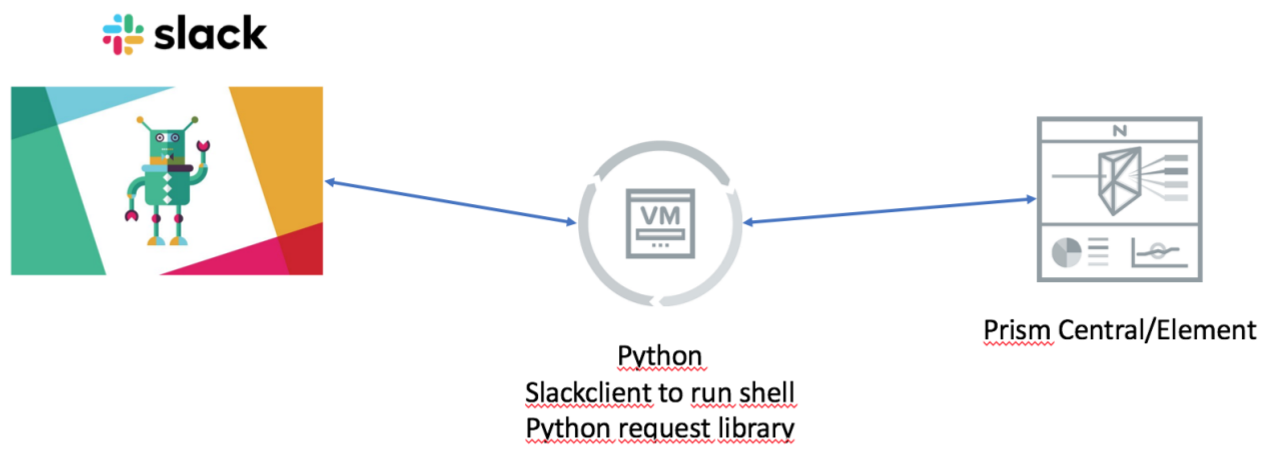
High level view:

