如何使用 airscheduler

url: http://192.168.9.64:30887/api/v1/airscheduler/task

一、快速启动任务或服务

简单示例：

```
# coding: utf-8
import json
import requests

input_data = {
        'name': ...,
        ...
        'running_config': json.dumps(running_config), ---------- #  任务配置信息
        'resource_info': json.dumps(resource_info), ---------- #  任务资源需求信息
        ...
        'start_now': True
}    ---------- #  任务基本信息

return_data = requests.post(' http://192.168.9.64:30887/api/v1/airscheduler/task ',
json=input_data, headers={'token': token})
```

二、输入数据含义

(a) 任务基本信息 — input_data (dict)

键名	类型	描述	默认值
name	string	该任务或服务的名称	
usage	string	说明该任务申请资源的用途(建议中文)	
source_service_id	integer	源服务端存储 id	
running_type	integer	0: k8s 调度	0
working_type	integer	0: 任务　1: 服务	
namespace	string	命名空间	
image_repo_tag	string	镜像全称	
start_now	boolean	是否快速启动	

(b) 任务配置信息 — running_config (dict)

针对任务的配置信息：

键名	类型	描述	默认值
volumes	list	容器外部路径与内部路径的映射	
pvc	dict	pvc	
command	string	任务运行命令	例：python run.py
work_dir	string	容器内代码工作路径	/code
yaml_data	Json.dumps(dict)	**特殊字段**，支持直接从 yaml 文件启动任务。普通任务忽略该字段	

pvc 填写示例:

{ pvc_requests: 100Mi, pvc_limits: 1Gi}

volumes 填写示例：

```
[{
        'is_nfs': False,
        'server': '',
        'path': 容器外路径,
        'mount_path': 容器内路径,
        'mount_name': 'code'
}]
```

针对服务的配置信息：

键名	类型	描述	默认值
volumes	list	容器外部路径与内部路径的映射	
pvc	dict	pvc	
port	list	服务端口号之间映射	
work_dir	string	容器内代码工作路径	/code
command	string	任务运行命令	例：python run.py

pvc 填写示例:

{ pvc_requests: 100Mi, pvc_limits: 1Gi}

dry

port 填写示例：

```
[{
    node_port: 30000,   # 外部端口号
    code_port: 5000      # 映射端口号
}]
```

volumes 填写示例：

```
[{
    'is_nfs': True,
    'server': '',
    'path':  容器外路径,
    'mount_path':  容器内路径,
    'mount_name': ''
},
    {
        'is_nfs': False,
        'server': '',
        'path':  容器外路径,
        'mount_path':  容器内路径,
        'mount_name': 'code'
    }
]
```

(c) 任务资源需求信息 — resource_info (dict)

键名	类型	描述	默认值
cpu_count	integer	任务或服务需要的 cpu 个数（单位：个）	
mem_size	integer	任务或服务需要的内存大小（单位：字节）	
gpu_dict	Json.dumps(dict)	任务或服务需要的 GPU 类型与数量	
shm_size	integer	任务或服务需要的共享内存大小（单位：字节）	

- 附：目前可调用的 GPU 型号

GeForce GTX 1080; GeForce 1080 Ti; GeForce RTX 2080 Ti 等

三、功能

*** 库位置：lib.service_util**

（a）创建新任务或新服务 -- 一键启动

请求：

```
# coding: utf-8
from lib.service_util import create_object_in_airscheduler

return_data = create_object_in_airscheduler ( base_url, input_data, token )
```

功能描述：

- 创建新任务，如果 start_now 为 True：快速启动任务。反之，挂起任务，等待功能（d）启动该任务。
- token 来源于用户登录，默认采用 账号：qiyan1 密码：123456 获取 token

