利用後設資料通過API和Python自定義報告

目錄

簡介

本文檔介紹如何將後設資料與API結合使用，以便在python指令碼中自定義報告。

必要條件

需求

思科建議您瞭解以下主題：

- CloudCenter
- Python

採用元件

本文件所述內容不限於特定軟體和硬體版本。

本文中的資訊是根據特定實驗室環境內的裝置所建立。文中使用到的所有裝置皆從已清除（預設）的組態來啟動。如果您的網路正在作用，請確保您已瞭解任何指令可能造成的影響。

背景資訊

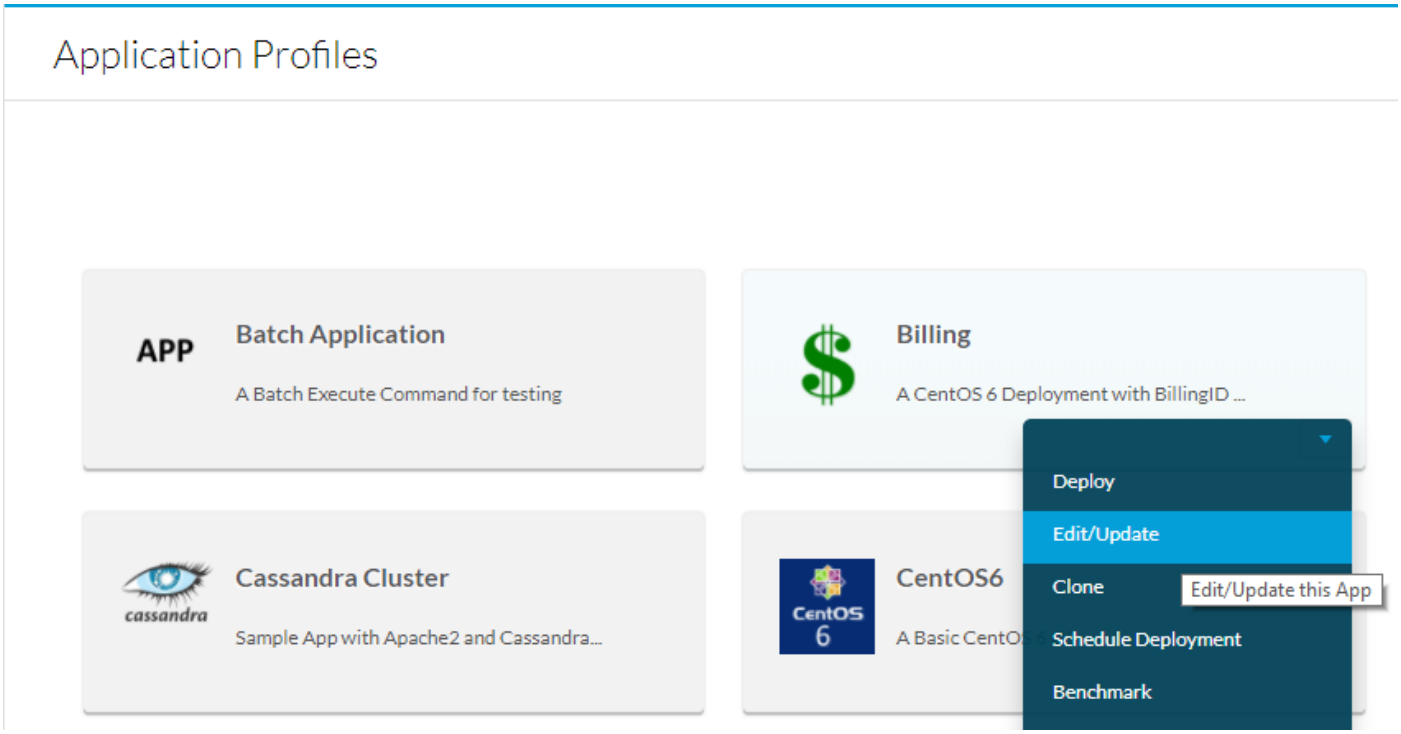
CloudCenter提供一些開箱即用的報告，但不允許使用基於自定義篩選器的報告方式。為了使用API直接從資料庫中獲取資訊，以及連線到作業的後設資料，您可以允許自定義報告。