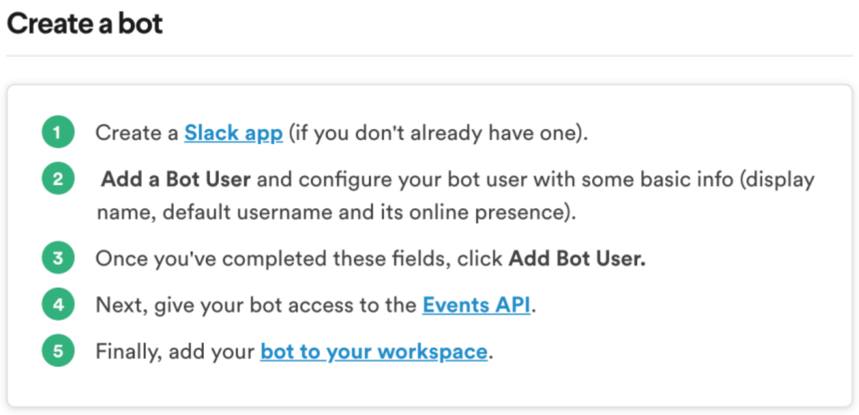


Step 1: have a linux VM connected to internet with python installed

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 .
```

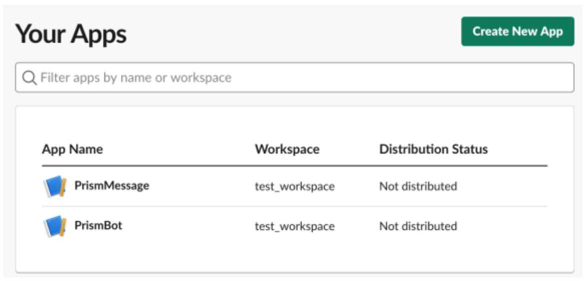
Step 2: Create a Slackbot and obtain your token
create a bot on your slack workspace:
https://slack.com/intl/en-it/help/articles/115005265703-create-a-bot-for-your-workspace



Create a bot

1. Create a Slack app (if you don't already have one).
2. **Add a Bot User** and configure your bot user with some basic info (display name, default username and its online presence).
3. Once you've completed these fields, click **Add Bot User.**
4. Next, give your bot access to the Events API.
5. Finally, add your bot to your workspace.

You can find your applications here:
https://api.slack.com/



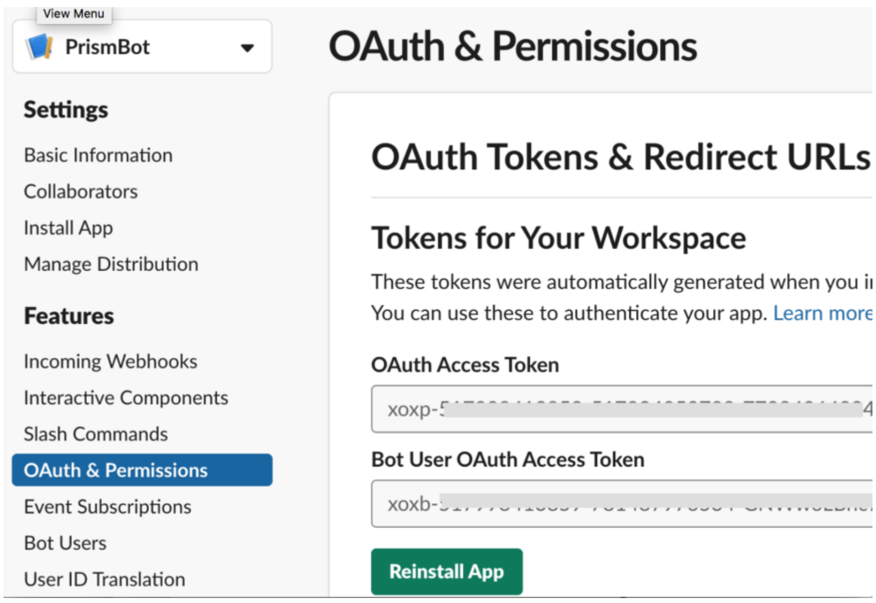
Your Apps

Create New App

Filter apps by name or workspace

App Name	Workspace	Distribution Status
PrismMessage	test_workspace	Not distributed
PrismBot	test_workspace	Not distributed

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



OAuth & Permissions

OAuth Tokens & Redirect URLs

Tokens for Your Workspace

These tokens were automatically generated when you i