返回状态：

0：成功，返回已创建任务的详细信息，添加专属字段 id

193000：输入数据错误

193002：无法请求资源

193004：任务启动失败

其他：可连接资源请求，但没有对应资源或请求资源失败等等

（b）修改任务或服务

请求：

```
# coding: utf-8
from lib.service_util import modify_object_in_airscheduler

return_data = modify_object_in_airscheduler ( base_url, input_data, token )
```

- input_data 中必须额外包含已创建任务的 id

功能描述：

- 修改某个已创建的任务的详细信息，如果该任务已运行，会强制停止并删除已运行容器，并将修改后的任务挂起，等待功能（d）启动该任务。

- 如果该任务未运行，修改该任务的详细信息并挂起，等待功能（d）启动该任务。
- token 来源于用户登录，默认采用 账号：qiyan1 密码：123456 获取 token

返回状态：

0：成功

193000：输入数据错误

193001：该任务不存在

193006：删除任务错误

（c）彻底删除任务或服务

请求：

```
# coding: utf-8
from lib.service_util import delete_object_in_airscheduler

return_data = delete_object_in_airscheduler ( base_url, task_id, token )
```

功能描述:

- 彻底删除某个任务以及其对应的数据库
- token 来源于用户登录，默认采用 账号：qiyan1 密码：123456 获取 token

返回状态：

0：成功

193000：输入数据错误

193001：该任务不存在

193006：删除任务错误

（d）启动任务或服务

请求：

```
# coding: utf-8
from lib.service_util import start_object_in_airscheduler

return_data = start_object_in_airscheduler ( base_url, task_id, token )
```

功能描述：

- 启动某一个挂起的任务。
- token 来源于用户登录，默认采用 账号：qiyan1 密码：123456 获取 token

返回状态：

0：成功

193000：输入数据错误

193001：该任务不存在

193002：资源请求出错

193004：任务启动出错

193006：删除任务错误

193007：重复启动该任务

（e）获取任务或服务的详细信息

请求：

```
# coding: utf-8
from lib.service_util import observe_object_in_airscheduler

return_data = observe_object_in_airscheduler ( base_url, task_id, token )
```

功能描述：

- 查询某一个任务的详细信息。
- token 来源于用户登录，默认采用 账号：qiyan1 密码：123456 获取 token

返回状态：

0：成功

193000：输入数据错误

193001：该任务不存在

（f）筛选任务或服务

请求：

```
# coding: utf-8
from lib.service_util import filter_object_in_airscheduler
```

return_data = filter_object_in_airscheduler (base_url, col_name, col_values, token)

功能描述：

- 根据特定条件批量筛选任务，得到多个任务的详细信息。
- token 来源于用户登录，默认采用 账号：qiyan1 密码：123456 获取 token

返回状态：

0：成功

193000：输入数据错误

193003：查询键值错误

（g）停止任务或服务

请求：

```
# coding: utf-8
from lib.service_util import stop_object_in_airscheduler

return_data = stop_object_in_airscheduler ( base_url, task_id, token )
```

功能描述：

- 删除某一个已启动的任务。
- token 来源于用户登录，默认采用 账号：qiyan1 密码：123456 获取 token

返回状态：

0：成功

193000：输入数据错误

193001：该任务不存在

193006：删除任务错误

193008：该任务启动超时或未启动

（h）任务静态日志查询

请求：

```
# coding: utf-8
from lib.service_util import object_logs_in_airscheduler
```

return_data = object_logs_in_airscheduler (base_url, task_id, start_line, end_line, token)

- start_line : 取值为 -1,+∞ 。-1 表示读取当前所有日志
- end_line: 取值为 -1,+∞ 。-1 表示读取到当前日志的结尾

功能描述：

- 读取并返回某任务所创建容器的日志信息。
- token 来源于用户登录，默认采用 账号：qiyan1 密码：123456 获取 token

返回状态：

0：成功，包含当前日志的总行数，以及日志内容

193000：输入数据错误

193001：该任务不存在

- 特殊 code：

  2：该任务容器运行出错，错误详见返回信息

  3：该任务容器创建与查询出错，错误详见返回信息

返回数据：

{ code : 0, message: success, logs : '当前任务截取的日志', 'total_lines': 当前任务日志的总行数}

（i）任务静态资源使用率查询

请求：

```
# coding: utf-8
from lib.service_util import object_resources_in_airscheduler

return_data = object_ resources_in_airscheduler ( base_url, task_id, token )
```

