Utilice los meta datos al informe personalizado con los API y Python

Contenido

Introducción

Este documento describe cómo utilizar los meta datos conjuntamente con el informe personalizado API para dentro de un script del pitón.

Prerequisites

Requisitos

Cisco recomienda que tenga conocimiento sobre estos temas:

- CloudCenter
- Python

Componentes Utilizados

Este documento no tiene restricciones específicas en cuanto a versiones de software y de hardware.

La información que contiene este documento se creó a partir de los dispositivos en un ambiente de laboratorio específico. Todos los dispositivos que se utilizan en este documento se pusieron en funcionamiento con una configuración verificada (predeterminada). Si la red está funcionando, asegúrese de haber comprendido el impacto que puede tener cualquier comando.

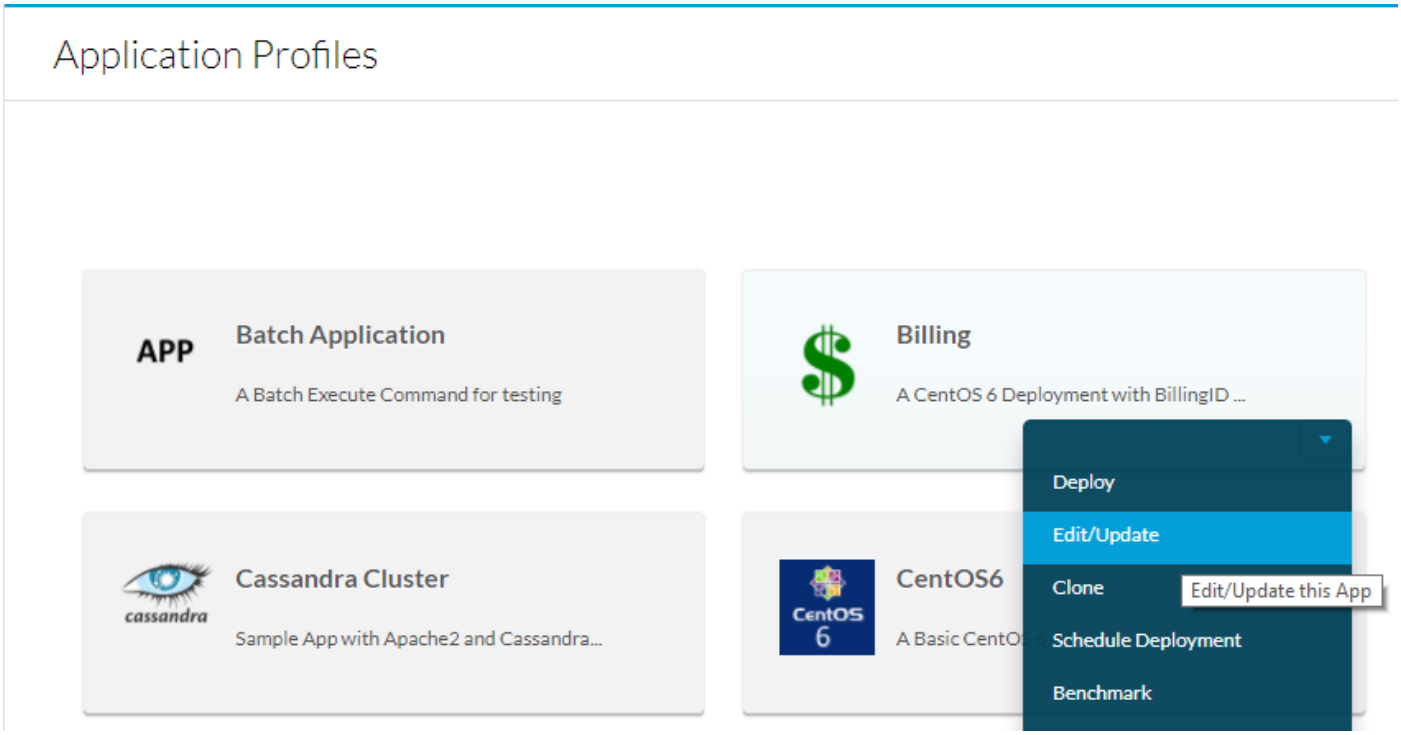
Antecedentes

CloudCenter proporciona un poco de cuadro de los de la información, no obstante no permite una manera para los informes basados sobre los filtros de encargo. Para utilizar los API para asir la información directamente de la base de datos, conjuntamente con los meta datos asociada a los trabajos, usted puede tener en cuenta los informes personalizados.

