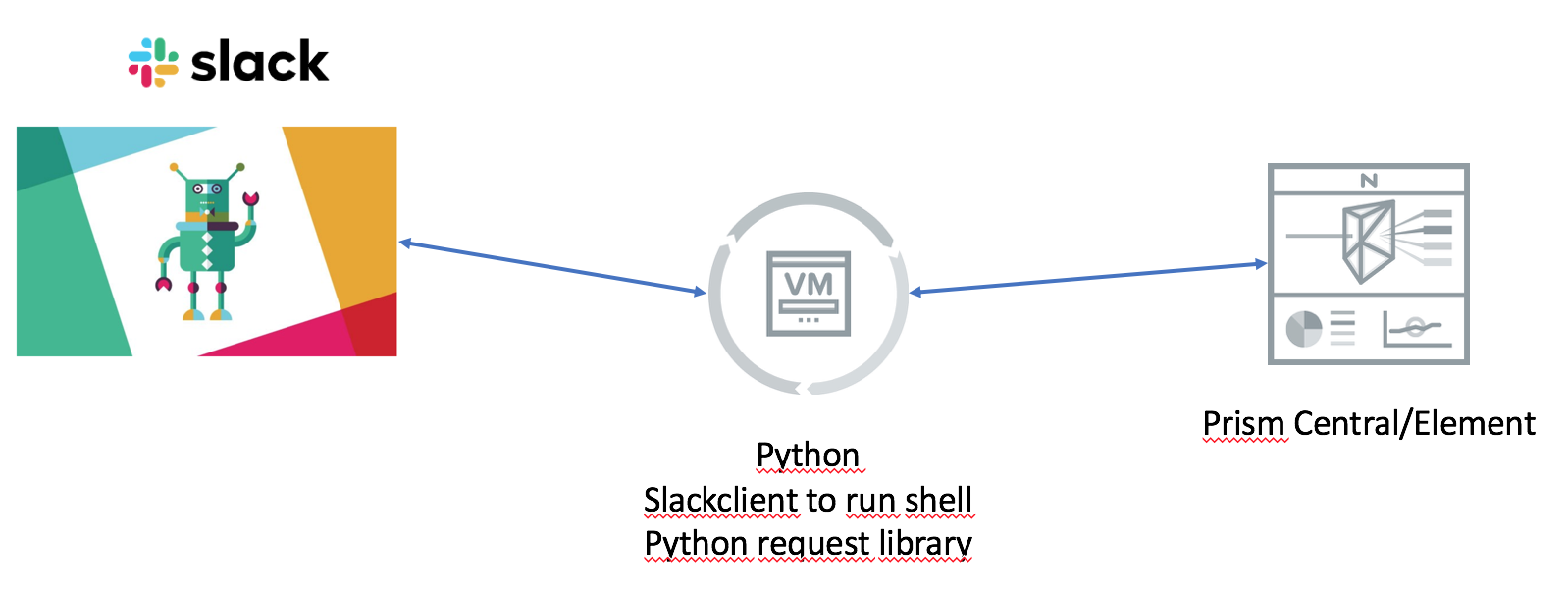
High level view:



**Step 1:** have a linux VM connected to internet with python installed

Need a python installation (could already be installed on your linux VM):

example: <https://tecadmin.net/install-python-3-7-on-centos/>

libraries and several updates are installed on my linux VM (history of the pip installation)