You can use these to authenticate your app. Learn more

OAuth Access Token

xoxp-5[redacted]4

Bot User OAuth Access Token

xoxb-[redacted]

Reinstall App

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:

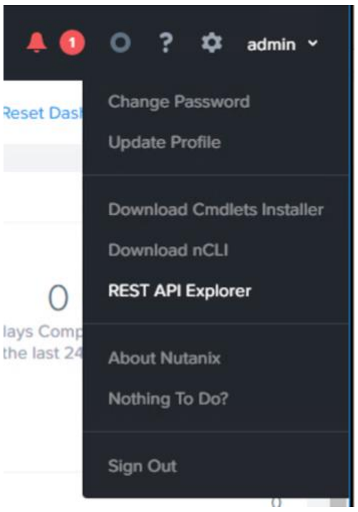
1. Downgrading the `websocket-client` library to `0.47.0`
2. Or, download the certificate (`wget https://www.tbs-certificats.com/issuerdata/DigiCertGlobalRootCA.crt`), then set the environment variable `export WEBSOCKET_CLIENT_CA_BUNDLE=DigiCertGlobalRootCA.crt`

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'

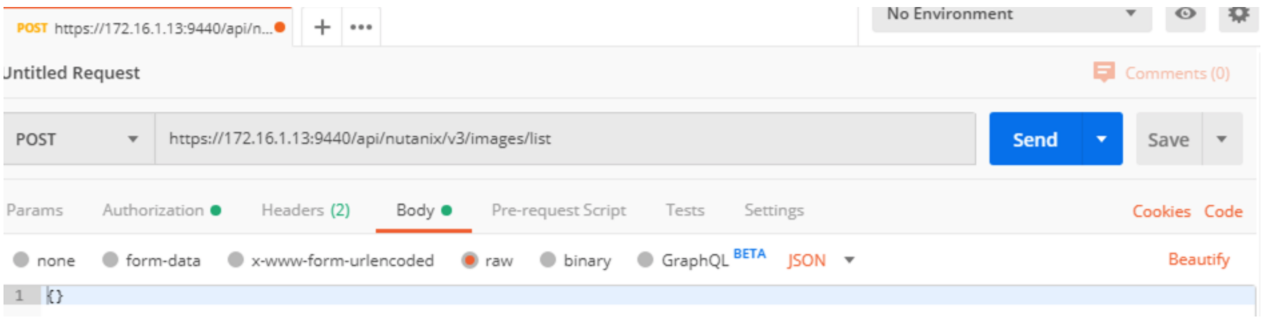
Step 4:
We have several examples on how to use API with Nutanix, common method to learn is the use of API explorer in all Nutanix PRISM interface (both Prism Element and Prism Central):



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…


POST https://172.16.1.13:9440/api/n...

No Environment

Untitled Request

Comments (0)

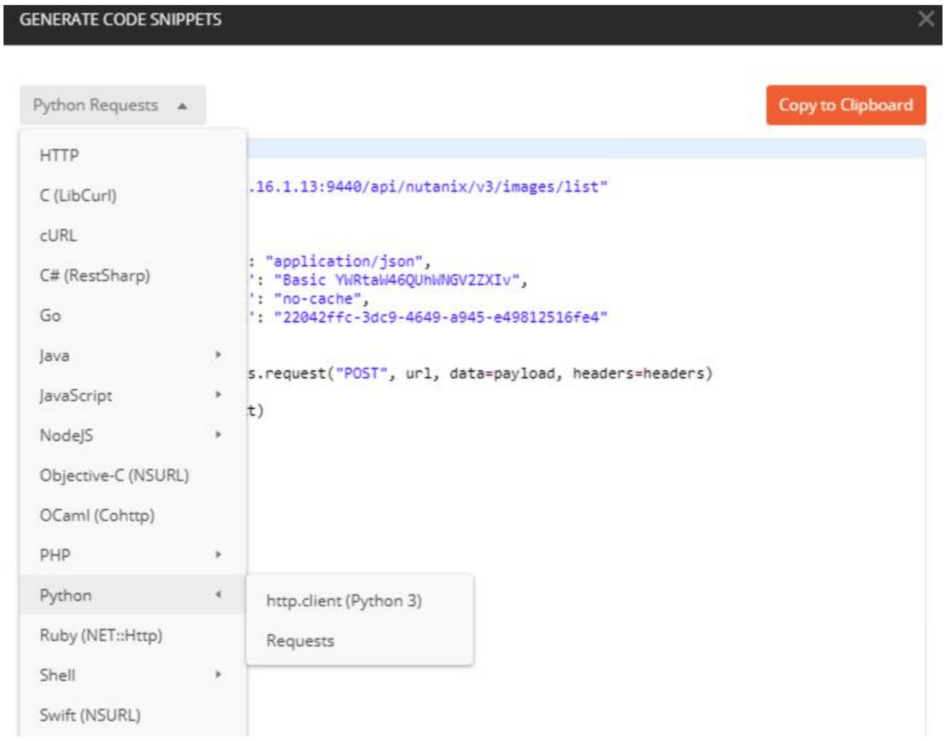
POST https://172.16.1.13:9440/api/nutanix/v3/images/list

Send Save

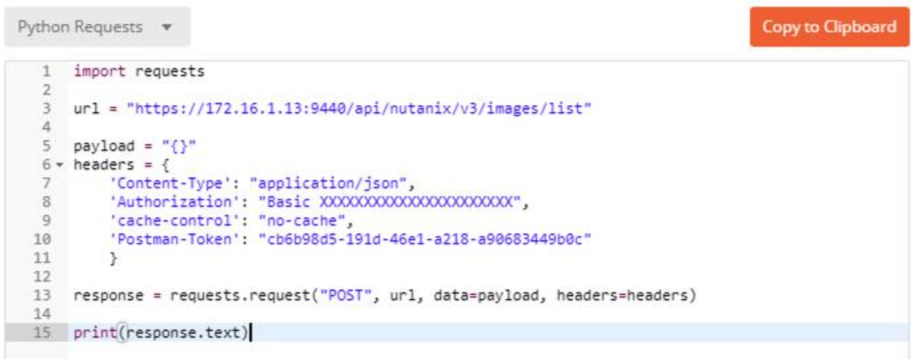
Params Authorization Headers (2) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL BETA JSON Beautify

