Utilize Metadata to Custom Report with APIs and Python

Contents

Introduction

This document describes how to use metadata in conjunction with APIs in order to custom report within a python script.

Prerequisites

Requirements

Cisco recommends that you have knowledge of these topics:

- CloudCenter
- Python

Components Used

This document is not restricted to specific software and hardware versions.

The information in this document was created from the devices in a specific lab environment. All of the devices used in this document started with a cleared (default) configuration. If your network is live, make sure that you understand the potential impact of any command.

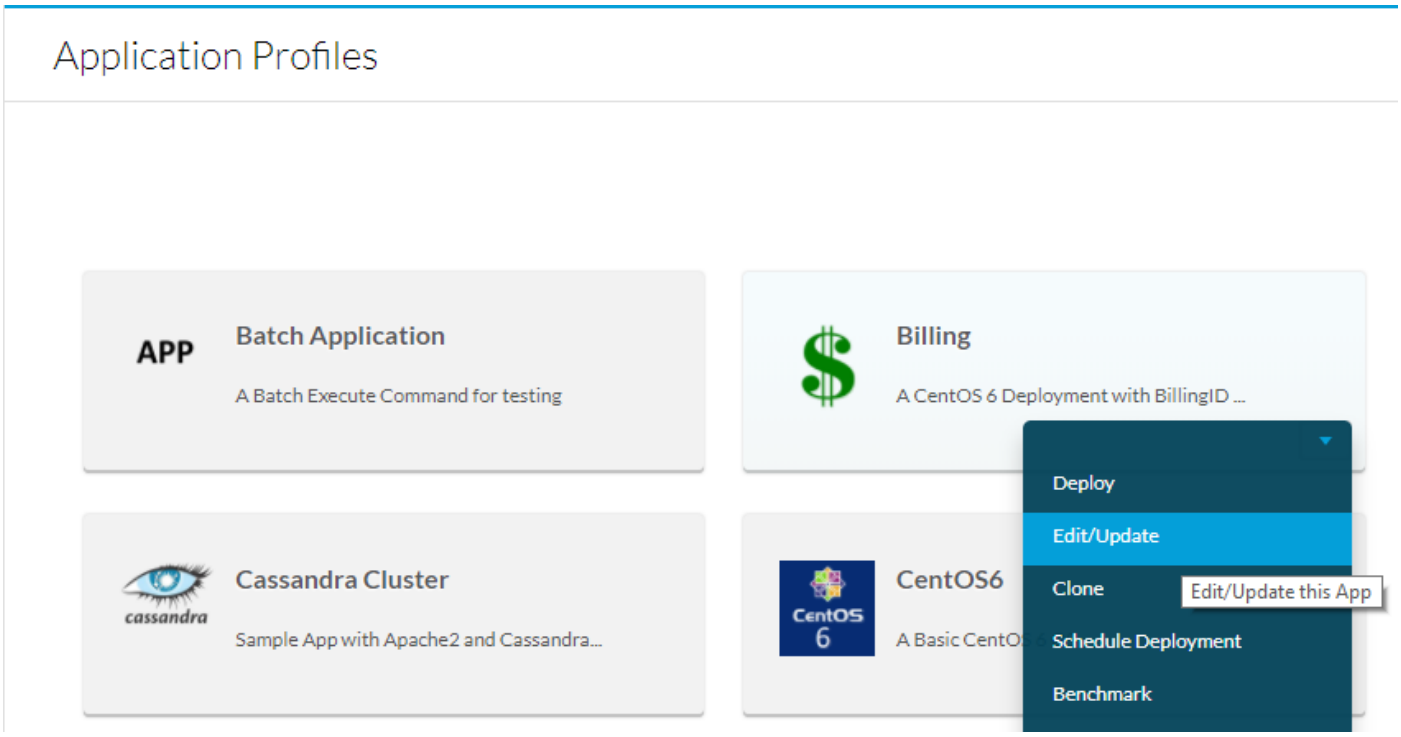
Background Information

CloudCenter provides some reporting out of the box, however it does not allow a way for reports based upon custom filters. In order to use APIs in order to grab the information directly from the database, in conjunction with metadata attached to the jobs, you can allow for custom reports.

Set up the Metadata

Metadata must be added on a per application level, so every application that needs to be tracked with the use of the custom report will have to be modified.

In order to do this, navigate to **Application Profiles**, then select the dropdown for the App to be edited and then select **Edit/Update** as shown in the image.