pip install --upgrade pip

pip install json

pip install simplejson

pip install jsonlib

pip install python3

yum install -y python36u python36u-libs python36u-devel python36u-pip

yum install python3-pip

pip3.6 install --upgrade pip

yum install -y ntp ntpdate unzip stress nodejs python-pip s3cmd awscli

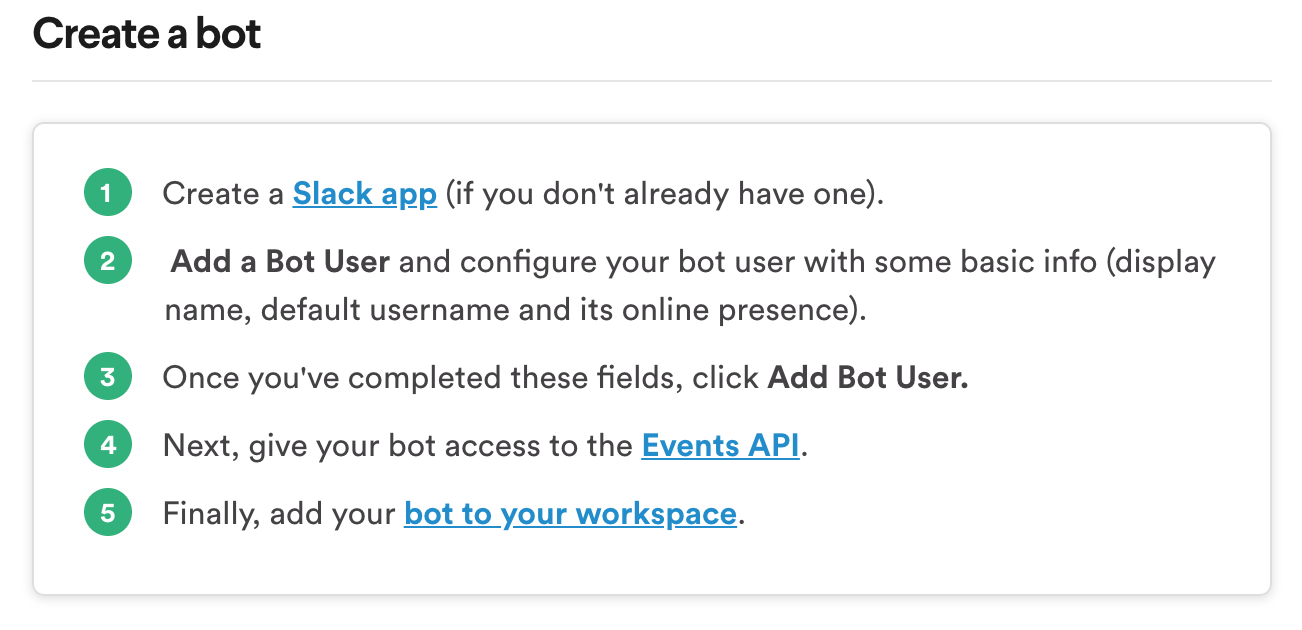
pip install boto3

pip3 install -e .

**Step 2:** Create a Slackbot and obtain your token

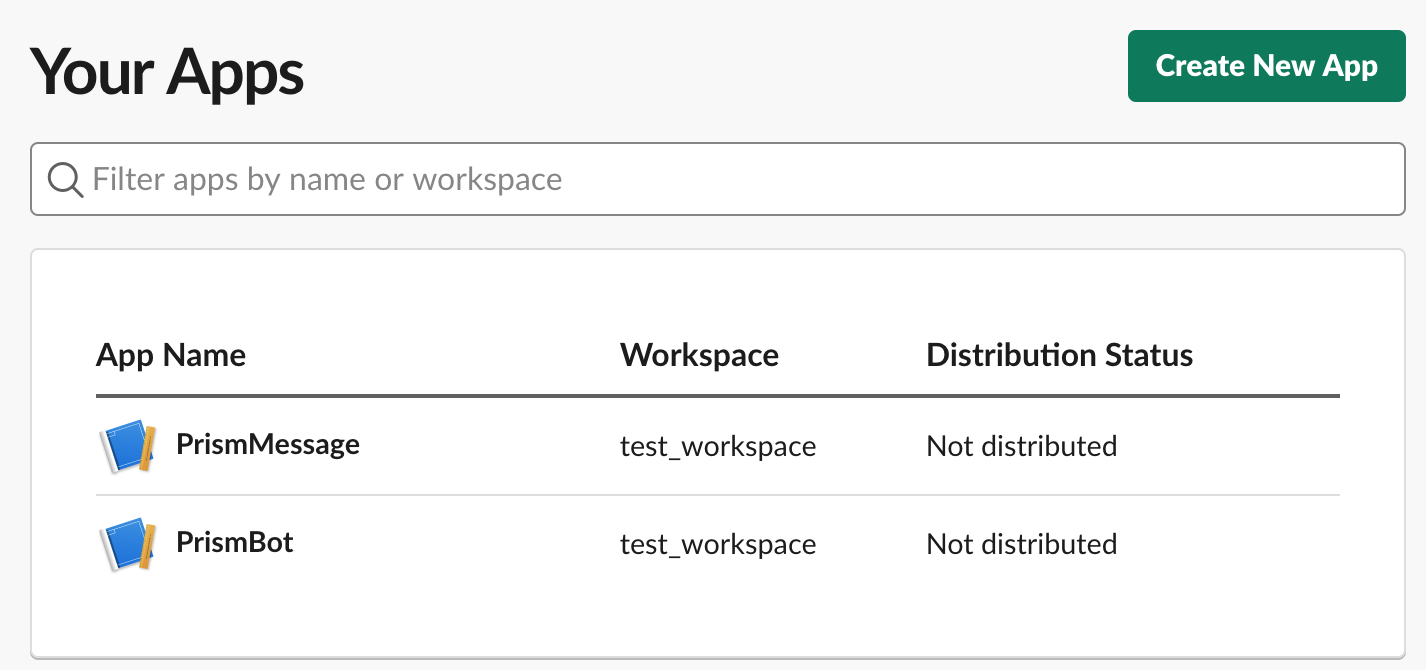
create a bot on your slack workspace:

<https://slack.com/intl/en-it/help/articles/115005265703-create-a-bot-for-your-workspace>

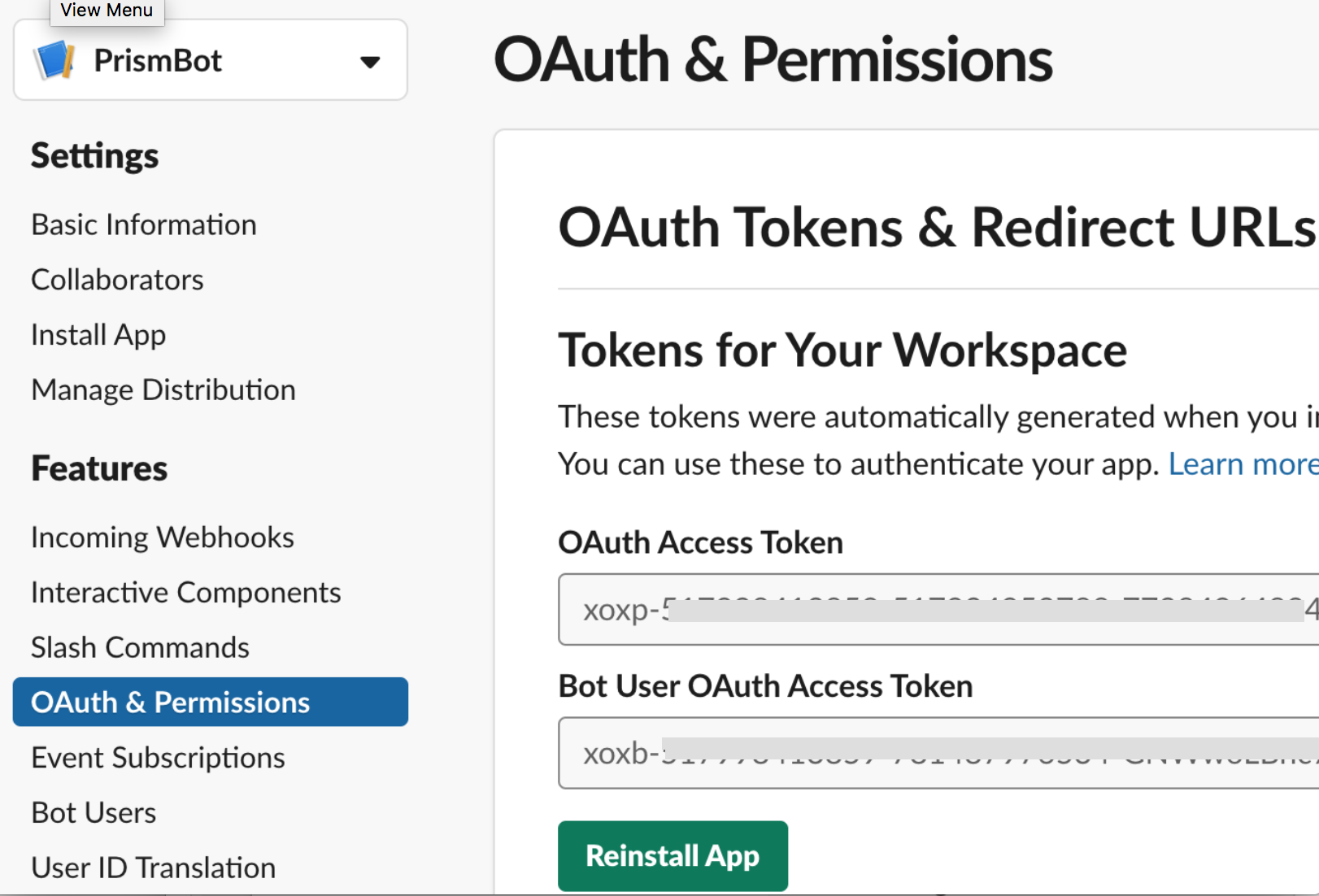


You can find your applications here:

<https://api.slack.com/>



All you need is to get your tokens to configure the application to talk with your linux client…



**Step 3:** Install slack client and bot:

<https://github.com/agasy18/slack-remote-terminal>

this application, python based, is able to run any OS command on the linux client received from slak bot.