設定後設資料

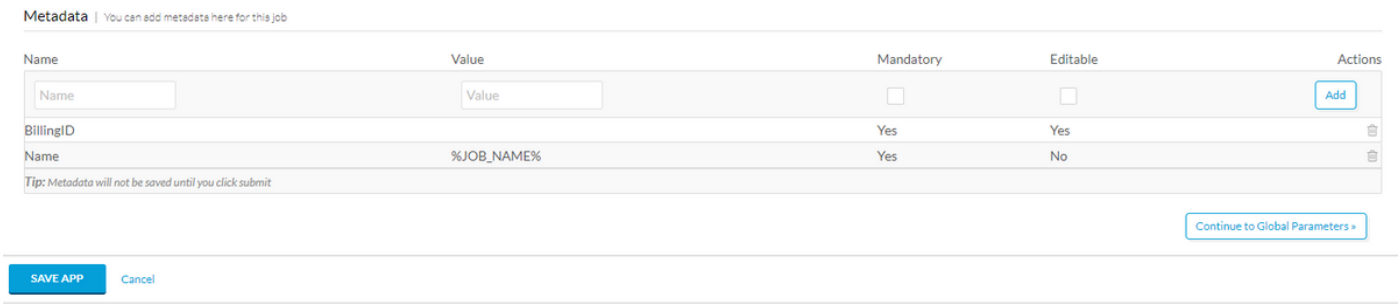
必須在每個應用程式級別上新增後設資料，因此必須修改需要使用自定義報告進行跟蹤的每個應用程式。

為此，請導航到**應用程式配置檔案**，然後選擇要編輯的應用程式的下拉選單，然後選擇**編輯/更新**

（如圖所示）。



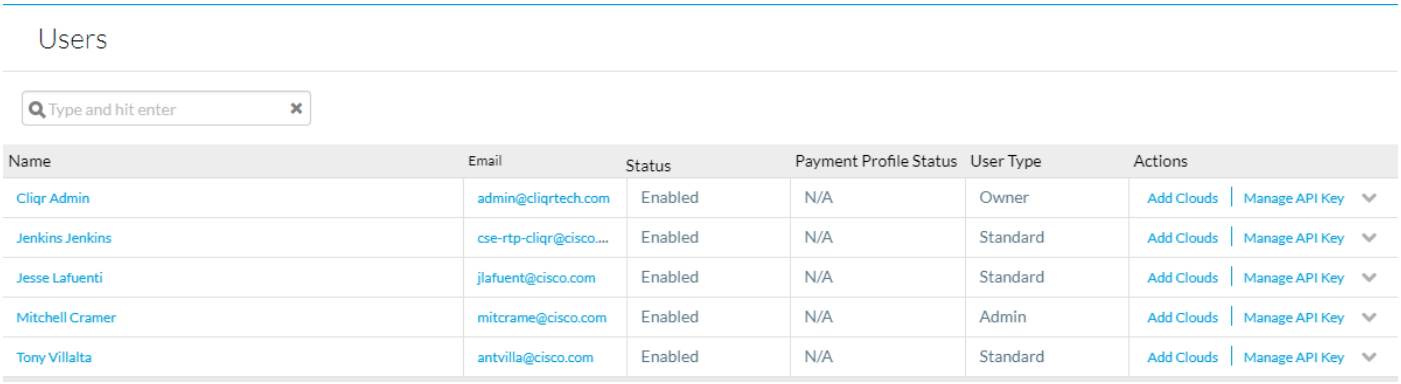
如果使用者要填寫此後設資料，請滾動到**基本資訊**的底部並新增後設資料標籤(例如**BillingID**)，使其成為必需和可編輯的後設資料。如果它只是一個宏，則填寫預設值並且不能使其可編輯。填寫後設資料後，選擇**Add**，然後選擇**Save App**，如下圖所示。



