基于阿里云ECS与ACR的python微服务镜像构建、部署与接口访问实验

实验目的

- 熟练掌握使用Dockerfile构建容器镜像的基本方法.
- 熟练掌握阿里云云服务器ECS与容器镜像服务ACR的相关操作.
- 理解HTTP协议基本概念,熟练掌握HTTP调试工具的使用.
- 理解网络服务API的基本概念、分类, 重点掌握HTTP RESful API的定义原则、应用模式.
- 理解并掌握基于python Requests 库撰写代码访问HTTP RESful API定义原则、应用模式.

实验要求

本实验以设备管理服务中设备实体的增删改查接口为例, 提供采用python Sanic 撰写的微服务范例(使用 `Pyinstaller` 打包为可执行程序, 不直接提供代码). 要求学生在学习微服务接口定义与相关数据结构的基础上完成以下具体任务:

- 在本地开发环境启动、测试微服务范例
- 在本地开发环境创建Dockerfile, 将微服务范例程序打包成基础镜像, 并上传至阿里云容器仓库.
- SSH登陆阿里云ECS, 从ECS登陆阿里云容器镜像ACR, 拉取承载微服务范例程序的镜像并部署应用.
- 基于ECS公网IP访问微服务范例程序接口, 查阅四个范例接口的API文档, 随后使用python Request 库编码访问微服务范例接口, 体验HTTP RESTful API接口应用模式与能力。

实验环境

- 本地开发环境: Python3、 Python Requests
- 阿里云云服务器ECS(Ubuntu Server): Docker
- 阿里云容器镜像服务(ARC)
- Python微服务范例:exp_files\exp_pyms_api_demo.tar.gz

实验步骤

定义服务接口

学习范例接口

设备管理服务面向设备实体的增删改查接口定义

功能	方法	URL	说明
查找设备	GET	/v1/devices	基于设备编号，或设备设备「序号、类型」组合查询设备
添加设备	POST	/v1/devices	基于设备信息添加设备
更新设备信息	PUT	/v1/devices	基于设备编号定位设备后更新设备信息
删除设备	DELETE	/v1/devices	基于设备编号列表定位设备后删除设备，返回实际删除的设备编号列表

设备信息数据格式定义

设备信息数据格式包含JSON 与CSV 两种形式. JSON形式数据主要用于访问接口时的数据交互, 而CSV 形式数据主要用于持久化存储, 以作为学习使用数据库之前的持久化存储方案.

- 设备信息数据定义JSON

```
{
    "id":"string",
    "name":"string",
    "type":"string",
    "hardware":{
        "model":"string",
        "sn":"string"
    },
    "software":{
        "version":"string",
        "last update":"string"
    },
    "nic":[
      {
        "type":"string",
        "mac":"string",
        "ipv4":"string"
      }
    ],
    "status":"string"
}
```

- 设备信息数据定义CSV

```
<id>,<name>,<type>,<hardware_model>,<hardware_sn>,<software_version>,
<software_last_update>,<nic_type>,<nic_mac>,<nic_ipv4>,<status>
```

- 设备信息数据范例CSV

```
2193806689,EcnlfkHh,adapter,x86 PC,YJsaS,55.88,2023-08-06 20:03:50,eth,7d:0c:28:f9:95:1f,246.27.153.247,,,,offline
9479178747,pilDFeKC,controller,Raspberry Pi 4,XRmiI,92.22,2023-08-06 18:17:59,wifi,09:f2:04:1e:04:4d,231.70.94.238,,,,offline
6713124465,EEjadTUF,controller,x86 PC,PIcPV,77.56,2023-08-06 20:27:35,eth,ba:14:74:57:f3:01,119.194.56.96,wifi,92:d0:b1:7e:3a:dd,236.209.203.20,offline
3116641919,tviSPqIE,controller,Raspberry Pi 4,iQgLB,18.64,2023-08-06 18:10:45,wifi,3c:9b:3f:87:aa:64,31.76.203.52,eth,dd:0e:dc:ad:68:3d,20.101.58.98,offline
5993786245,ORhtJVeN,adapter,Raspberry Pi 4,tKvOj,47.18,2023-08-06 20:01:58,wifi,60:e3:97:b1:82:db,219.151.211.29,,,,offline
```

按需设计接口

本实验系列最终将支持学生在大作业中实现一个**设备管理微服务**，而该服务的北向接口主要涉及两个实体及其关系管理，即「设备」、「分组」以及它们之间的多对多关联关系。在 6.1.1节，已经学习了面向「设备」实体的「增删改查」接口定义，请参照课件所属 HTTP RESTfUl API接口定义一般原则，自行定义面向分组实体，以及「设备与分组间的关联关系」的「增删改查」接口定义。设备管理服务全部北向接口将在大作业要求中提供。

构建实验环境

云服务器以及docker引擎等实验环境已完成，与“ECS_ACR_Container_Management_Experiment” 实验相同。

基于阿里云云容器镜像服务ACR配置镜像仓库

登录阿里云, 访问容器镜像服务ACR产品首页(https://www.aliyun.com/product/acr), 点击[管理控制台] 进入[实例]列表,创建使用[个人实例]. 随后根据页面提示:

- 设置访问密码
- 创建命名空间，命名空间建议设定为「exp_<name>」(因该命名空间为全局命名空间，故而必须唯一)