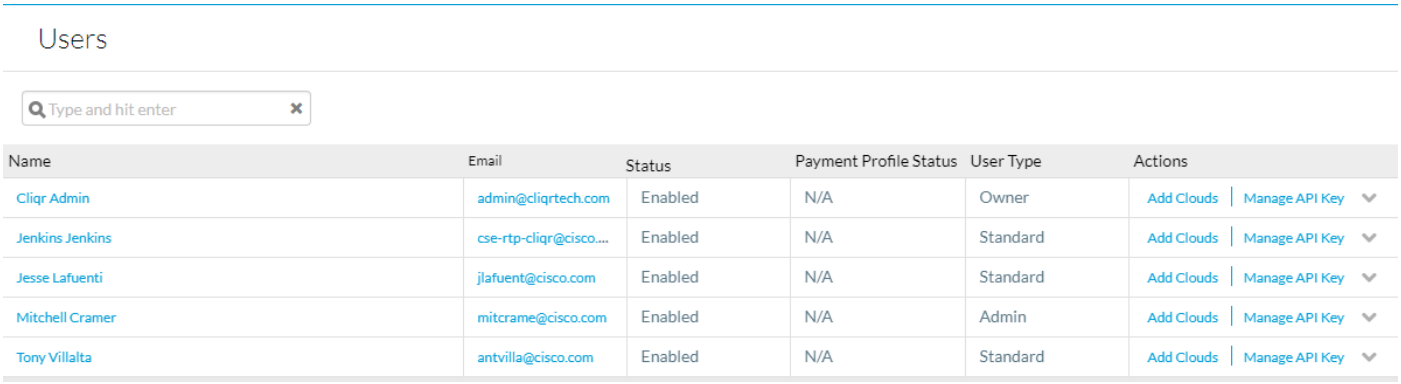


Scroll to the bottom of **Basic Information** and add a Metadata tag, for example **BillingID**, if this metadata is to be filled out by the suer make it both mandatory and editable. If it is just a macro, then fill in the default value and do not make it editable. After you fill out the metadata, select **Add** then **Save App** as shown in the image.

Gather API Keys

In order to process the API calls, username and API keys will be required. These keys provide the same level of access as the user, so if all users deployments are to be added in the report, it is recommended in order to get the admin of the tenants API keys. If multiple sub tenants are to be recorded together, either the root tenant needs access to all the deployment environments, or the API keys of all sub tenant admins will be required.

To get the API keys navigate to **Admin > Users > Manage API Key**, copy the username and key for the users required.



Create the Custom Report

Before you create the python script that creates the report, ensure that python and pip have been installed on it. Then run **pip install tabulate**, tabulate is a library that handles formatting the report automatically.

Two sample reports are attached to this guide, the first simply collects information about all deployments then outputs it in a table. The second uses the same information to create a custom report with the use of BillingID metadata. This script is explained in detail to use as a guide.

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

datetime is used to accurately calculate the date, this is done to create a report of the most recent X days.

json is used to help parse json data, the output of api calls.

sys is used for system calls.

requests is used to simplify making web requests for the API calls.

tabulate is used to automatically format the table.

itemgetter is used as an iterator to sort a 2D table.

Decimal is used to round cost to two decimal places.

```
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()
```

This portion is used to parse the command line parameter of number of days.

If there are no command line parameters (sys.argv ==1), then reporting will be done for all time.

If there is one command line parameter check if it is an integer that is greater than or equal to 1, if it is reported on that number of days, if not, return an error.

If there is more than one parameter return an error.

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

departments is the list that will hold the final output.

users is a list of all users who will make the API calls, if there are multiple sub-tenants each user would be the admin of a different subtenant.

passwords is a list of the users API keys, the order of users and keys needs to be identical for the correct key to be used.

```
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]])
```

for j in xrange(0,len(users)): is for loop to iterate through every user defined in the previous code chunk, this is the main loop that handles all API calls.

jobs is a temporary list that will be used to hold the information for jobs while it is collated into the list.

r = requests.get..... is the first API call, this one lists all jobs, for more information see List Jobs.

The results are then stored in json format in **data.**

for i in xrange(0,len(data["jobs"])): iterates through all the jobs that were returned from the previous API call.

The time for each job is pulled from the json and converted to a datetime object, then it is compared to the command line parameter entered to see if it is within bounds.

If it is, it is this information from the json that is appended to the jobs list: **id, totalCost, status, name, start time**. Not all of this information is used, nor is this all the information that can be returned. List Jobs shows all information returned that can be added in the same way.

After you iterate through all the jobs returned from that user, you move to **for id in jobs:** which iterates through all the jobs that were taken after you check the start date.

q = requests.get(..... is the second API call, this one lists all information related to the job ID that was taken from the first API call. For more information see Get Job Details.

The json file is then stored in **data2.**

The cost, which is stored in **id[2]** is rounded to two decimal places.

for i in xrange(0,len(data2["metadatas"])): iterates through all the metadata associated with the job.

If there is metadata called **BillingID** then it is stored in the job information.

added is a flag used to determine if the **BillingID** has already been added to the **departments** list or not.

for i in xrange(0,len(departments)): iterates through all the departments that have been added.

If this job is part of a department that already exists, then the job count is iterated by one, and the cost is added to the total cost for that department.

If not, then a new line is appended to departments with a job count of 1 and total cost equal to the cost of this one job.

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

departments = sorted(departments, key=itemgetter(1)) sorts the departments by the Number of Jobs.

print(tabulate(departments,headers=['Department','Number of Jobs', 'Total Cost'])) prints a table created by tabulate with three headers.

Related Information

- CloudCenter API
- **Technical Support & Documentation - Cisco Systems**