收集API金鑰

為了處理API呼叫，需要使用者名稱和API金鑰。這些金鑰提供與使用者相同的訪問級別，因此，如果要在報告中新增所有使用者部署，則建議獲取租戶API金鑰的管理許可權。如果要一起記錄多個子租戶，則根租戶需要訪問所有部署環境，或者需要所有子租戶管理員的API金鑰。

要獲取API金鑰，請導航到**Admin > Users > Manage API Key**，複製所需使用者的使用者名稱和金鑰。



建立自定義報告

建立用於建立報告的python指令碼之前，請確保已在其上安裝python和pip。然後運行**pip install tabulate**,tabulate是一個自動處理格式化報表的庫。

此指南附加了兩個示例報告，第一個報告只是收集所有部署的資訊，然後將其輸出到表中。第二種使用相同資訊建立使用BillingID後設資料的自定義報告。將對此指令碼進行詳細解釋以用作指南。

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

datetime用於準確計算日期，建立最近X天的報告。

json用於幫助分析json資料，即api呼叫的輸出。

sys用於系統呼叫。

請求用於簡化對API呼叫的網路請求。

表格用於自動設定表格格式。

itemgetter用作對2D表進行排序的迭代器。

十進制用於將成本舍入為兩個小數位。

```
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()
```

此部分用於分析天數的命令列引數。

如果沒有命令列引數(sys.argv ==1)，則報告將始終完成。

如果有一個命令列引數，請檢查該引數是否為大於或等於1的整數，如果在該天數上報告該引數，如果不是，則返回錯誤。

如果有多個引數，則返回錯誤。

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

departments是儲存最終輸出的清單。

用戶是將要進行API呼叫的所有使用者的清單，如果存在多個子租戶，則每個使用者將成為不同子租戶的管理員。

passwords是使用者API金鑰的清單，使用者和金鑰的順序需要相同，才能使用正確的金鑰。

```
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]])
```

對於xrange(0,len(users))中的j:is for loop to iterate through evidual code chunk中定義的每個使用者，這是處理所有API呼叫的主循環。

作業是一個臨時清單，在將作業資訊整理到清單中時，該清單將用於儲存該作業的資訊。

r = requests.get.....是第一個API呼叫，此呼叫列出所有作業，有關詳細資訊，請參閱列出作業。

然後，將結果以json格式儲存在**資料**中。

對於xrange(0,len(data["jobs"])中的i：重複執行從上一個API呼叫返回的所有作業。

從json提取每個作業的時間並將其轉換為datetime對象，然後將其與輸入的命令列引數進行比較，以檢視其是否在界限內。

如果是，則附加至作業清單的是json中的以下資訊：**id、totalCost、status、name、start time**。不會使用所有這些資訊，也不會使用可以返回的所有資訊。清單作業顯示可以相同方式新增的所有返回資訊。

在迭代該使用者返回的所有作業後，將移至**作業中的for id:**它循環檢查開始日期後執行的所有作業。

q = requests.get(...)..是第二個API呼叫，該呼叫列出與從第一個API呼叫中獲取的作業ID相關的所有資訊。 有關詳細資訊，請參閱獲取作業詳細資訊。

json檔案隨後儲存在**data2中。**

儲存在id[2]中的**成本將四捨五入**為兩個小數位。

對於xrange(0,len(data2["metadatas"])中的i:循環與作業關聯的所有後設資料。

如果有名為**BillingID**的後設資料，則此後設資料將儲存在作業資訊中。

added是用於確定BillingID是否**已**新增到部門列**表**的標誌。

對於xrange(0,len(departments))中的i:循環訪問新增的所有部門。

如果此作業是已存在部門的一部分，則作業計數將逐個迭代，並且成本將新增到該部門的總成本中。

如果不是，則將新行附加到任務計數為1且總成本等於此任務的成本的部門。

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

departments = sorted(departments， key=itemgetter(1))按作業數對部門排序。

print(tablet(departments，headers=['Department','Jobs Number', 'Total Cost']))列印由包含三個標題的表格創建的表格。

相關資訊

- CloudCenter API
- **技術支援與文件 - Cisco Systems**