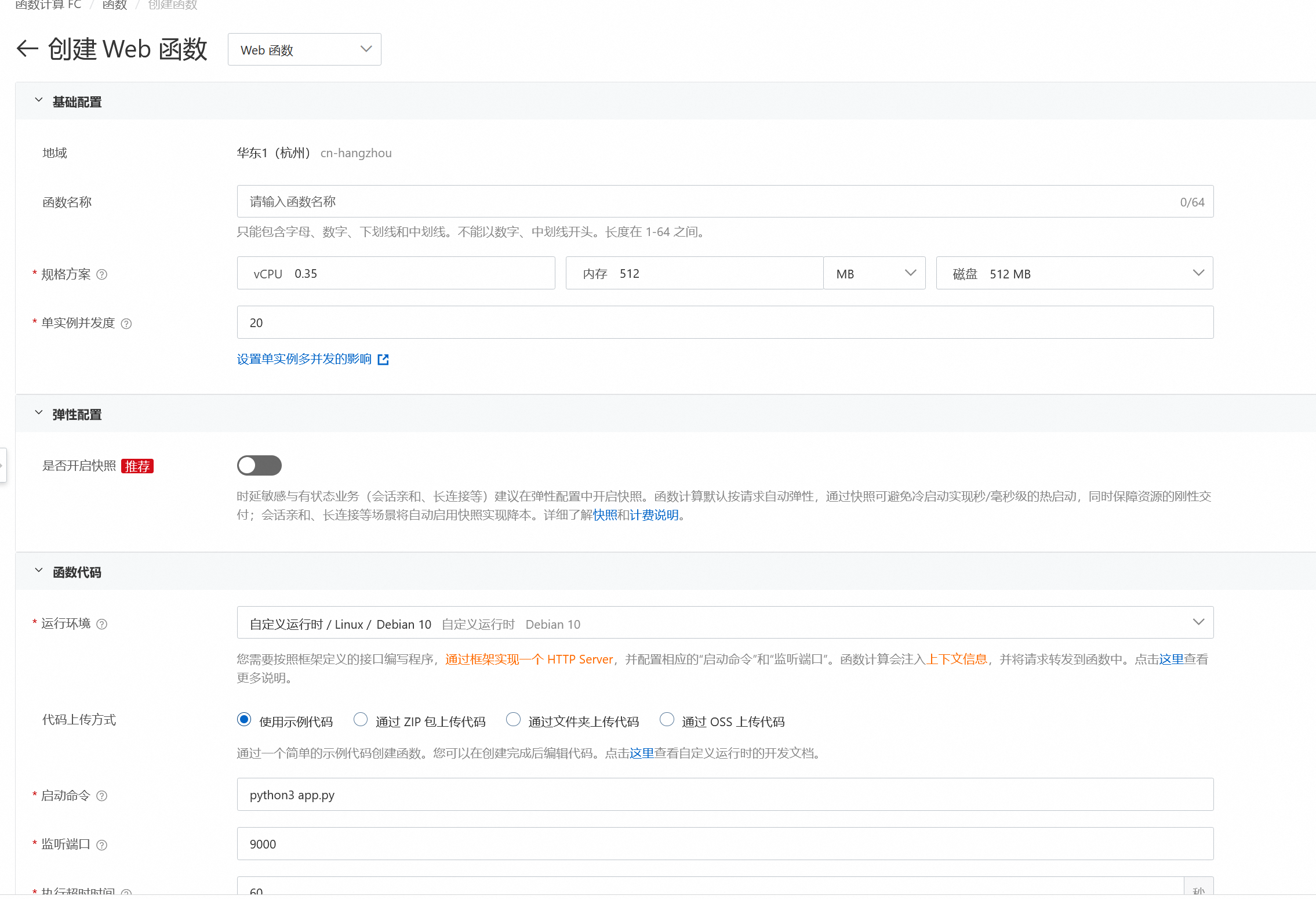
**（1）功能介绍**

编写python程序、直接发布为API服务

**（2）前端逻辑**

整体与定制分析差不多：点击之后，每个用户自己发布的工具API列表和描述->点击每个工具后，进入code editor界面->python模板->发布；

创建工具按钮：弹出对话框，参考如下



创建后，打开code editor界面，弹出默认python代码模板，参考如下：

API的输入输出限定死：输入必须为一个Cube的编号，输出必须为Minio的路径。此外一个API可以有很多其他自定义参数

from flask import Flask

from flask import request

REQUEST\_ID\_HEADER = 'x-fc-request-id'

app = Flask(\_\_name\_\_)

@app.route('/', defaults={'path': ''})

@app.route('/<path:path>', methods=['GET', 'POST', 'PUT', 'DELETE'])

def hello\_world(path):

        rid = request.headers.get(REQUEST\_ID\_HEADER)

        print("FC Invoke Start RequestId: " + rid)

        data = request.stream.read()

        print("Path: " + path)

        print("Data: " + str(data))