- 创建镜像仓库，本实验中名称可统一定义为 `exp_pyms_api_demo`，且「代码源」选择本地仓库。

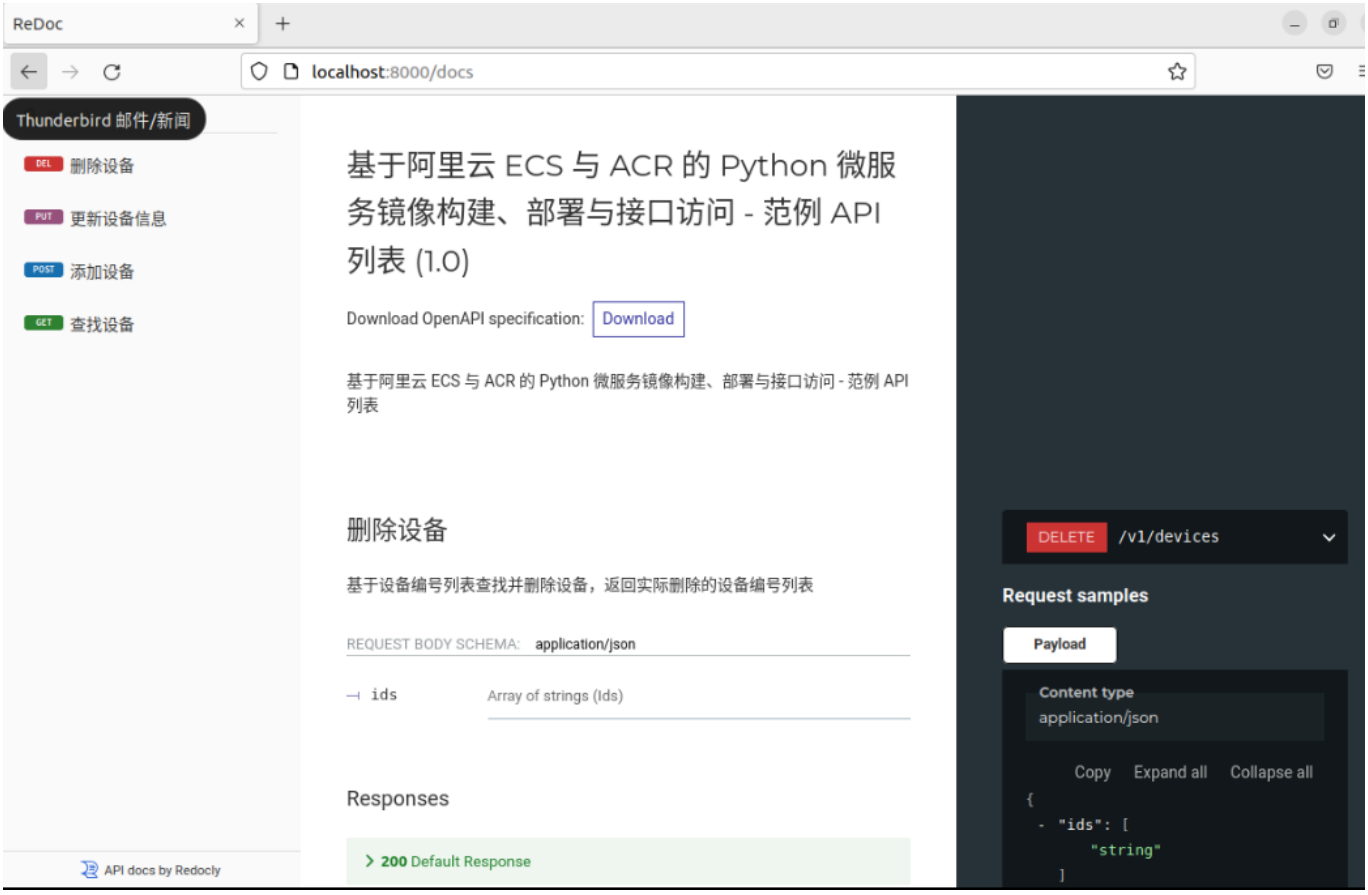
在本地开发环境测试微服务范例程序

在本地开发环境启动本实验提供的微服务范例，体验 HTTP RESTfuLAPI 接口。 本微服务范例在 `8000` 端口监听，输入启动命令:

```
./exp_pyms_api_demo ./device.csv
#./exp_pyms_api_demo<csv file〉，csv_file为存储设备信息的CSV文件路径。若该路径中CSV文件已存在，则从cSV读取设备信息:否则自动创建5条 设备信息，并生成csv文件
```

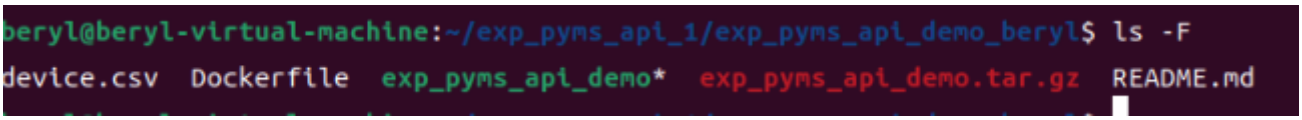
```
[2024-11-05 11:40:05 +0800] [4586] [INFO] 文件./device.csv不存在，自动创建如下 ID 的设备：['3483611286', '9425944421', '6835637929', '1459035705', '5068903808']
[2024-11-05 11:40:05 +0800] [4586] [INFO]
                Sanic v23.6.0
              Goin' Fast @ http://0.0.0.0:8000
Build Fast. Run Fast.
    mode: production, single worker
  server: sanic, HTTP/1.1
  python: 3.10.6
platform: Linux-6.8.0-48-generic-x86_64-with-glibc2.35
packages: sanic-routing==23.6.0, sanic-ext==23.6.0

[2024-11-05 11:40:05 +0800] [4586] [WARNING] Sanic is running in PRODUCTION mode. Consider using '--debug' or '--dev' while actively developing your application.
[2024-11-05 11:40:05 +0800] [4586] [INFO] Sanic Extensions:
[2024-11-05 11:40:05 +0800] [4586] [INFO]   > injection [0 dependencies; 0 constants]
[2024-11-05 11:40:05 +0800] [4586] [INFO]   > openapi [http://0.0.0.0:8000/docs]
[2024-11-05 11:40:05 +0800] [4586] [INFO]   > http
[2024-11-05 11:40:05 +0800] [4586] [INFO] Starting worker [4586]
```

此时访问http://localhost:8000/docs



且当前目录将产生device.csv 文件:

```
ls -F
```



测试完毕, 输入CTRL+C 关闭微服务范例程序。

在阿里云ECS以容器的方式部署微服务范例

构建微服务范例镜像(本地)

在本地开发环境创建Dockerfile 文件如下:

```
# 使用Ubuntu22.04作为基础镜像
FROM ubuntu:22.04
LABEL maintainer="beryl<3049736719@qq.com>"

# 指定工作目录为 /app
WORKDIR /app
exp_pyms_api_demo (前)复制到Docker镜像中的exp_pyms_api_demo(后)文件夹内
COPY exp_pyms_api_demo exp_pyms_api_demo

# 暴露服务端口号 8000
EXPOSE 8000

# 指定容器的启动命令，device.csv 将存储在容器的 /var/lib/exp_pyms_api_data/ 目录下面，
因此该目录应持久化
CMD ["/app/exp_pyms_api_demo", "/var/lib/exp_pyms_api_data/device.csv"]
```

