Utilize Metadata aos relatórios personalizados com API e pitão

Índice

Introdução

Este documento descreve como usar metadata conjuntamente com relatórios personalizados API dentro de um script do pitão.

Pré-requisitos

Requisitos

A Cisco recomenda que você tenha conhecimento destes tópicos:

- CloudCenter
- Pitão

Componentes Utilizados

Este documento não se restringe a versões de software e hardware específicas.

As informações neste documento foram criadas a partir de dispositivos em um ambiente de laboratório específico. Todos os dispositivos utilizados neste documento foram iniciados com uma configuração (padrão) inicial. Se a sua rede estiver ativa, certifique-se de que entende o impacto potencial de qualquer comando.

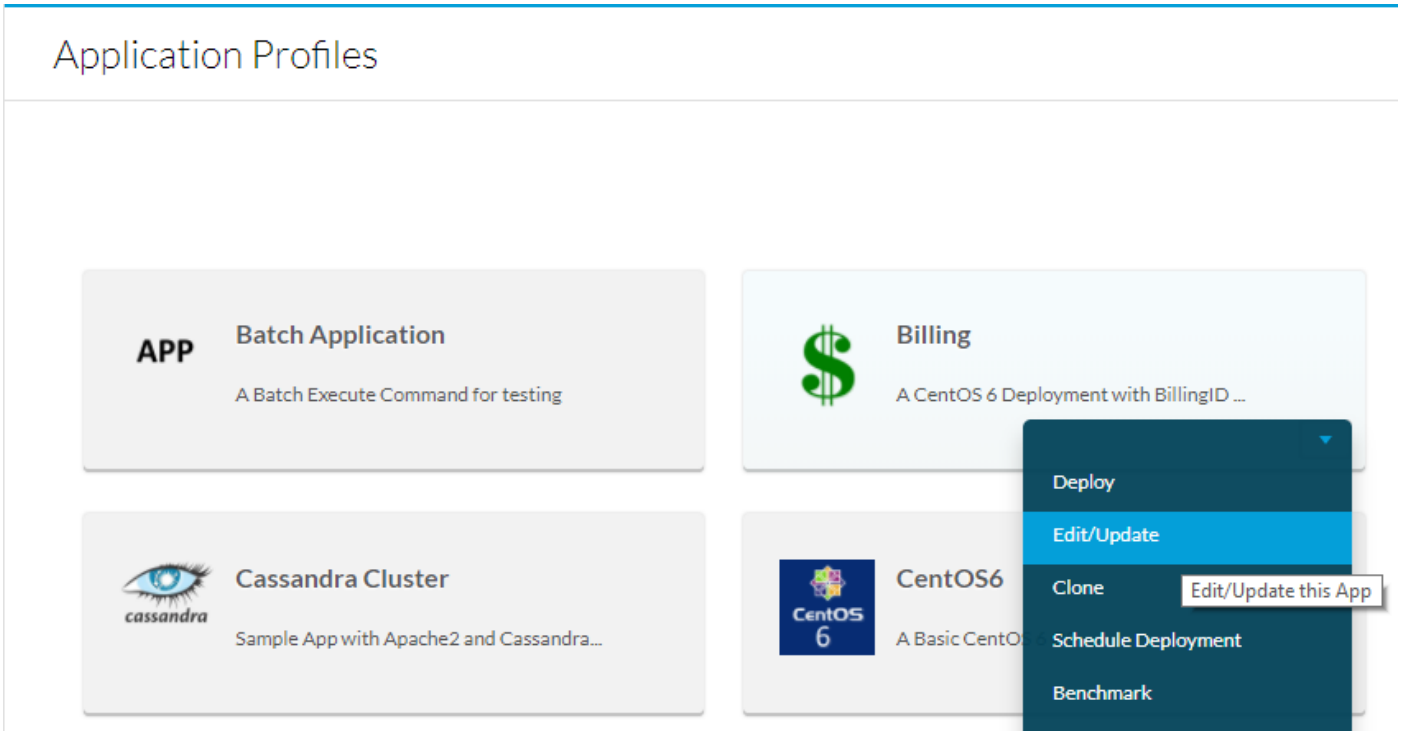
Informações de Apoio

CloudCenter fornece algum relatório fora da caixa, porém não permite uma maneira para os relatórios baseados em filtros feitos sob encomenda. A fim usar API a fim agarrar a informação diretamente do base de dados, conjuntamente com metadata anexado aos trabalhos, você pode permitir relatórios personalizados.

