使用元数据通过API和Python自定义报告

目录

[简介](#)
[先决条件](#)
[要求](#)
[使用的组件](#)
[背景信息](#)
[设置元数据](#)
[收集API密钥](#)
[创建自定义报告](#)
[相关信息](#)

简介

本文档介绍如何将元数据与API结合使用，以便在Python脚本中自定义报告。

先决条件

要求

Cisco 建议您了解以下主题：

- CloudCenter
- 蟒

使用的组件

本文档不限于特定的软件和硬件版本。

本文档中的信息都是基于特定实验室环境中的设备编写的。本文档中使用的所有设备最初均采用原始（默认）配置。如果您使用的是真实网络，请确保您已经了解所有命令的潜在影响。

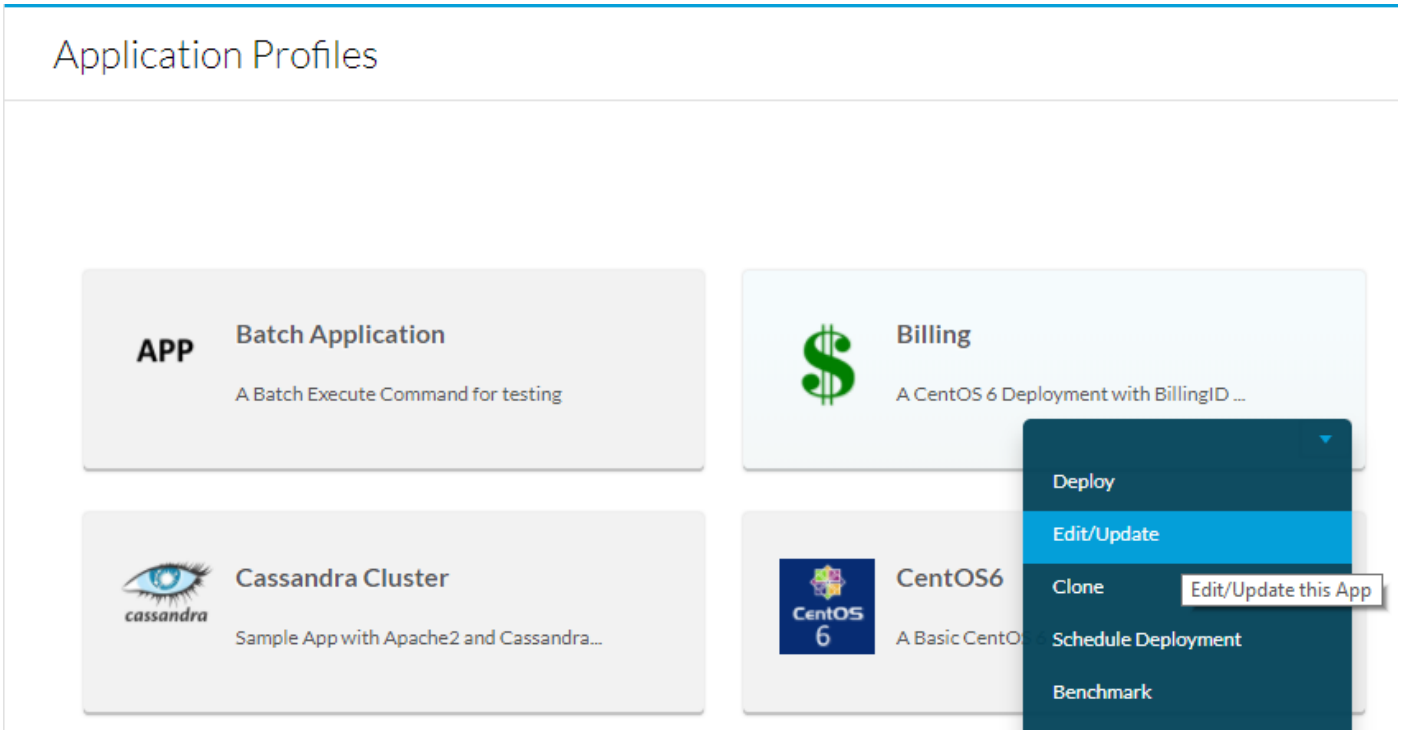
背景信息

CloudCenter提供开箱即用的一些报告，但它不允许使用基于自定义过滤器的报告。要使用API直接从数据库获取信息，连同附加到作业的元数据，您可以允许自定义报告。