创建一个目录exp 将Dockerfile与exp_pyms_api_demo文件复制进去, 确保工作目录中仅有Dockerfile与exp_pyms_api_demo

然后使用docker build命令构建容器镜像,并为镜像设置名为exp_pyms_api_demo:1.0

```
# -t, tag:后跟镜像名称(REPOSITORY:TAG)
# exp_pyms_api_demo:1.0:自定义镜像名称
# .:PATH，执行命令的上下文路径，构造过程中可以引用该上下文中路径中的任何文件
sudo docker build -t exp_pyms_api_demo:1.0 .
```

提示没有docker指令,要求先下载docker,下载docker指令如下:

```
# 更新包列表并安装依赖
sudo apt-get update
sudo apt-get install ca-certificates curl

# 创建 Docker GPG 密钥目录并下载阿里云的 Docker GPG 密钥
```

```
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# 添加阿里云 Docker 源到 Apt 源列表
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://mirrors.aliyun.com/docker-ce/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# 更新包列表
sudo apt-get update

# 安装 Docker 引擎
sudo apt-get install docker-ce docker-ce-cli containerd.io

# 检验Docker版本
docker --version
```

如果显示了 Docker 的版本信息，说明安装成功。

配置国内镜像

用下面的指令打开配置文件

```
sudo nano /etc/docker/daemon.json
```

输入以下内容:

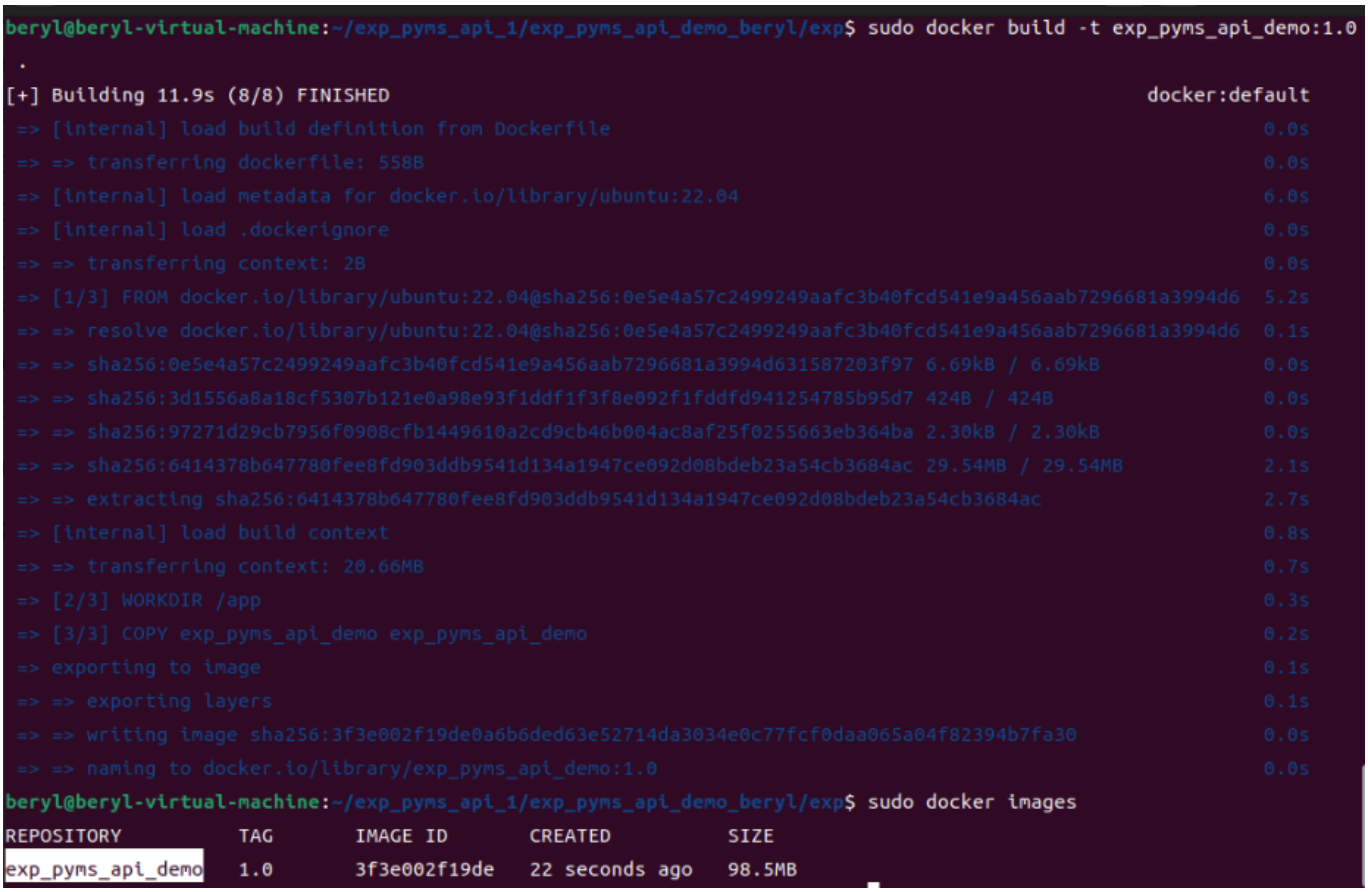
```
{
  "registry-mirrors": [
    "https://docker.1panel.live",
    "https://hub-mirror.c.163.com",
    "https://mirror.ccs.tencentyun.com",
    "https://registry.docker-cn.com"
  ]
}
```

```
sudo systemctl daemon-reload
sudo systemctl restart docker # 重启服务
```

然后，继续执行下面的命令

```
# -t, tag:后跟镜像名称(REPOSITORY:TAG)
# exp_pyms_api_demo:1.0:自定义镜像名称
# .:PATH, 执行命令的上下文路径, 构造过程中可以引用该上下文中路径中的任何文件
sudo docker build -t exp_pyms_api_demo:1.0 .
sudo docker images # 查看
```