Estabelecer os Metadata

Os Metadata devem ser adicionados na pelo nível do aplicativo, tão cada aplicativo que precisa de ser seguido com o uso dos relatórios personalizados terá que ser alterado.

A fim fazer isto, para navegar aos **perfis do aplicativo**, para selecionar então o dropdown para que o App seja editado e para selecioná-lo então **edite/atualização** segundo as indicações da imagem.



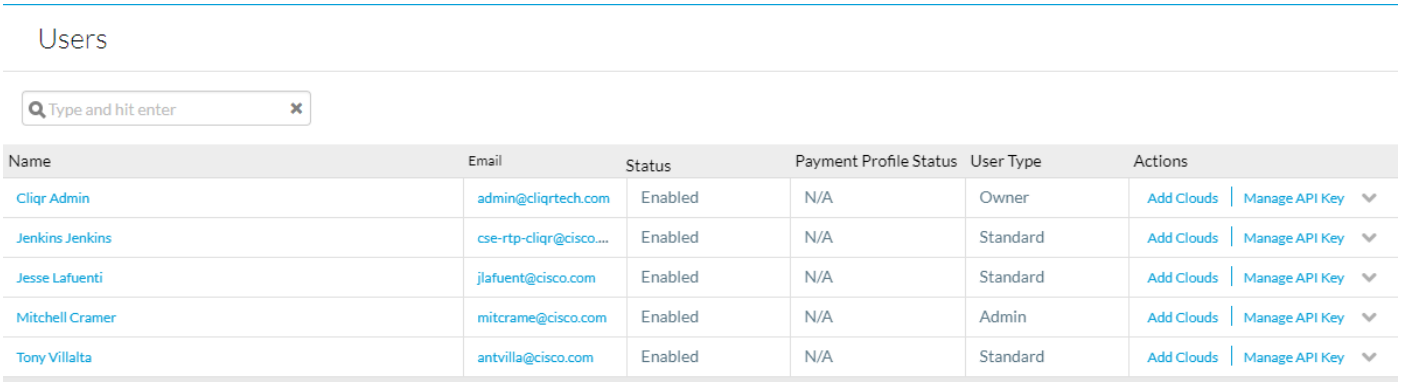
O rolo à parte inferior da **informação básica** e adiciona Metadata etiqueta, por exemplo **BillingID**, se este os metadata devem ser completados pelo suer fá-lo imperativo e editável. Se é apenas um macro, a seguir preencha o valor padrão e não o faça editável. Depois que você completa os metadata, seletos **adicionar** então o **App da salvaguarda** segundo as indicações da imagem.

Recolha chaves API

A fim processar os atendimentos API, o username e as chaves API serão exigidos. Estas chaves fornecem o mesmo nível do acesso que o usuário, assim que se todas as disposições dos usuários devem ser adicionadas no relatório, recomenda-se a fim obter o admin das chaves dos inquilinos API. Se os inquilinos secundários múltiplos devem ser gravada junto, ou o inquilino da raiz precisa o acesso a todos os ambientes do desenvolvimento, ou as chaves API de todos os admins secundários do inquilino serão exigidas.

Para obter as chaves API navegue a **Admin > usuários > controlam a chave API**, copiam o

username e fecham-no para os usuários exigidos.



Crie os relatórios personalizados

Antes que você crie o script do pitão que cria o relatório, assegure-se de que o pitão e a semente estejam instalados nele. Então a **semente da** corrida **instala tabula**, tabula é uma biblioteca que segure o formato do relatório automaticamente.

Dois relatórios da amostra são anexados a este guia, o primeiro recolhem simplesmente a informação sobre todas as disposições outputs então a em uma tabela. O segundo usa a mesma informação para criar uns relatórios personalizados com o uso de metadata de BillingID. Este script é explicado em detalhe para usar-se como guia.

