Utilisez les métadonnées au rapport personnalisé avec les API et le python

Contenu

Introduction

Ce document décrit comment utiliser des métadonnées en même temps que le rapport personnalisé API dans un script de python.

Conditions préalables

Conditions requises

Cisco vous recommande de prendre connaissance des rubriques suivantes :

- CloudCenter
- Python

[Composants utilisés](#)

Ce document n'est pas limité à des versions de matériel et de logiciel spécifiques.

Les informations contenues dans ce document ont été créées à partir des périphériques d'un environnement de laboratoire spécifique. Tous les périphériques utilisés dans ce document ont démarré avec une configuration effacée (par défaut). Si votre réseau est opérationnel, assurez-vous que vous comprenez l'effet potentiel de toute commande.

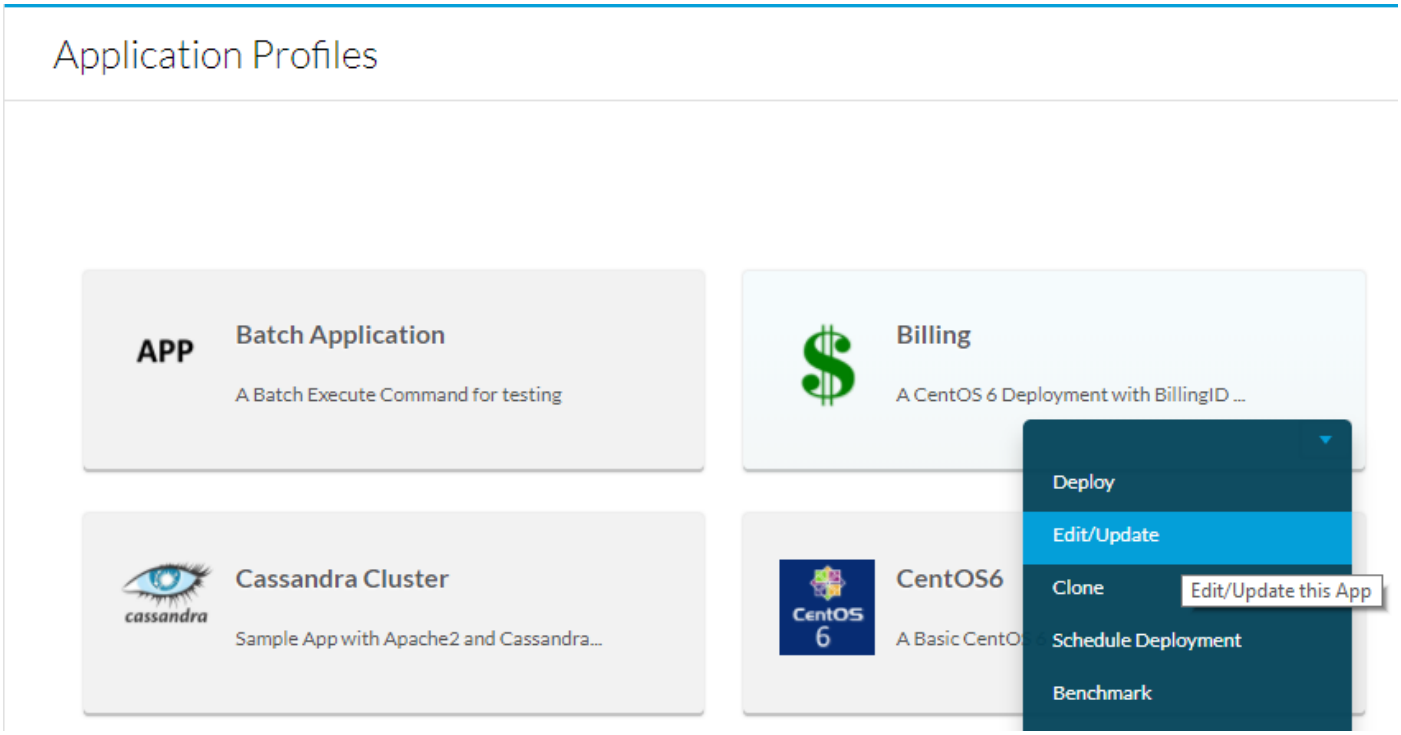
[Informations générales](#)

CloudCenter fournit un certain enregistrement hors de la case, toutefois elle ne permet pas une manière pour des états basés sur des filtres personnalisés. Afin d'employer des API afin de saisir les informations directement de la base de données, en même temps que des métadonnées reliées aux travaux, vous pouvez tenir compte des rapports personnalisés.