终端执行界面如下，可知执行成功。



上传镜像至镜像仓库(本地)

在本地开发环境登录阿里云镜像服务ACR

```
docker login --username=<你的阿里云名字> registry.cn-chengdu.aliyuncs.com
```

随后为本地 `exp_pyms_api_demo:1.0` 镜像设置带有阿里云镜像仓库标记的标签

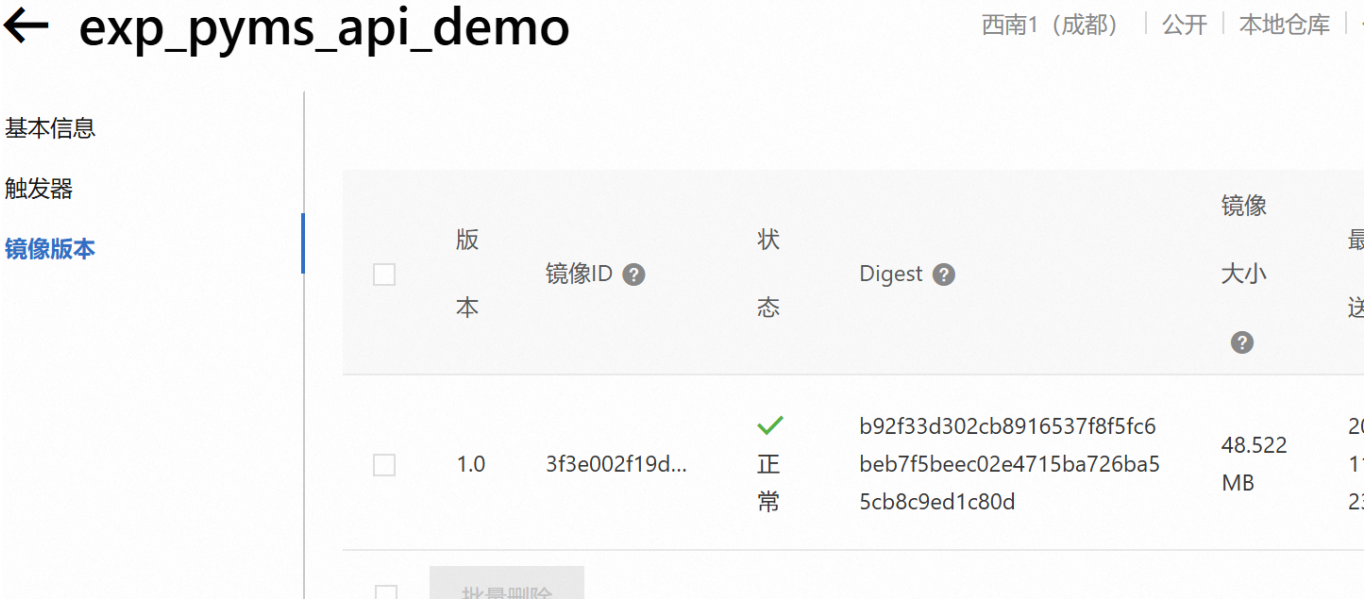
```
docker tag exp_pyms_api_demo:1.0 registry.cn-
chengdu.aliyuncs.com/exp_pyms_api_demo/exp_pyms_api_demo:1.0
```

随后使用`docker push` 命令将本地镜像上传至阿里云镜像仓库

```
beryl@beryl-virtual-machine:~$ sudo docker tag exp_pyms_api_demo:1.0 crpi-y10v9g9mwj5yaosh.cn-chengdu.personal.cr.aliyuncs.com/myexp_1/exp_pyms_api_demo_0913027:1.0
```

```
beryl@beryl-virtual-machine:~$ sudo docker push crpi-y10v9g9mwj5yaosh.cn-chengdu.personal.cr.aliyuncs.com/myexp_1/exp_pyms_api_demo_0913027:1.0
The push refers to repository [crpi-y10v9g9mwj5yaosh.cn-chengdu.personal.cr.aliyuncs.com/myexp_1/exp_pyms_api_demo_0913027]
56ca2b853dd6: Pushed
4922219550f4: Pushed
2573e0d81582: Pushed
1.0: digest: sha256:b92f33d302cb8916537f8f5fc6beb7f5beec02e4715ba726ba55cb8c9ed1c80d size: 947
```

访问阿里云 ACR 控制台页面，可以看见刚推送上去的镜像 exp pyms_api demo:1.0，如下图所示：



拉取镜像并部署服务(ECS操作)

以SSH方式登录阿里云ECS, 从ECS登录阿里云ACR拉取镜像

```
# 登录指令
docker login --username=<你的阿里云名字> registry.cn-chengdu.aliyuncs.com

# 拉取镜像
docker pull registry.cn-chengdu.aliyuncs.com/exp_ecs_and_acr/beryl_exp_pyms_data:[镜像版本号]
```

在 ECS 工作目录(自行指定)创建一持久化目录 `beryl_exp_pyms_data`，用于在 ECS 存储容器内服务生成的业务数据文档，即设备信息数据，随后运行微服务范例程序容器镜像。

```
mkdir beryl_exp_pyms_data
ls
```