At this point you should be done with the slack bot that should be able to communicate with your workspace and execute shell commands on your linux VM.

If it does not work, you need to troubleshoot since next steps depends from this pre-work.

In case of certificate error on the slack client, I found this workaround:

<https://github.com/slackapi/python-slackclient/issues/334>

in particular:

1. Downgrading the websocket-client library to 0.47.0

2. Or, download the certificate (wget https://www.tbs-certificats.com/issuerdata/DigiCertGlobalRootCA.crt), then set the environment variable export WEBSOCKET\_CLIENT\_CA\_BUNDLE=DigiCertGlobalRootCA.crt

In my implementation I used to export the variable.

I also made the bot running with nohup, so my screen looks like:

*# export WEBSOCKET\_CLIENT\_CA\_BUNDLE=DigiCertGlobalRootCA.crt*

*# nohup python bot.py &*

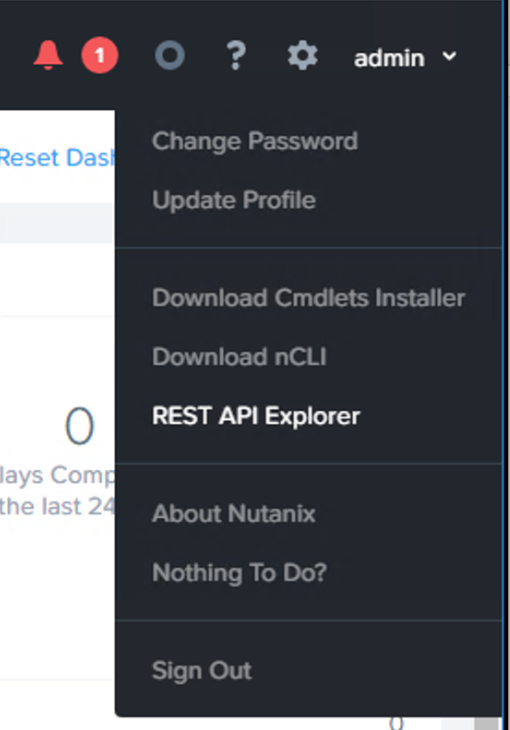
I put directly my slack token in the bot.py program looks like:

#SLACK\_BOT\_TOKEN = os.environ['SLACK\_BOT\_TOKEN'] or config['SLACK\_BOT\_TOKEN']

SLACK\_BOT\_TOKEN = 'xoxb-1234567890123-0123456789012-GNWcrEXntnx4evernoVMware'

**Step 4:**

We have several examples on how to use API with Nutanix, common method to learn is the use of API explorer in all Nutanix PRISM interface (both Prism Element and Prism Central):



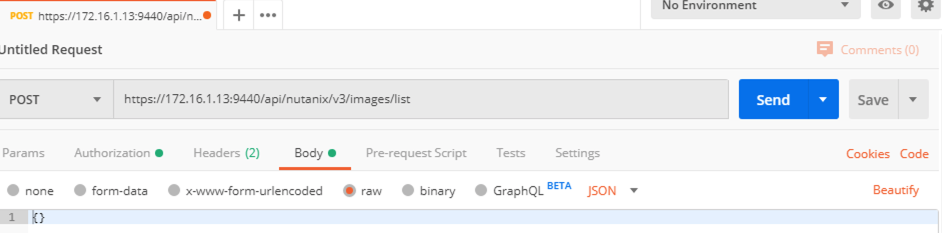
https://<Your PRISM IP>/api/nutanix/v3/api\_explorer/index.html