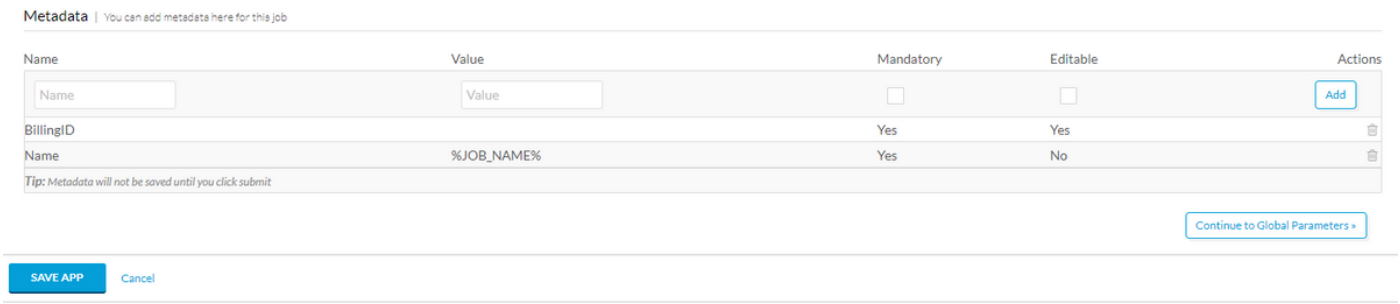
Installez les métadonnées

Des métadonnées doivent être ajoutées sur a par niveau application, tellement chaque application qui doit être dépistée avec l'utilisation du rapport personnalisé devra être modifiée.

Afin de faire ceci, naviguer vers des **profils d'application**, puis sélectionner le déroulant pour que l'app soit édité et le sélectionner alors **éditez/mise à jour** suivant les indications de l'image.



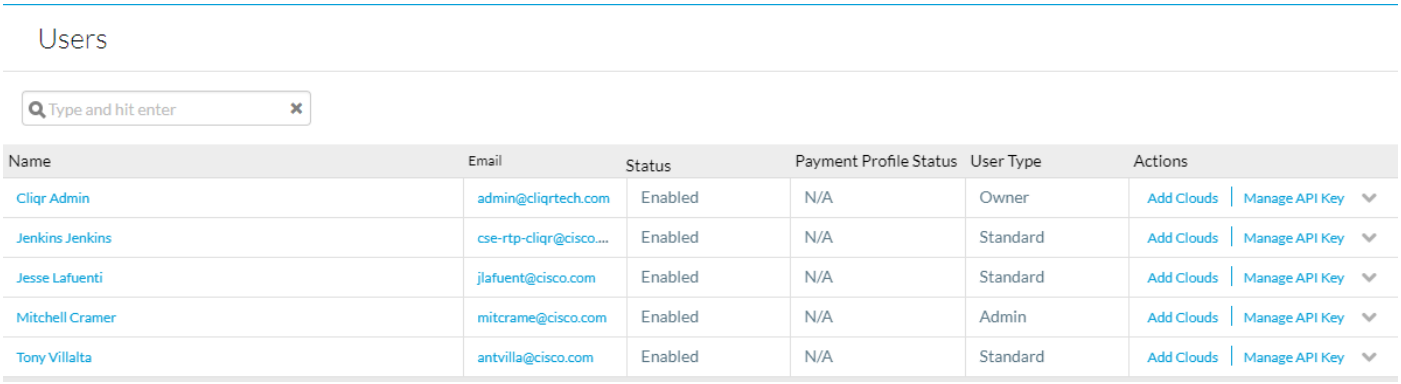
Le défilement au bas des **informations de base** et ajoutent une balise de métadonnées, par exemple **BillingID**, si ce des métadonnées doit être complétées par le suer le rendent obligatoire et editable. Si c'est juste une macro-instruction, alors complétez la valeur par défaut et ne la rendez pas editable. Après que vous complétiez les métadonnées, choisies **ajoutez** alors l'**app de sauvegarde** suivant les indications de l'image.



Clés du rassemblement API

Afin de traiter les appels API, le nom d'utilisateur et les clés API seront exigés. Ces clés fournissent le même niveau d'accès que l'utilisateur, ainsi si tous les déploiements d'utilisateurs doivent être ajoutés dans l'état, il est recommandé afin d'obtenir l'admin des clés des locataires API. Si de sous locataires de multiple doivent être enregistrés ensemble, l'accès des besoins de locataire de racine au tout les environnements de déploiement, ou les clés API de tous les sous admins de locataire seront exigés.

Pour obtenir les clés API naviguez vers l'**admin > les utilisateurs > gèrent la clé API**, copient le nom d'utilisateur et la clé pour les utilisateurs requis.