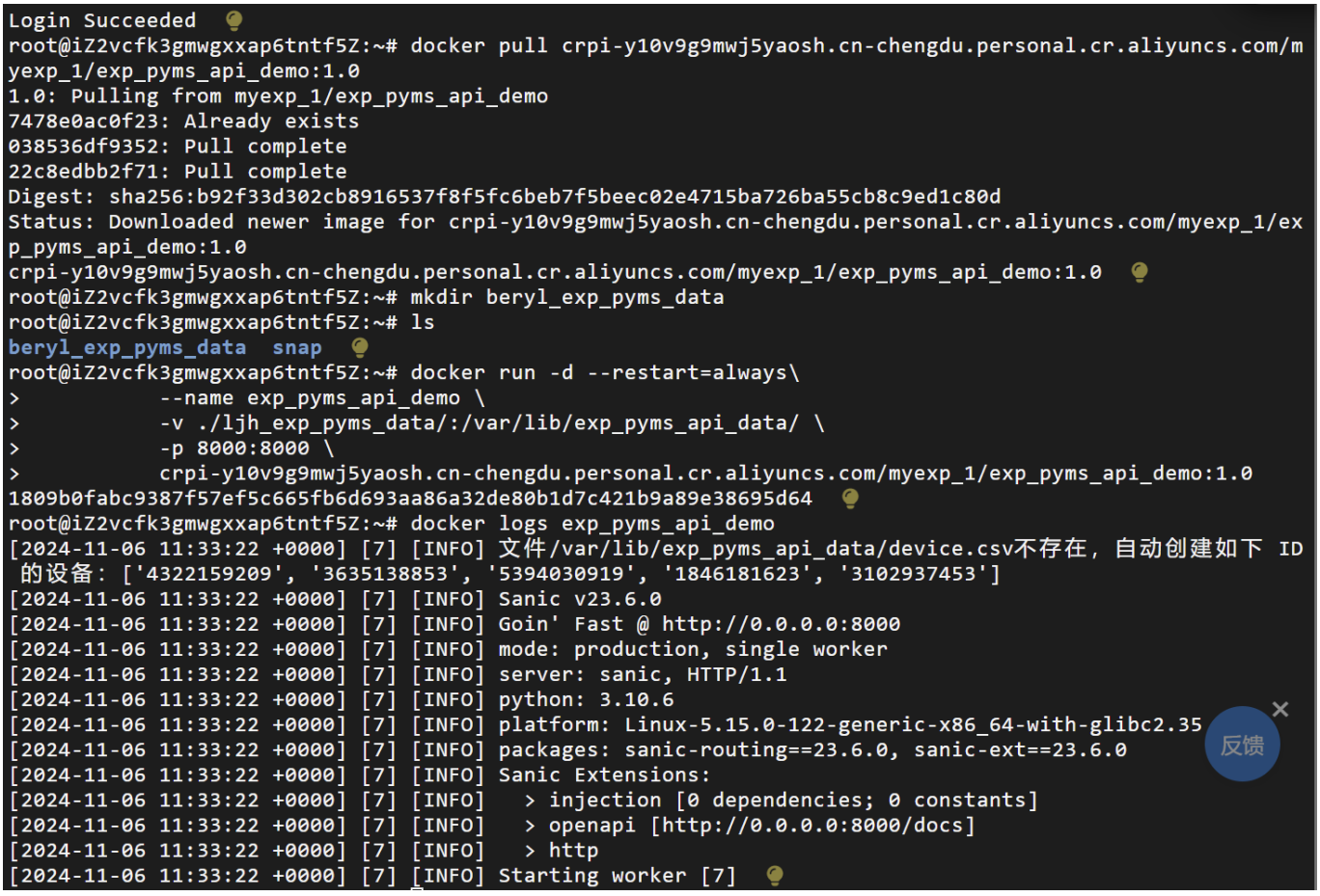
运行一下指令, 进行`docker run`

```
docker run -d --restart=always\
           --name exp_pyms_api_demo \
           -v ./beryl_exp_pyms_data/:/var/lib/exp_pyms_api_data/ \
           -p 8000:8000 \
           registry.cn-chengdu.aliyuncs.com/exp_ecs_and_acr/beryl_exp_pyms_data:
[镜像版本号]
```

查看容器运行日志, 可以确认服务已经成功启动

```
docker logs exp_pyms_api_demo
```

具体操作如图所示:

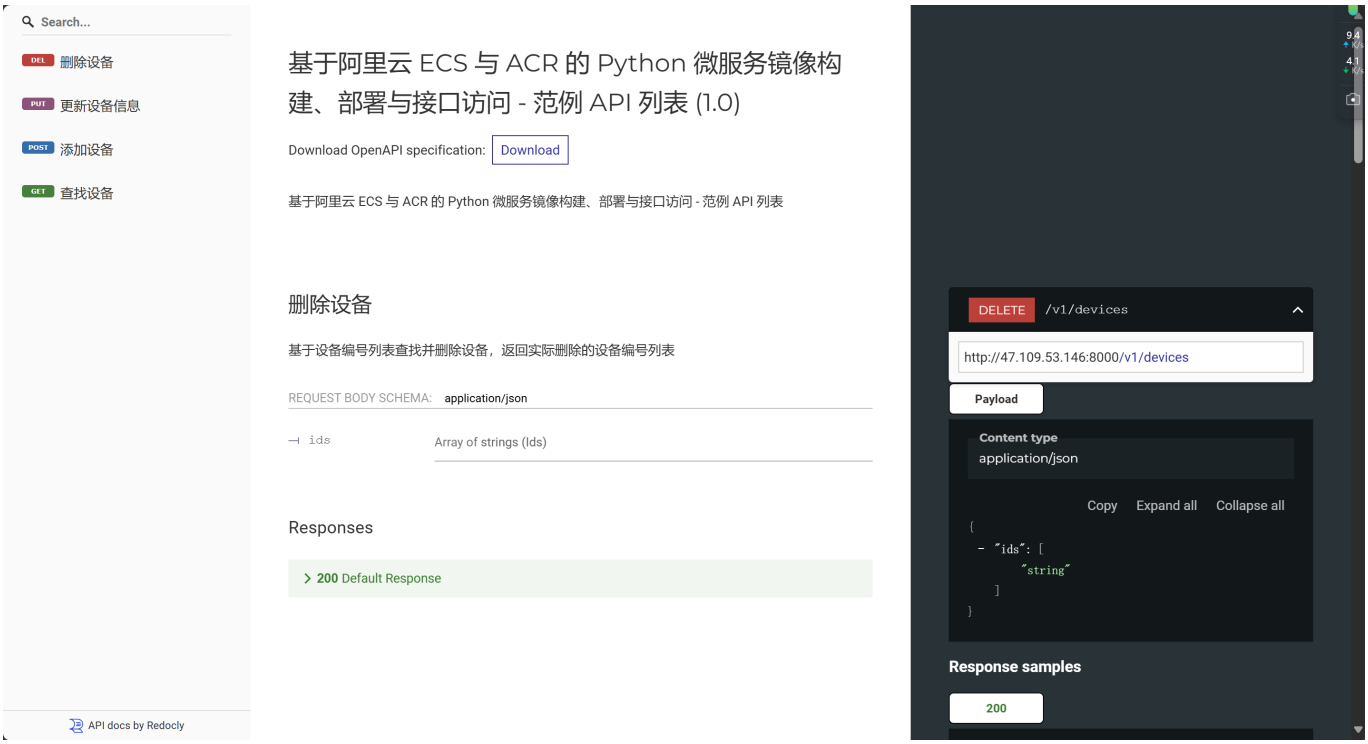


使用OpenAPI查看微服务接口

本实验微服务范例提供两种OpenAPI文档页面, 分别为Redocly 文档和Swagger .