功能描述：

- 读取并返回某任务所创建容器的资源使用情况。
- token 来源于用户登录，默认采用 账号：qiyan1 密码：123456 获取 token

返回状态：

0：成功

193000：输入数据错误

193001：该任务不存在

- 特殊 code：
  2：该任务容器运行出错，错误详见返回信息
  3：该任务容器创建与查询出错，错误详见返回信息

返回数据：

```
{    code : 0,
     message: success,
     data: {
          cpu_usage: cpu 的使用量,
          cpu_total: cpu 申请的总量,
          cpu_use_rate: cpu 的使用率,
          memory_usage:  内存的使用量,
          memory_total:  内存申请的总量 ,
          memory_use_rate:  内存使用率}
}
```

如何使用 airscheduler – real time monitor

Socket host: http://192.168.9.64:30888/

四、功能 real time monitor

(a) 实时 cpu、内存占用监控

请求：

event : monitor_cpu

- 客户端发送的 message：

  task_id: 任务 id

- 客户端接收的 message：

情况 1：

任务不存在，任务查询失败

{ code: 1, msg: 任务信息查询失败}

情况 2:

任务信息不存在，查询不到对应容器

{ code: 1, msg: 任务信息不存在}

情况 3：

查询成功，返回

- 任务 id，
- cpu 占用量，cpu 占用率，
- 内存占用量，内存占用率，
- 运行时间，
- cpu 请求总量，内存请求总量

功能描述：

- 实时监控某个任务的当前 cpu 占用率等信息。

(b) 实时日志监控

请求：

event : monitor_logs

- 客户端发送的 message：

  { task_id: 任务 id，start_line: 日志起始行(-1,+∞)，end_line: 日志结尾行(-1,+∞) }

- 客户端接收的 message：

情况 1：

任务不存在，任务查询失败

{ code: 1，msg: 任务信息查询失败}

情况 2:

未获得任何日志信息

{ code: 1，msg: 未获得任何日志信息}

情况 3：

查询成功，返回

- 任务 id，
- 日志内容，
- 当前时间，
- 运行时间，
- **当前日志总行数**

功能描述：

- 实时监控某个任务的日志信息。支持全量日志显示，部分日志显示，日志增量显示

(c) 实时任务状态监控

请求：

event : monitor_status

- 客户端发送的 message：

  { task_ids: 任务 ids }

- 客户端接收的 message：

情况 1：

任务不存在，任务查询失败

{ code: 1, msg: 任务信息查询失败}

情况 2：

查询成功，返回

- 每一个任务 id 对应的状态，
- 当前时间，

功能描述：

- 实时监控批量任务的运行状态

五、sample for real time monitor

```
# coding: utf-8
import time

import socketio

def sample_f_rt_monitor(uid=0):
    sio = socketio.Client()
    event = 'monitor_cpu' ----------------------------------修改 event，选择对应功能

    @sio.on(event)
    def response(*args):
        print('return_data:', args)

    @sio.event
    def connect():
        print('connected')

    @sio.event
    def connect_error():
        print('failed')

    @sio.event
    def disconnect():
        print('disconnected')
```

```
    @sio.event
    def send_message(data):
        sio.emit(event, data=data)

    host = 'http://192.168.9.64:30888'
    sio.connect(host)

    patience = 10000
    t = 0
    while t < patience:
        time.sleep(1)
        data = uid    # task_id ----------------------------------修改 data，适配功能
        send_message(data)
        t += 1

    sio.wait()

sample_f_rt_monitor(uid=1)    # uid: task_id
```