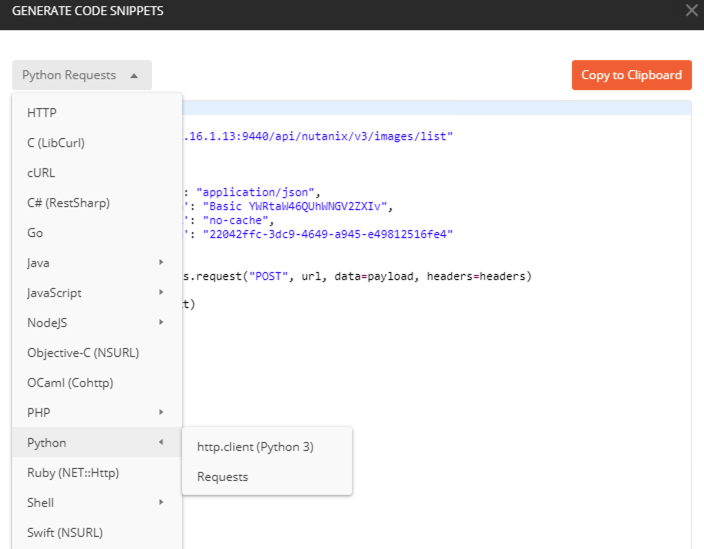
CALM API are here:

https://<Your PRISM IS>:9440/api/nutanix/v3/api\_explorer/index.html?url=/apps/api/static/json/styx-swagger.json

Another very good tool is Postman application to understand API usage and getting also code to start Python doces.

A very important thing Postman helps with, is generation of an authorization token to make the calls, avoiding putting in clear username and password in the python scripts…







The Authorization field has been used in the python scripts instead of passing admin and password in clear text.

A good learning example for the API is here:

<http://myvirtualcloud.net/how-to-use-python-with-nutanix-rest-api-part-1/>

Nutanix version 3 API are more powerful and complete of the previous versions, also the standard list of entities requires passing a json payload to the request.

An example here of code is the following, for listing all VMs in the cluster:

|  |
| --- |
| *import requests*  *import json*  *import pprint*  *# Nutanix API URL*  *cluster\_ip\_address = '172.16.1.13'*  *API\_AHV = ':9440/api/nutanix/v3/'*  *request = 'vms/list'*  *#here is a compose of the url to ask for the info*  *base\_url = "https://" + cluster\_ip\_address + API\_AHV + request*  *#some requests do not need any json payload to pass, while others are more structured and needs more info as input*  *#payload = "{}"*  *payload = json.dumps(*  *{"kind": "vm"*  *}, sort\_keys=True)*  *#here the usage of Postman is really helpful since no user and password are exposed in a text format*  *headers = {*  *'Content-Type': "application/json",*  *'Authorization': "Basic YWRtaW46QUhWNGV2ZXIv",*  *}*  *#suppress warning messages due to unsecure request*  *requests.packages.urllib3.disable\_warnings()*  *response = requests.request("POST", base\_url, data=payload, headers=headers, verify=False)*  *answer = json.loads(response.text)*  *lenght = len(answer["entities"])*  *#print full json answer to see the structure of it and filter the appropriate data*  *#pprint.pprint(answer)*  *#pint just entities*  *#print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")*  *#pprint.pprint(answer["entities"])*  *#print just some meaning info for humans, in this case, name of all VMs registered on the cluster*  *print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")*  *i = 0*  *while i < lenght:*  *pprint.pprint(answer["entities"][i]["spec"]["name"])*  *i += 1* |

The only parameters to take care are the highlighted red one, if you just need listing Calm blueprint you just need to change the following 3 lines, last one since json result does not have spec field, or you need a different field than the before example:

request = 'blueprints/list'

payload = json.dumps(

{"kind": "blueprint"

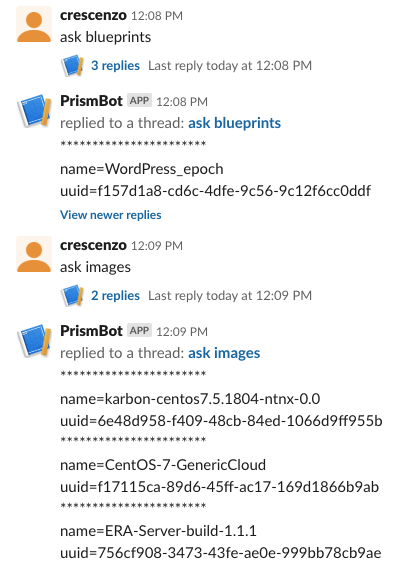
}, sort\_keys=True)

pprint.pprint(answer["entities"][i]["metadata"]["name"])