```
{}
```



```
.16.1.13:9440/api/nutanix/v3/images/list"


: "application/json",
': "Basic YWRtaW46QUhWNGV2ZXIv",
': "no-cache",
': "22042ffc-3dc9-4649-a945-e49812516fe4"


s.request("POST", url, data=payload, headers=headers)

t)
```



```python
import requests

url = "https://172.16.1.13:9440/api/nutanix/v3/images/list"

payload = "{}"
headers = {
    'Content-Type': "application/json",
    'Authorization': "Basic XXXXXXXXXXXXXXXXXXXXXX",
    'cache-control': "no-cache",
    'Postman-Token': "cb6b98d5-191d-46e1-a218-a90683449b0c"
    }

response = requests.request("POST", url, data=payload, headers=headers)

print(response.text)
```

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'

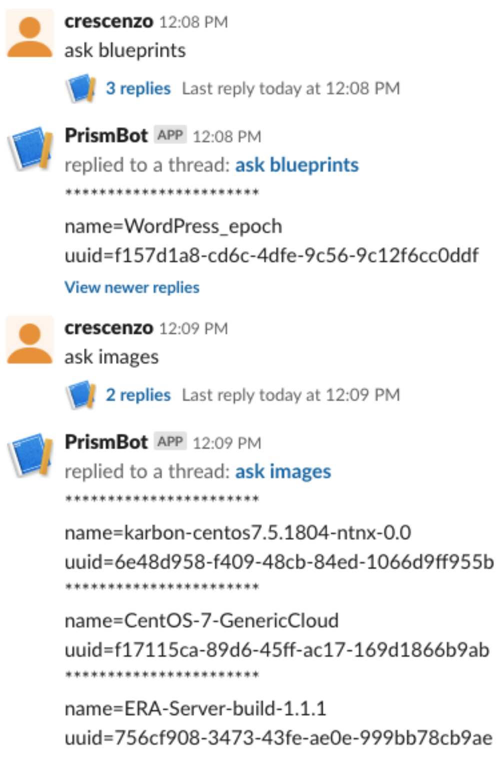
payload = json.dumps(
{"kind": "blueprint"
}, sort_keys=True)

```
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:


crescenzo 12:08 PM
ask blueprints
3 replies Last reply today at 12:08 PM

PrismBot APP 12:08 PM
replied to a thread: ask blueprints

```
**********************
name=WordPress_epoch
uuid=f157d1a8-cd6c-4dfe-9c56-9c12f6cc0ddf
```

View newer replies

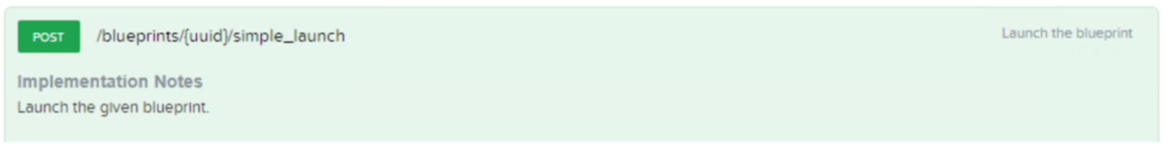
crescenzo 12:09 PM
ask images
2 replies Last reply today at 12:09 PM

PrismBot APP 12:09 PM
replied to a thread: ask images

```
**********************
name=karbon-centos7.5.1804-ntnx-0.0
uuid=6e48d958-f409-48cb-84ed-1066d9ff955b
**********************
name=CentOS-7-GenericCloud
uuid=f17115ca-89d6-45ff-ac17-169d1866b9ab
**********************
name=ERA-Server-build-1.1.1
uuid=756cf908-3473-43fe-ae0e-999bb78cb9ae
```

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.



/blueprints/{uuid}/simple_launch

Launch the blueprint

Implementation Notes

Launch the given blueprint.