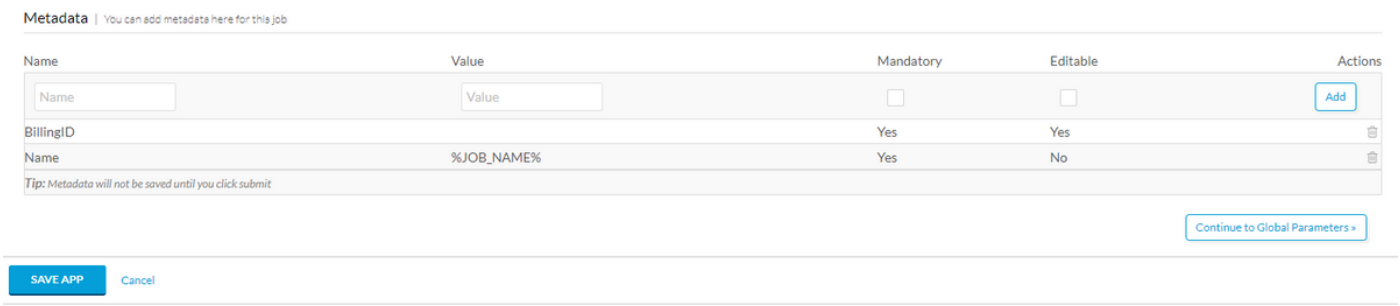
设置元数据

元数据必须在每个应用级别上添加，因此需要使用自定义报告跟踪的每个应用都必须修改。

为此，请导航至“应用配置文件”，然后为要编辑的应用选择下拉列表，然后选择“编辑/更新”，如图**所示。**



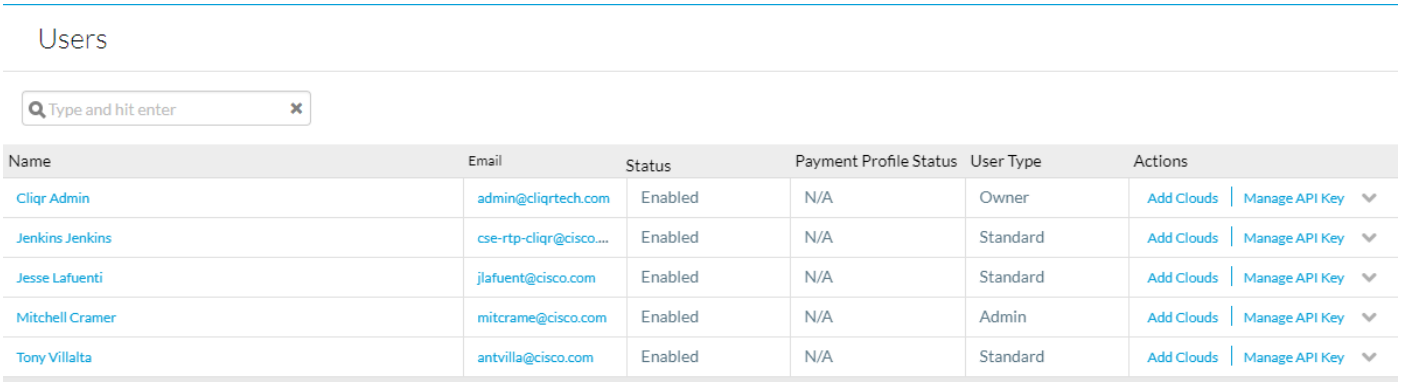
滚动到“基本信**息**”的底部并添加元数据标记(例如**BillingID**)，如果用户要填写此元数据，则将其设为强制和可编辑。如果它只是宏，则填写默认值，不使其可编辑。填写元数据后，选择**添加**,**保存应**用，如图所示。



收集API密钥

要处理API调用，需要用户名和API密钥。这些密钥提供与用户相同的访问级别，因此，如果要在报告中添加所有用户部署，则建议使用该密钥来获取租户API密钥的管理员。如果要同时记录多个子租户，则根租户需要访问所有部署环境，或者需要所有子租户管理员的API密钥。

要获取API密钥，请导航至**Admin > Users > Manage API Key**，复制所需用户的用户名和密钥。



创建自定义报告

在创建用于创建报告的python脚本之前，请确保已在其上安装了python和pip。然后运行**pip install tablate**,tablate是一个自动处理格式化报告的库。

本指南附有两个示例报告，首先只是收集有关所有部署的信息，然后将其输出到表中。第二个使用相同信息创建使用BillingID元数据的自定义报告。本脚本将作为指南作详细说明。

```
import datetime
import json
import sys
import requests
##pip install tabulate
from tabulate import tabulate
from operator import itemgetter
from decimal import Decimal
```

datetime用于准确计算日期，这是为了创建最近X天的报告。

json用于帮助解析json数据，即api调用的输出。

sys用于系统调用。

请求用于简化对API调用的Web请求。

表格用于自动格式化表格。

itemgetter用作迭代器对2D表进行排序。

十进制用于将成本舍入为两个小数位。

```
if(len(sys.argv)==1):
        days = -1
elif(len(sys.argv)==2):
        try:
                days = int(sys.argv[1])
                if(days < 1):
                        raise ValueError('Less than 1')
                start=datetime.datetime.now()+datetime.timedelta(days*-1)
        except ValueError:
                print("Number of days must be an integer greater than 0")
                exit()
else:
        print("Enter number of days to report on, or leave blank to report all time")
        exit()
```