Configure los meta datos

Los meta datos se deben agregar en a por el nivel de aplicación, tan cada aplicación que necesite ser seguida con el uso del informe personalizado tenga que ser modificada.

Para hacer esto, navegar a los **perfiles de aplicación**, después seleccionar el dropdown para que el App sea editado y después seleccionarlo **edite/actualización** tal y como se muestra en de la imagen.

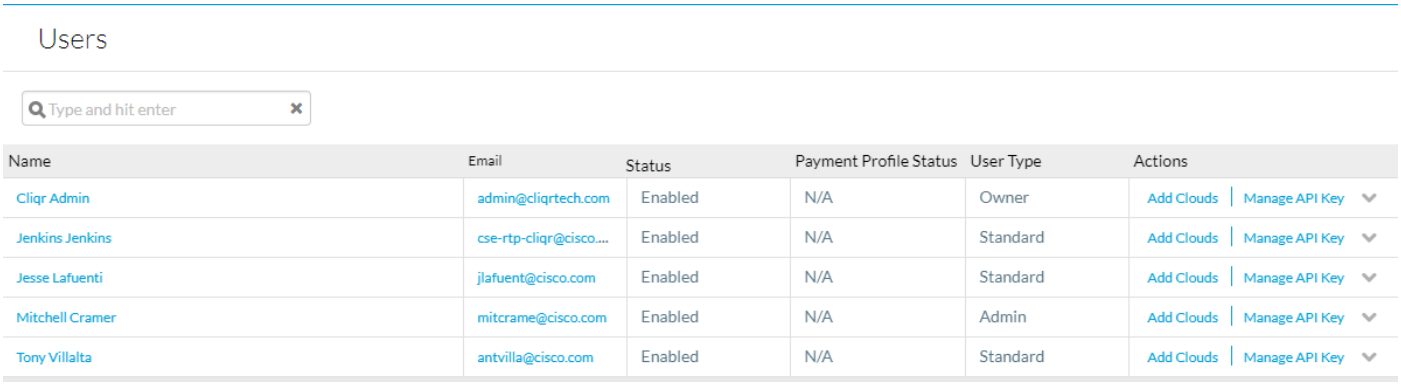


Navegue a la parte inferior de la **información básica** y agregue los meta datos marcan con etiqueta, por ejemplo **BillingID**, si este se van los meta datos a ser completados por el suer lo hace obligatorio y editable. Si es apenas una macro, después complete el valor predeterminado y no lo haga editable. Después de que usted complete los meta datos, selectos **agregue** entonces el **App de la salvaguardia** tal y como se muestra en de la imagen.

Recolecte las claves API

Para procesar las llamadas API, el nombre de usuario y las claves API serán requeridos. Estas claves proporcionan el mismo nivel de acceso que el usuario, así que si se van todas las implementaciones de los usuarios a ser agregadas en el informe, se recomienda para conseguir el admin de las claves de los arrendatarios API. Si se van los arrendatarios sub múltiples a ser registrados juntos, el acceso de las necesidades del arrendatario de la raíz a todo el los entornos del despliegue, o las claves API de todos los admins sub del arrendatario serán requeridos.

Para conseguir las claves API navegue al **Admin > Users > manejan la clave API**, copian el nombre de usuario y lo cierran para los usuarios requeridos.



Cree el informe personalizado

Antes de que usted cree el script del pitón que crea el informe, asegúrese de que el pitón y la pipa hayan estado instalados en él. Entonces la **pipa del** funcionamiento **instala tabula**, tabula es una biblioteca que maneja formatar el informe automáticamente.

Dos informes de la muestra se asocian a esta guía, el primer recogen simplemente la información sobre todas las implementaciones entonces la hacen salir en una tabla. El segundo utiliza la misma información para crear un informe personalizado con el uso de los meta datos de BillingID. Este script se explica detalladamente para utilizar como guía.