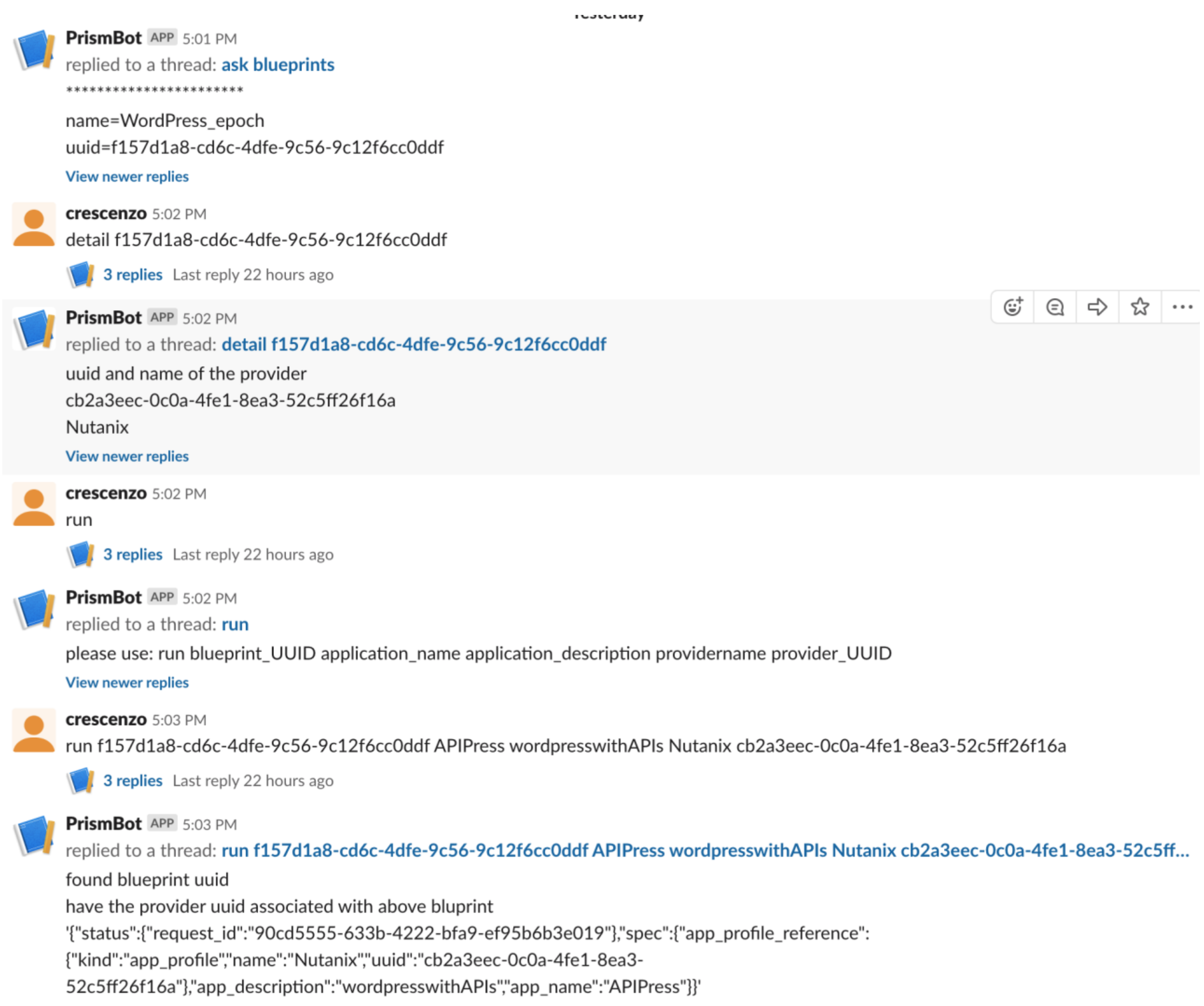
It is required to have the blueprint id and the application profile provider name and uuid.

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:



Yesterday

PrismBot APP 5:01 PM
replied to a thread: ask blueprints
****
name=WordPress_epoch
uuid=f157d1a8-cd6c-4dfe-9c56-9c12f6cc0ddf
View newer replies

crescenzo 5:02 PM
detail f157d1a8-cd6c-4dfe-9c56-9c12f6cc0ddf
3 replies Last reply 22 hours ago

PrismBot APP 5:02 PM
replied to a thread: detail f157d1a8-cd6c-4dfe-9c56-9c12f6cc0ddf
uuid and name of the provider
cb2a3eec-0c0a-4fe1-8ea3-52c5ff26f16a
Nutanix
View newer replies

crescenzo 5:02 PM
run
3 replies Last reply 22 hours ago

PrismBot APP 5:02 PM
replied to a thread: run
please use: run blueprint_UUID application_name application_description providername provider_UUID
View newer replies

crescenzo 5:03 PM
run f157d1a8-cd6c-4dfe-9c56-9c12f6cc0ddf APIPress wordpresswithAPIs Nutanix cb2a3eec-0c0a-4fe1-8ea3-52c5ff26f16a
3 replies Last reply 22 hours ago

PrismBot APP 5:03 PM
replied to a thread: run f157d1a8-cd6c-4dfe-9c56-9c12f6cc0ddf APIPress wordpresswithAPIs Nutanix cb2a3eec-0c0a-4fe1-8ea3-52c5ff...
found blueprint uuid
have the provider uuid associated with above bluprint
'{"status":{"request_id":"90cd5555-633b-4222-bfa9-ef95b6b3e019"},"spec":{"app_profile_reference":{"kind":"app_profile","name":"Nutanix","uuid":"cb2a3eec-0c0a-4fe1-8ea3-52c5ff26f16a"},"app_description":"wordpresswithAPIs","app_name":"APIPress"}}'

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.