- Redocly 文档提供了清晰、完整、详尽的接口与参考信息. 通过该文档, 可以了解服务包含哪些接口、接口请求与响应消息体等内容. 本实验Redocly 文档地址为<ecs_ip>:8000/docs

```
# 查看ecs_ip的方式
curl http://ipinfo.io/ip
```



- Swagger 文档除了类似 Redocly 提供接口与参数信息以外，还提供了最基础的调试功能。本实验Swagger文档地址为<ecs ip>:8000/docs/swagger。

请任选一类文档，认真阅读理解4个接口的详细定义，包括接口名称、所应用的HTTP 方法，在请求体 Request Body 中提交的 JSON 形式的接口参数，以及在 HTTP响应中返回的 JSON 形式的结果数据

观察结果： 以下是对该JSON文档中接口的进一步梳理和总结：

接口概述

- 该文档定义了一个基础路径为/v1/devices的API，包含了对设备进行增删改查操作的四个接口，分别对应HTTP方法DELETE、PUT、POST和GET。

1. DELETE接口（删除设备）

- **接口详情**
  - **接口名称**：delete~delete_devices
  - **功能总结**：根据提供的设备编号列表，删除相应设备，并返回实际删除的设备编号列表。
  - **请求参数说明**
    - `ids`（必需）：一个字符串数组，包含要删除的设备编号。例如：`["device1", "device2"]`。
  - **响应结果说明**
    - `status`：整数类型，表示操作的状态码，如200表示成功。
    - `message`：字符串类型，描述操作的结果消息，如“设备删除成功”。
    - `ids`：字符串数组，实际被删除的设备编号列表。

2. PUT接口（更新设备信息）

- **接口详情**
  - **接口名称**：put~update_devices
  - **功能总结**：先根据设备编号查找设备，然后根据提供的其他信息更新设备的各项属性。
  - **请求参数说明**
    - `id`（必需）：要更新的设备编号，字符串类型。
    - `name`（可选）：设备的新名称，字符串类型。
    - `type`（可选）：设备的新类型，字符串类型。
    - `hardware`（可选）：包含硬件信息的对象，有`model`（硬件型号，字符串类型）和`sn`（硬件序列号，字符串类型）两个属性。
    - `software`（可选）：包含软件信息的对象，有`version`（软件版本，字符串类型）和`last_update`（软件最后更新时间，字符串类型）两个属性。
    - `status`（可选）：设备的新状态，字符串类型。
  - **响应结果说明**
    - `status`：整数类型，表示操作的状态码。
    - `message`：字符串类型，操作结果消息。
    - `data`：对象类型，可能包含更新后的设备完整信息（文档未详细说明，可根据实际情况用于确认更新后的设备状态等）。

3. POST接口（添加设备）

- **接口详情**
  - **接口名称**：post~add_devices
  - **功能总结**：根据提供的设备信息，添加新设备到系统中。
  - **请求参数说明**
    - `name`（必需）：设备名称，字符串类型。
    - `type`（必需）：设备类型，字符串类型。
    - `hardware`（必需）：包含硬件信息的对象，有`model`（硬件型号，字符串类型）和`sn`（硬件序列号，字符串类型）两个属性。
    - `software`（必需）：包含软件信息的对象，有`version`（软件版本，字符串类型）和`last_update`（软件最后更新时间，字符串类型）两个属性。
    - `nic`（必需）：包含网络接口信息的数组，每个元素是一个对象，有`type`（网络接口类型，字符串类型）、`mac`（网络接口MAC地址，字符串类型）和`ipv4`（网络接口IPv4地址，字符串类型）三个属性。
    - `status`（必需）：设备初始状态，字符串类型。
  - **响应结果说明**

    - `status`：整数类型，表示操作的状态码。
    - `message`：字符串类型，操作结果消息。
    - `data`：字符串类型，可能是表示添加成功后设备的一些标识信息（文档未详细说明，可能是新设备的编号等）。

4. GET接口（查找设备）

- **接口详情**
    - **接口名称**：get~get_devices
    - **功能总结**：根据设备编号或"序列号SN-类型"组合查询设备信息。
    - **请求参数说明**
        - `id`（可选）：设备编号，字符串类型，用于精确查找特定设备。
        - `model`（可选）：设备型号，字符串类型，可用于模糊查询或组合查询。
        - `sn`（可选）：设备序列号，字符串类型，可用于精确或组合查询。可以单独使用这些参数，也可以组合使用进行更精确的查询。
    - **响应结果说明**
        - `status`：整数类型，表示操作的状态码。
        - `message`：字符串类型，操作结果消息。
        - `data`：数组类型，每个元素是一个对象，表示找到的设备信息，包含设备的各种详细属性，如`id`、`name`、`type`、`hardware`、`software`、`nic`和`status`等，具体属性含义与前面接口中的定义一致。

使用requests库编码访问微服务范例接口(本地)

配置Python虚拟环境

Python虚拟环境可以实现为不同的项目设置独立的依赖库, 以实现在统一系统中不同项目所需软件包的隔离. 例如开发环境中的两个项目都依赖一个包, 但是依赖于不同的版本, 或者一些依赖包是相互冲突的, 那么使用Python虚拟环境就能为不同的项目创建独立的虚拟环境.