Making it more general, asking PrismCentral to give list of: VMs, Images, blueprints as input in the python file:

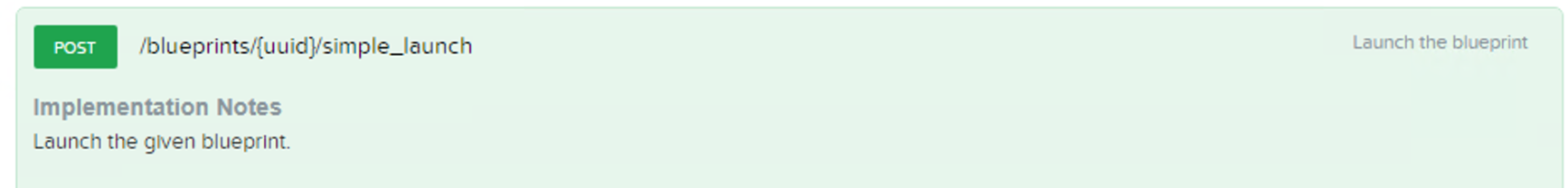
|  |
| --- |
| *#!/usr/bin/python3.6*  *import requests*  *import json*  *import pprint*  *import sys*  *# Nutanix API URL*  *cluster\_ip\_address = '172.16.1.13'*  *API\_AHV = ':9440/api/nutanix/v3/'*  *payload = "{}"*  *#next payload for later usage*  *#payload = json.dumps(*  *#{"kind": "image"*  *#}, sort\_keys=True)*  *headers = {*  *'Content-Type': "application/json",*  *'Authorization': "Basic YWRtaW46QUhWNGV2ZXIv",*  *}*  *def \_suppress\_security():*  *# suppress the security warnings*  *requests.packages.urllib3.disable\_warnings()*  *def \_http\_request(base\_url, json\_data, element):*  *\_suppress\_security()*  *response = requests.request("POST", base\_url, data=payload, headers=headers, verify=False)*  *answer = json.loads(response.text)*  *lenght = len(answer["entities"])*  *#print full json answer*  *#pprint.pprint(answer)*  *#print just some meaning info for humans*  *#print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")*  *i = 0*  *while i < lenght:*  *name = answer["entities"][i][element]["name"]*  *uuid = answer["entities"][i]["metadata"]["uuid"]*  *print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")*  *print('name='+name)*  *print('uuid='+uuid)*  *i += 1*  *return 0*  *if len(sys.argv) == 1 :*  *print("implemented today is images, vms, blueprints")*  *else :*  *if len(sys.argv) > 2 :*  *print("too many arguments, bye bye")*  *else :*  *ask = str(sys.argv[1])*  *if ask == "images" :*  *element = 'spec'*  *request = 'images/list'*  *if ask == "vms" :*  *element = 'spec'*  *request = 'vms/list'*  *if ask == "blueprints" :*  *element = 'metadata'*  *request = 'blueprints/list'*  *base\_url = "https://" + cluster\_ip\_address + API\_AHV + request*  *\_http\_request(base\_url, payload, element)* |

Putting in the PATH the location where python scripts resides, give also a nice interaction with Slack, python script is named ask:



Moving on with another request, let’s try to launch a Blueprint via API.

In this case we use the simple\_launch API to facilitate the job.