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

la fecha y hora se utiliza para calcular exactamente la fecha, esto se hace para crear un informe de los días más recientes X.

el json se utiliza para ayudar a analizar los datos del json, la salida de las llamadas api.

el sys se utiliza para las llamadas del sistema.

se utilizan las **peticiones** de simplificar la elaboración de las solicitudes web las llamadas API.

tabule se utiliza para formatar automáticamente la tabla.

el itemgetter se utiliza como iterator para clasificar una 2.a tabla.

El decimal se utiliza para redondear el coste a dos lugares decimales.

```
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 porción se utiliza para analizar el parámetro de la línea de comando del número de días.

Si no hay parámetros de la línea de comando (sys.argv ==1), después la información será hecha por toda la hora.

Si hay un control del parámetro de la línea de comando si es un número entero que es mayor o igual 1, si está señalado sobre ese número de días, si no, vuelva un error.

Si hay más de una vuelta del parámetro al error.

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

los departamentos son la lista que llevará a cabo la salida final.

los usuarios son una lista de todos los usuarios que hagan las llamadas API, si hay subarrendatarios múltiples cada usuario sería el admin de un diverso subarrendatario.

las contraseñas son una lista de las claves de los usuarios API, la orden de los usuarios y las claves necesitan ser idénticas para que la clave correcta sea utilizada.

```
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 en xrange(0,len(users)): está para que el loop itere con cada definido por el usario en el pedazo anterior del código, esto es el Loop principal que maneja todas las llamadas API.

los trabajos son una lista temporal que será utilizada para llevar a cabo la información para los trabajos mientras que se clasifica en la lista.

r = requests.get ..... es la primera llamada API, ésta enumera todos los trabajos, porque más información considera ListJobs.

Los resultados entonces se salvan en el formato del json en los **datos.**

para i en xrange(0,len(data["jobs"])): itera con todos los trabajos que fueron vueltos de la llamada anterior API.

La época para cada trabajo se tira del json y se convierte a un objeto de la fecha y hora, después se compara al parámetro de la línea de comando ingresado para considerar si está dentro de los límites.

Si es, es esta información del json que se añade al final del fichero a la lista de los trabajos: **identificación, totalCost, estatus, nombre, hora de inicio**. No toda esta información se utiliza, ni es esta toda la información que puede ser devuelta. Los trabajos de la lista muestran toda la información devuelta que se pueda agregar de la misma manera.

Después de que usted itere con todos los trabajos vueltos de ese usuario, usted se traslada **para a la identificación en los trabajos:** cuál itera con todos los trabajos que fueron tomados después de que usted marque la Fecha de inicio.

q = requests.get (..... es la segunda llamada API, ésta enumera todo relacionado con la información al trabajo ID que fue tomado de la primera llamada API. Para más información vea los detalles de GetJob.

El archivo del json entonces se salva en **data2.**

El coste, que se salva en **id[2]** se redondea a dos lugares decimales.

para i en xrange(0,len(data2["metadatas"])): itera con todos los meta datos asociados al trabajo.

Si hay meta datos llamados **BillingID** entonces se salva en la información del trabajo.

se agrega un indicador usado para determinar si el **BillingID** se ha agregado ya a la lista de los **departamentos** o no.

para i en xrange(0,len(departments)): itera con todos los departamentos se han agregado que.

Si este trabajo es parte de al departamento que existe ya, después la cuenta del trabajo es iterada por una, y el coste se agrega al costo total para ese departamento.

Si no, entonces una línea nueva se añade al final del fichero a los departamentos con una cuenta del trabajo de 1 y el costo total igual al coste de este un trabajo.

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

los departamentos = clasificaron (los departamentos, key=itemgetter(1)) clasifican los departamentos por el número de trabajos.

impresión (tabule (los departamentos, el ["Department","Number of Jobs", "Total Cost"] del headers=)) imprime una tabla creada por tabulan con tres encabezados.

Información Relacionada

- CloudCenter API
- **Soporte Técnico y Documentación - Cisco Systems**