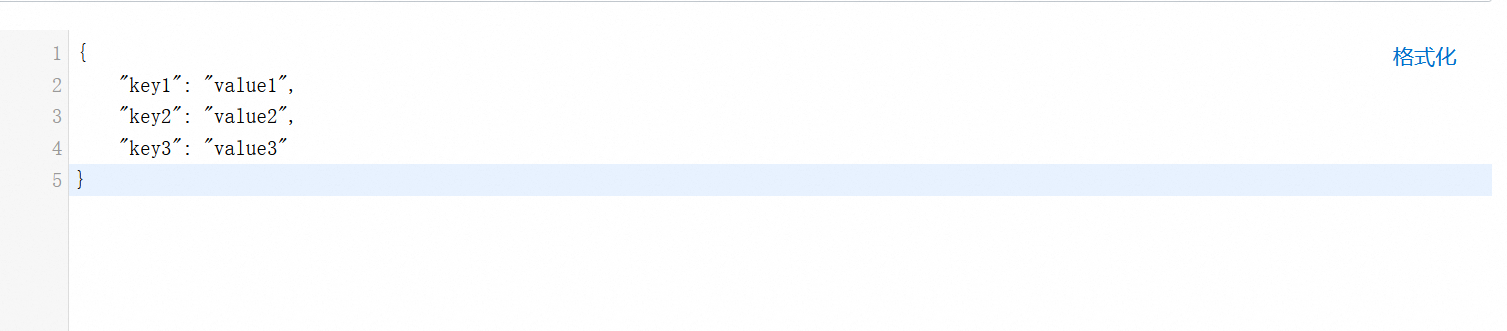
        print("FC Invoke End RequestId: " + rid)

        return "Hello, World!"

if \_\_name\_\_ == '\_\_main\_\_':

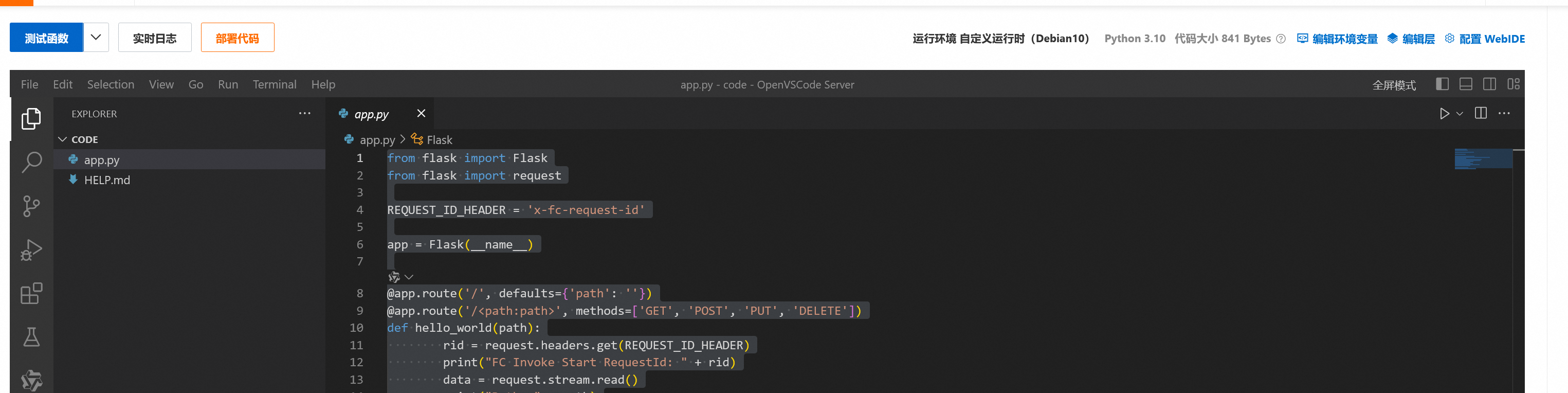
        app.run(host='0.0.0.0',port=9000)

需要有部署工具按钮，让用户输入这个API的参数和返回的config



需要有测试工具按钮，用于验证这个工具是否创建成功，根据config发送请求，查看状态是否成功

需要有编辑环境变量按钮：让用户可以安装python依赖



**（3）后端逻辑**

用户容器Image制作：以python-slim镜像为基础，采用dockerfile的形式打包ARDService、flask、request的依赖，注意所有执行压缩为一行run；

工具部署完成后，需要在资源中心里面添加关于该工具的介绍

数据中心-分析展示里面也能够调用该工具，分析展示页面调用该工具根据他的API的config文件

# 3．用户中心界面

参考OenGMS的门户网站

关于用户的信息、用户上传/产生的MINIO数据、用户的定制分析project、用户发布的工具的列表