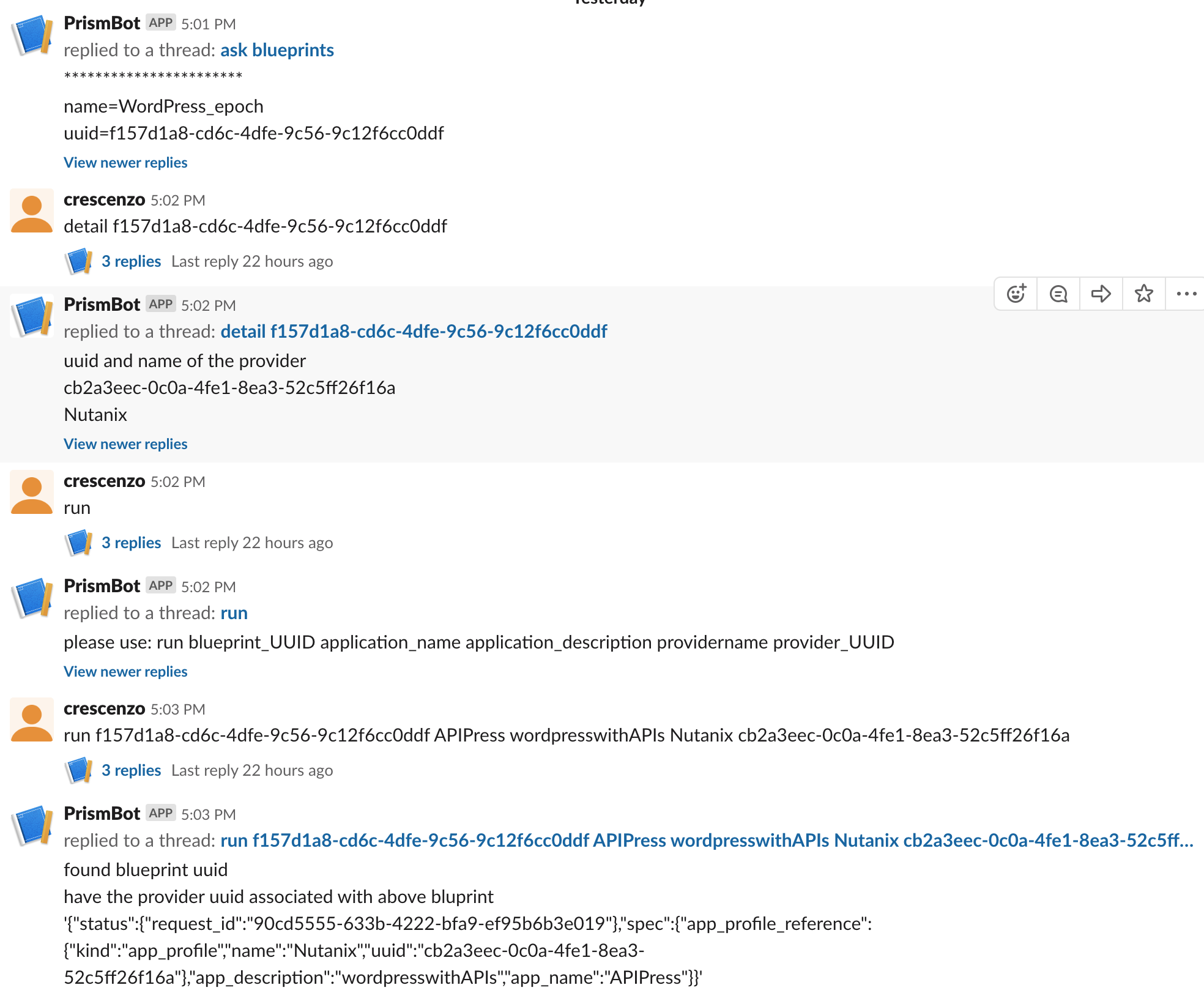
It is required to have the blueprint id and the application profile provider name and uuid.

To obtain those parameters, the following is the python to GET details of the Blueprint uuid:

|  |
| --- |
| *#!/usr/bin/python3.6*  *import requests*  *import json*  *import pprint*  *import sys*  *# Nutanix API URL*  *cluster\_ip\_address = '172.16.1.13'*  *API\_AHV = ':9440/api/nutanix/v3/'*  *payload = "{}"*  *headers = {*  *'Content-Type': "application/json",*  *'Authorization': "Basic YWRtaW46QUhWNGV2ZXIv",*  *}*  *def \_suppress\_security():*  *# suppress the security warnings*  *requests.packages.urllib3.disable\_warnings()*  *def \_http\_request(base\_url, json\_data):*  *\_suppress\_security()*  *response = requests.request("GET", base\_url, data=payload, headers=headers, verify=False)*  *answer = json.loads(response.text)*  *#print just some meaning info for humans*  *uuid\_out = []*  *for v1 in answer["status"]["resources"]["app\_profile\_list"]:*  *if "uuid" in v1:*  *uuid\_out.append(v1["uuid"])*  *uuid\_out.append(v1["name"])*  *return uuid\_out*  *if len(sys.argv) < 2 :*  *print("please give blueprint uuid")*  *else :*  *if len(sys.argv) > 2 :*  *print("too many arguments, bye bye")*  *else :*  *ask = str(sys.argv[1])*  *request = 'blueprints/'*  *base\_url = "https://" + cluster\_ip\_address + API\_AHV + request + ask*  *uuid\_out=\_http\_request(base\_url, payload)*  *print("uuid and name of the provider")*  *for i in uuid\_out:*  *print(i)* |

The most difficult component is here to filter the massive json answer coming from the cluster (BP contain a lot of informations).