Créez le rapport personnalisé

Avant que vous créiez le script de python qui crée l'état, assurez-vous que le python et le pépin ont été installés là-dessus. Alors le **pépin de** passage **installent tabulent**, tabulent est une bibliothèque qui manipule formater l'état automatiquement.

Deux états d'échantillon sont reliés à ce guide, le premier collecte simplement des informations au sujet de tous les déploiements les sort alors dans une table. Le deuxième emploie les mêmes informations pour créer un rapport personnalisé avec l'utilisation des métadonnées de BillingID. Ce script est expliqué en détail pour l'utiliser comme guide.

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

la date-heure est utilisée pour calculer exactement la date, ceci est faite pour créer un état des jours X les plus récents.

le json est utilisé pour aider à analyser des données de json, la sortie des appels api.

le système est utilisé pour des appels système.

des demandes est utilisées de simplifier faire des requêtes Web des appels API.

tabulez est utilisé pour formater automatiquement la table.

l'itemgetter est utilisé comme un iterator pour trier une 2D table.

La décimale est utilisée pour arrondir le coût à deux positions décimales.

```
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()
```

Cette partie est utilisée pour analyser le paramètre de la ligne de commande du nombre de jours.

S'il n'y a aucun paramètre de la ligne de commande (sys.argv ==1), alors l'enregistrement sera fait pendant toute l'heure.

S'il y a un contrôle de paramètre de la ligne de commande si c'est un entier qui est supérieur ou égal à 1, s'il est signalé sur ce nombre de jours, sinon, renvoyez une erreur.

S'il y a plus d'un retour de paramètre par erreur.

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

les services est la liste qui tiendra la sortie finale.

les utilisateurs est une liste de tous les utilisateurs qui feront les appels API, s'il y a de plusieurs sous locataire chaque utilisateur serait l'admin d'une sous locataire différente.

les mots de passe est une liste des clés des utilisateurs API, la commande des utilisateurs et les clés doit être identiques pour que la clé correcte soit utilisée.

```
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]])
```

pour j dans xrange(0,len(users)) : est pour que la boucle réitère par chaque défini par l'utilisateur dans le bloc précédent de code, c'est la boucle principale qui traite tous les appels API.

les travaux est une liste provisoire qui sera utilisée pour tenir les informations pour les travaux tandis qu'elles sont assemblées dans la liste.

r = requests.get ..... est le premier appel API, celui-ci répertorie tous les travaux, pour en savoir plus voient ListJobs.

Les résultats sont alors enregistrés dans le format de json dans les **données.**

pour l dans xrange(0,len(data["jobs"])) : réitère par tous les travaux qui ont été retournés de l'appel précédent API.

Le moment pour chaque travail est tiré du json et converti en objet date-heure, puis il est comparé au paramètre de la ligne de commande entré pour voir s'il est dans des limites.

S'il est, c'est ces informations du json qui est ajouté à la liste des travaux : **id, totalCost, état, nom, heure de début**. Pas toutes ces informations sont utilisées, ni sont ces toutes les informations qui peuvent être renvoyées. Les travaux de liste affiche toutes les informations renvoyées qui peuvent être ajoutées de la même manière.

Après que vous réitériez par tous les travaux retournés de cet utilisateur, vous vous déplacez à **pour l'id dans les travaux :** ce qui réitère par tous les travaux qui ont été pris après que vous vérifiiez la date de début.

q = requests.get (..... est le deuxième appel API, celui-ci répertorie tout relatif à l'information à l'identification des tâches qui a été prise du premier appel API. Le pour en savoir plus voient des détails de GetJob.

Le fichier de json est alors enregistré dans **data2.**

Le coût, qui est enregistré dans **id[2]** est arrondi à deux positions décimales.

pour l dans xrange(0,len(data2["metadatas"])) : réitère par toutes les métadonnées associées avec le travail.

S'il y a des métadonnées appelées **BillingID** puis il est enregistré dans les informations du travail.

ajouté est un indicateur utilisé pour déterminer si le **BillingID** déjà a été ajouté à la liste de **services** ou pas.

pour l dans xrange(0,len(departments)) : réitère par tous les services qui ont été ajoutés.

Si ce travail fait partie d'un service qui existe déjà, alors le compte du travail est réitéré par on, et le coût est ajouté au coût total pour ce service.

Sinon, alors une nouvelle ligne est ajoutée aux services avec un compte du travail de 1 et le coût total égal au coût de ce un travail.

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

les services = trié (des services, key=itemgetter(1)) trie les services par le nombre de travaux.

copie (tabulez (services, headers= [« service », « nombre des travaux », de « coût total »])) imprime une table créée par tabulent avec trois en-têtes.

Informations connexes

- CloudCenter API
- Support et documentation techniques - Cisco Systems