本地开发环境创建工作目录`exp_pyms_cli` ,进入该目录安装启用Python虚拟环境`exp_venv`

```
python3 -m venv exp_venv
source exp_venv/bin/activate
```

随后使用`pip install` 命令在虚拟环境中安装Python requests库

```
pip install requests
# 觉得下载慢的话使用下面的指令 使用清华源
pip install requests -i https://pypi.tuna.tsinghua.edu.cn/simple
```

安装完成后, 可以在Python交互界面中进行初步验证

输入以下指令进入Python交互界面

```
python3

# 输入以下指令进行初步验证
import requests
requests.__version__
```

```
beryl@beryl-virtual-machine:~/exp_pyms_cli$ python3 -m venv exp_venv
beryl@beryl-virtual-machine:~/exp_pyms_cli$ source exp_venv/bin/activate
(exp_venv) beryl@beryl-virtual-machine:~/exp_pyms_cli$ pip install requests
Collecting requests
  Using cached requests-2.32.3-py3-none-any.whl (64 kB)
Collecting charset-normalizer<4,>=2
  Using cached charset_normalizer-3.4.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (144
 kB)
Collecting certifi>=2017.4.17
  Using cached certifi-2024.8.30-py3-none-any.whl (167 kB)
Collecting idna<4,>=2.5
  Using cached idna-3.10-py3-none-any.whl (70 kB)
Collecting urllib3<3,>=1.21.1
  Using cached urllib3-2.2.3-py3-none-any.whl (126 kB)
Installing collected packages: urllib3, idna, charset-normalizer, certifi, requests
Successfully installed certifi-2024.8.30 charset-normalizer-3.4.0 idna-3.10 requests-2.32.3 urllib3-2.2
.3
(exp_venv) beryl@beryl-virtual-machine:~/exp_pyms_cli$ python3
Python 3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> requests.__version__
'2.32.3'
>>> exit()
```

体验查询设备接口(本地+ECS)

在 ECS 工作目录中，查看先前创建的 exp_pyms_data目录，可以找到微服务范例程序启动时自动生成的 csv 文件，查看文件内容，可发现五条设备信息数据。(ECS)

```
cat beryl_exp_pyms_data/device.csv
```

```
4322159209,agTTIMCJ,controller,Raspberry Pi 4,itOYJ,42.58,2024-11-06 11:16:40,eth,50:71:1d:38:
f7:c9,98.165.26.115,wifi,b0:5b:96:91:9a:1d,245.111.182.79,online
3635138853,liJbFaIh,controller,x86 PC,zPXbV,57.31,2024-11-06 10:42:33,eth,34:4e:23:41:81:85,18
7.112.144.243,,,,online
5394030919,sJyAcmPW,adapter,Raspberry Pi 4,XznnF,19.17,2024-11-06 10:24:44,wifi,d5:62:0c:e6:7f
:51,51.211.27.123,,,,online
1846181623,YHtdVJAL,controller,Raspberry Pi 4,pxVHs,22.5,2024-11-06 11:32:54,wifi,36:41:65:80:
94:3e,164.38.116.213,,,,online
3102937453,FaqpGrqN,adapter,x86 PC,aIJOW,69.33,2024-11-06 08:53:34,eth,9d:f1:b2:cd:fc:f1,103.1
93.4.121,eth,5f:b4:88:19:cb:c0,7.45.212.2,offline
root@iZ2vcfk3gmwgxxap6tntf5Z:~#
```

以设备编号，即ID，为 4322159209的设备为例，通过 Requests 库访问「查询设备」接口，可基于设备编号获取该设备信息。(本地)

输入以下指令:

```
# 激活虚拟环境
source exp_venv/bin/activate

# 进入交互界面
python3
```

进行编程

```
import requests
resp = requests.get("http://<ecs_ip>:8000/v1/devices", json={"id": "5661391086"})

resp

resp.status_code

resp.text
```

```
(exp_venv) beryl@beryl-virtual-machine:~/exp_pyms_cli$ python3
Python 3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> resp = requests.get("http://47.109.53.146:8000/v1/devices", json={"id": "4322159209"})
>>> resp
<Response [200]>
>>> resp.status_code
200
>>> resp.text
'{"status":1,"message":"OK","data":[{"id":"4322159209","name":"agTTIMCJ","type":"controller","hardware"
:{"model":"Raspberry Pi 4","sn":"itOYJ"},"software":{"version":"42.58","last_update":"2024-11-06 11:16:
40"},"nic":[{"type":"eth","mac":"50:71:1d:38:f7:c9","ipv4":"98.165.26.115"},{"type":"wifi","mac":"b0:5b
:96:91:9a:1d","ipv4":"245.111.182.79"}],"status":"online"}]}'
>>>
```

这一查询接口同时还是支持另一种查询方式，即以设备“序号SN-类型”的组合信息为基准进行设备查询。

```
resp_ = requests.get("http://<ecs_ip>:8000/v1/devices", json={"sn": "itOYJ",
"model": "Raspberry Pi 4"})

resp_
resp_.status_code
resp_.test
```

```
>>> resp_ = requests.get("http://47.109.53.146:8000/v1/devices", json={"sn": "itOYJ", "model": "Raspber
ry Pi 4"})
>>> resp
<Response [200]>
>>> resp.status_code
200
>>> resp.text
'{"status":1,"message":"OK","data":[{"id":"4322159209","name":"agTTIMCJ","type":"controller","hardware"
:{"model":"Raspberry Pi 4","sn":"itOYJ"},"software":{"version":"42.58","last_update":"2024-11-06 11:16:
40"},"nic":[{"type":"eth","mac":"50:71:1d:38:f7:c9","ipv4":"98.165.26.115"},{"type":"wifi","mac":"b0:5b
:96:91:9a:1d","ipv4":"245.111.182.79"}],"status":"online"}]}'
```

体验[删除设备]接口

删除接口的消息体是一个设备编号列表，微服务范例程序将根据客户端提交的「删除设备编号列表」进行删除操作，并将世实际删除的设备编号列表返回。

以下执行「删除设备」接口访问练习。要求删除设备编号为 3635138853、5394030919、1234567890 的三台设备，其中编号为 1234567890 的设备并不存在。

注意观察返回结果。

```
resp_del = requests.delete("http://<ecs_ip>:8000/v1/devices", json={"ids":
["3635138853", "5394030919", "1234567890 "]})
resp_del.json()
```

```
>>> resp_del = requests.delete("http://47.109.53.146:8000/v1/devices", json={"ids": ["3635138853", "539
4030919", "1234567890 "]})
>>> resp_del.json()
{'status': 1, 'message': 'OK', 'data': ['5394030919', '3635138853']}
>>>
```