Having the details of the Blueprint, in particular uuid of the providers registered to the Blueprint itself, we can send a generic command to run known BluePrint:



The python script is the following:

|  |
| --- |
| *#!/usr/bin/python3.6*  *import requests*  *import json*  *import pprint*  *import sys*  *# Nutanix API URL*  *cluster\_ip\_address = '172.16.1.13'*  *API\_AHV = ':9440/api/nutanix/v3/'*  *payload = "{}"*  *def \_payload2(providername, provideruuid, appname, appdesc):*  *#create payload to launch Blueprint assuming blueprint application profile name and uuid exists and are associated with the given BP uuid*  *payload2 = json.dumps(*  *{*  *"spec": {*  *"app\_profile\_reference": {*  *"kind": "app\_profile",*  *"name": providername,*  *"uuid": provideruuid*  *},*  *"app\_name": appname,*  *"app\_description": appdesc*  *}*  *}, sort\_keys=True)*  *return payload2*  *headers = {*  *'Content-Type': "application/json",*  *'Authorization': "Basic YWRtaW46QUhWNGV2ZXIv",*  *}*  *def \_suppress\_security():*  *# suppress the security warnings*  *requests.packages.urllib3.disable\_warnings()*  *def \_http\_request(base\_url, json\_data, element):*  *\_suppress\_security()*  *response = requests.request("POST", base\_url, data=payload, headers=headers, verify=False)*  *answer = json.loads(response.text)*  *lenght = len(answer["entities"])*  *#print full json answer*  *# pprint.pprint(answer)*  *#print just some meaning info for humans*  *#print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")*  *i = 0*  *uuid\_out = []*  *while i < lenght:*  *name = answer["entities"][i][element]["name"]*  *uuid = answer["entities"][i]["metadata"]["uuid"]*  *uuid\_out.append(uuid)*  *i += 1*  *return uuid\_out*  *def \_http\_request\_2(base\_url, json\_data):*  *\_suppress\_security()*  *response = requests.request("GET", base\_url, data=payload, headers=headers, verify=False)*  *answer = json.loads(response.text)*  *uuid\_out = []*  *for v1 in answer["status"]["resources"]["app\_profile\_list"]:*  *if "uuid" in v1:*  *# pprint.pprint(v1["uuid"])*  *# pprint.pprint(v1["name"])*  *uuid\_out.append(v1["uuid"])*  *uuid\_out.append(v1["name"])*  *# import pdb;pdb.set\_trace()*  *return uuid\_out*  *if len(sys.argv) < 6:*  *print("please use: run blueprint\_UUID application\_name application\_description providername provider\_UUID")*  *else :*  *if len(sys.argv) > 6 :*  *print("too many arguments, bye bye")*  *else :*  *bpuuid = str(sys.argv[1])*  *appname = str(sys.argv[2])*  *appdesc = str(sys.argv[3])*  *providername = str(sys.argv[4])*  *provideruuid = str(sys.argv[5])*  *# verify if Blueprint uuid actually exists in the system otherwise request got errors*  *element = 'metadata'*  *request = 'blueprints/list'*  *base\_url = "https://" + cluster\_ip\_address + API\_AHV + request*  *uuid\_out\_bp=\_http\_request(base\_url, payload, element)*  *# verify if provider exists in the system otherwise request got error*  *request = 'blueprints/'*  *base\_url = "https://" + cluster\_ip\_address + API\_AHV + request + bpuuid*  *uuid\_out\_provider=\_http\_request\_2(base\_url, payload)*  *for i in uuid\_out\_bp :*  *if bpuuid == i :*  *print("found blueprint uuid")*  *for j in uuid\_out\_provider :*  *if provideruuid == j :*  *print("have the provider uuid associated with above bluprint")*  *bp\_payload = \_payload2(providername, provideruuid, appname, appdesc)*  *# pprint.pprint(json.loads(bp\_payload))*  *\_suppress\_security()*  *base\_url2 = "https://" + cluster\_ip\_address + API\_AHV + "blueprints/" + bpuuid + "/simple\_launch"*  *response = requests.request("POST", base\_url2, data=bp\_payload, headers=headers, verify=False)*  *pprint.pprint(response.text)* |

In case we don’t find BP uuid and/or the associated profile, request does not happen (otherwise will get an error.