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

o datetime é usado para calcular exatamente a data, isto é feito para criar um relatório dos dias os mais recentes X.

o json é usado para ajudar a analisar gramaticalmente dados do json, a saída de atendimentos api.

o SYS é usado para chamadas de sistema.

os pedidos são usados simplificar a fatura de pedidos da Web para os atendimentos API.

tabular é usado para formatar automaticamente a tabela.

o itemgetter é usado como um iterator para classificar uma 2D tabela.

O decimal é usado para arredondar o custo a dois lugares decimais.

```
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()
```

Esta parcela é usada para analisar gramaticalmente o parâmetro da linha de comando do número de dias.

Se não há nenhum parâmetro da linha de comando (sys.argv ==1), a seguir o relatório estará feito por toda a hora.

Se há uma verificação do parâmetro da linha de comando se é um inteiro que seja superior ou igual a 1, se se relata nesse número de dias, se não, retorne um erro.

Se há mais de um retorno do parâmetro um o erro.

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

os departamentos são a lista que guardará a saída final.

os usuários são uma lista de todos os usuários que farão os atendimentos API, se há secundário-inquilinos múltiplos cada usuário seria o admin de um sublocatário diferente.

as senhas são uma lista das chaves dos usuários API, a ordem de usuários e as chaves precisam de ser idênticas para que a chave correta seja usada.

```
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]])
```

para j em xrange(0,len(users)): é para que o laço itere com cada definido pelo utilizador no pedaço precedente do código, isto é o loop principal que segura todos os atendimentos API.

os trabalhos são uma lista provisória que esteja usada para guardar a informação para trabalhos quando for ordenada na lista.

r = requests.get ..... são o primeiro atendimento API, este alista todos os trabalhos, porque mais informação considera ListJobs.

Os resultados são armazenados então no formato do json nos **dados.**

para i em xrange(0,len(data["jobs"])): itera com todos os trabalhos que foram retornados do atendimento precedente API.

O momento para cada trabalho é puxado do json e convertido a um objeto do datetime, a seguir está comparado ao parâmetro da linha de comando incorporado para considerar se está dentro dos limites.

Se é, é esta informação do json que é adicionado à lista dos trabalhos: **identificação, totalCost, estado, nome, horas inicial**. Não toda esta informação é usada, nem é esta toda a informação que pode ser retornada. Os trabalhos da lista mostram toda a informação retornada que pode ser adicionada da mesma forma.

Depois que você itera com todos os trabalhos retornados desse usuário, você transporta-se **para à identificação nos trabalhos:** qual itera com todos os trabalhos que foram tomados depois que você verifica a data de início.

q = requests.get (..... é o segundo atendimento API, este alista toda relativo à informação ao trabalho ID que foi tomado do primeiro atendimento API. Para mais informação veja detalhes de GetJob.

O arquivo do json é armazenado então em **data2.**

O custo, que é armazenado em **id[2]** é arredondado a dois lugares decimais.

para i em xrange(0,len(data2["metadatas"])): itera com todos os metadata associados com o trabalho.

Se há uns metadata chamados **BillingID** então está armazenado na informação do trabalho.

está adicionada uma bandeira usada para determinar se o **BillingID** foi adicionado já à lista dos **departamentos** ou não.

para i em xrange(0,len(departments)): itera com todos os departamentos que foram adicionados.

Se este trabalho é parte de um departamento que já exista, a seguir a contagem do trabalho é iterada por uma, e o custo é adicionado aos custos total para esse departamento.

Se não, uma nova linha é adicionada então aos departamentos com uma contagem do trabalho de 1 e os custos total iguais ao custo deste um trabalho.

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

os departamentos = classificaram (departamentos, key=itemgetter(1)) classificam os departamentos pelo número de trabalhos.

cópia (tabular (departamentos, o ["Department","Number of Jobs", "Total Cost"] do headers=)) imprime uma tabela criada por tabulam com três encabeçamentos.

Informações Relacionadas

- CloudCenter API
- **Suporte Técnico e Documentação - Cisco Systems**