此时登陆 ECS 查看 device.csv 文件，设备编号为 3635138853、5394030919对设备信息已经被删除, 而 1234567890 没有影响。

```
root@iZ2vc6l[illegible]:~# cat [illegible]kp_pyms_data/device.csv
4322159209,agTTIMCJ,controller,Raspberry Pi 4,itOYJ,42.58,2024-11-06 11:16:40,eth,50:71:1d:38:
f7:c9,98.165.26.115,wifi,b0:5b:96:91:9a:1d,245.111.182.79,online
3635138853,liJbFaIh,controller,x86 PC,zPXbV,57.31,2024-11-06 10:42:33,eth,34:4e:23:41:81:85,18
7.112.144.243,,,,online
5394030919,sJyAcmPW,adapter,Raspberry Pi 4,XznnF,19.17,2024-11-06 10:24:44,wifi,d5:62:0c:e6:7f
:51,51.211.27.123,,,,online
1846181623,YHtdVJAL,controller,Raspberry Pi 4,pxVHs,22.5,2024-11-06 11:32:54,wifi,36:41:65:80:
94:3e,164.38.116.213,,,,online
3102937453,FaqpGrqN,adapter,x86 PC,aIJOW,69.33,2024-11-06 08:53:34,eth,9d:f1:b2:cd:fc:f1,103.1
93.4.121,eth,5f:b4:88:19:cb:c0,7.45.212.2,offline
root@i[illegible]~# cat [illegible]s_data/device.csv
4322159209,agTTIMCJ,controller,Raspberry Pi 4,itOYJ,42.58,2024-11-06 11:16:40,eth,50:71:1d:38:
f7:c9,98.165.26.115,wifi,b0:5b:96:91:9a:1d,245.111.182.79,online
1846181623,YHtdVJAL,controller,Raspberry Pi 4,pxVHs,22.5,2024-11-06 11:32:54,wifi,36:41:65:80:
94:3e,164.38.116.213,,,,online
3102937453,FaqpGrqN,adapter,x86 PC,aIJOW,69.33,2024-11-06 08:53:34,eth,9d:f1:b2:cd:fc:f1,103.1
93.4.121,eth,5f:b4:88:19:cb:c0,7.45.212.2,offline
```

体验[添加设备]接口

添加设备的接口的输入参数与获取设备信息的输入参数相比，只缺少了设备编号，即 `id` 字段设备编号信息将在成功添加设备后由云端微服务范例程序生成。

以下执行「添加设备」接口访问练习。

```
resp = requests.post(
    "http://<ecs_ip>:8000/v1/devices",
    json={
        "name": "test-name",
        "type": "controller",
        "hardware": {
            "model": "test-model",
            "sn": "test-sn"
        },
        "software": {
            "version": "0.1",
            "last_update": "2023-08-06 20:00:00"
        },
        "nic": [
            {
                "type": "wifi",
                "mac": "12:34:56:78:9a:bc",
                "ipv4": "192.168.1.2"
            }
        ],
        "status": "online"
    }
)
resp.json()
```

```
>>> resp = requests.post(
...     "http://47.109.53.146:8000/v1/devices",
...     json={
...         "name": "test-name",
...         "type": "controller",
...         "hardware": {
...             "model": "test-model",
...             "sn": "test-sn"
...         },
...         "software": {
...             "version": "0.1",
...             "last_update": "2023-08-06 20:00:00"
...         },
...         "nic": [
...             {
...                 "type": "wifi",
...                 "mac": "12:34:56:78:9a:bc",
...                 "ipv4": "192.168.1.2"
...             }
...         ],
...         "status": "online"
...     }
... )
>>> resp.json()
{'status': 1, 'message': 'OK', 'data': '6323569199'}
```

此时登陆 ECS 查看 device.csv 文件，发现已经增加了一条设备编号为9097363894的设备，其数据与刚才访问接口执行添加作的数据一致:

```
4322159209,agTTIMCJ,controller,Raspberry Pi 4,itOYJ,42.58,2024-11-06 11:16:40,eth,50:71:1d:38:
f7:c9,98.165.26.115,wifi,b0:5b:96:91:9a:1d,245.111.182.79,online
1846181623,YHtdVJAL,controller,Raspberry Pi 4,pxVHs,22.5,2024-11-06 11:32:54,wifi,36:41:65:80:
94:3e,164.38.116.213,,,,online
3102937453,FaqpGrqN,adapter,x86 PC,aIJOW,69.33,2024-11-06 08:53:34,eth,9d:f1:b2:cd:fc:f1,103.1
93.4.121,eth,5f:b4:88:19:cb:c0,7.45.212.2,offline
9097363894,test-name,controller,test-model,test-sn,0.1,2023-08-06 20:00:00,wifi,12:34:56:78:9a
:bc,192.168.1.2,,,,online
```

体验[更新设备信息]接口

微服务范例程序允许更新「设备名称、设备类型、设备硬件型号、设备硬件序号软件版本、软件上次更新时间、设备状态信息」，在请求体中组织 JSON 形式的参数时，以上字段的 JSON 嵌套方式与「添加设备」接口的 JSON 规范一致。此外，调用「更新设备信息」接口要求必须指定设备编号，否则无法定位具体设备。

以下执行「更新设备信息」接口访问练习, 这里以刚刚创建的编号为9097363894 的设施为例，将其设备名称更新为new-name，软件版本更新为0.5，设备状态更新为offline。

```
resp = requests.put(
    "http://<ecs_ip>:8000/v1/devices",
    json={
        "id": "9097363894",
        "name": "new-name",
        "software": {
            "version": "0.5",
            "last_update": "2023-08-06 10:00:00"
        },
        "status": "offline"
    }
)
resp.json()
```

```
>>> resp.json()
{'status': 1, 'message': 'OK', 'data': '6323569199'}
>>> resp = requests.put(
...     "http://47.109.53.146:8000/v1/devices",
...     json={
...         "id": "9097363894",
...         "name": "new-name",
...         "software": {
...             "version": "0.5",
...             "last_update": "2023-08-06 10:00:00"
...         },
...         "status": "offline"
...     }
... )
>>> resp.json()
{'status': 1, 'message': 'OK'}
```

此时到云端服务器中查看 `device.csv` 文件，数据已经成功修改:

```
9097363894,new-name,controller,test-model,test-sn,0.5,2023-08-06 10:00:00,wifi,12:34:56:78:9a:bc,192.168.1.2,,,,offline
```