此部分用于分析天数的命令行参数。

如果没有命令行参数(sys.argv ==1)，则报告将始终完成。

如果有一个命令行参数检查它是否是大于或等于1的整数，如果在该天数上报告它，则返回错误。

如果有多个参数，则返回错误。

```
departments = []
users = ['user1','user2','user3']
passwords = ['user1Key','user2Key','user3Key']
```

departments是包含最终输出的列表。

用户是将进行API调用的所有用户的列表，如果有多个子租户，则每个用户将是不同子租户的管理员。

密码是用户API密钥的列表，用户和密钥的顺序需要相同才能使用正确的密钥。

```
for j in xrange(0,len(users)):
        jobs = []
        r = requests.get('https://ccm2.cisco.com/v1/jobs', auth=(users[j], passwords[j]),
headers={'Accept': 'application/json'})
        data = r.json()
        for i in xrange(0,len(data["jobs"])):
                test = datetime.datetime.strptime((data["jobs"][i]["startTime"]), '%Y-%m-%d
%H:%M:%S.%f')
                if(days != -1):
                        if(start < test):
                                jobs.append([data["jobs"][i]["id"],'None',
data["jobs"][i]["cost"]["totalCost"],data["jobs"][i]["status"],data["jobs"][i]["displayName"],da
ta["jobs"][i]["startTime"]])
                else:
                        jobs.append([data["jobs"][i]["id"],'None',
data["jobs"][i]["cost"]["totalCost"],data["jobs"][i]["status"],data["jobs"][i]["displayName"],da
ta["jobs"][i]["startTime"]])
        for id in jobs:
                q = requests.get('https://ccm2.cisco.com/v1/jobs/'+id[0], auth=(users[j],
passwords[j]), headers={'Accept': 'application/json'})
                data2 = q.json()
                id[2]=round(id[2],2)
                for i in xrange(0,len(data2["metadatas"])):
                        if('BillingID' == data2["metadatas"][i]["name"]):
                                id[1]=data2["metadatas"][i]["value"]
                added=0
                for i in xrange(0,len(departments)):
                        if(departments[i][0]==id[1]):
                                departments[i][1]+= 1
                                departments[i][2]+=id[2]
                                added=1
                if(added==0):
                        departments.append([id[1],1,id[2]])
```

对于xrange(0,len(users))中的j:是循环，用于循环，以循环遍历在前一个代码块中定义的每个用户，这是处理所有API调用的主循环。

jobs是临时列表，用于保存将其整理到列表中的作业信息。

r =请求。获取.....是第一个API调用，此调用列出所有作业，有关详细信息，请参阅列表作业。

结果随后以json格式存储为数**据。**

对于xrange(0,len(data["jobs"]))中的i:迭代通过从上一个API调用返回的所有作业。

从json提取每个作业的时间并将其转换为日期时间对象，然后将其与输入的命令行参数进行比较

，以查看它是否在界限内。

如果是，则附加到作业列表的是json中的以下信息：**id、totalCost、status、name、start time**。并非所有这些信息都被使用，也不是所有可返回的信息。列表作业显示所有可以以相同方式添加的返回信息。

在迭代完成从该用户返回的所有作业后，将移至作**业中的id:**它迭代了您检查开始日期后执行的所有作业。

q =请求.get(.....是第二个API调用，此调用列出与从第一个API调用获取的作业ID相关的所有信息。有关详细信息，请参阅获取作业详细信息。

然后json文件存储在**data2中。**

存储在id[2]中的**成本四舍**一入为两个小数位。

对于xrange(0,len(data2["metadatas"])中的i:迭代所有与作业关联的元数据。

如果有称为**BillingID**的元数据，则该元数据存储在作业信息中。

added是用于确定BillingID是否已**添加到**部门列表的**标**志。

对于xrange(0,len(departments))中的i:通过所有已添加的部门进行迭代。

如果此作业是已存在的部门的一部分，则作业计数将重复1，并将成本添加到该部门的总成本中。

否则，系统会将新行附加到任务计数为1且总成本等于此任务成本的部门。

```
departments = sorted(departments, key=itemgetter(1))
print(tabulate(departments,headers=['Department','Number of Jobs','Total Cost']))
```

departments = sorted(departments， key=itemgetter(1))按 Number of Jobs对部门进行排序。

print(tabalet(departments，headers=['Department','Number of Jobs', 'Total Cost']))打印由带有三个标题的tableate创建的表。

相关信息

- CloudCenter API
- **技术支持和文档 - Cisco Systems**