KPMG TECHNICAL CHALLENGE

## Challenge #1

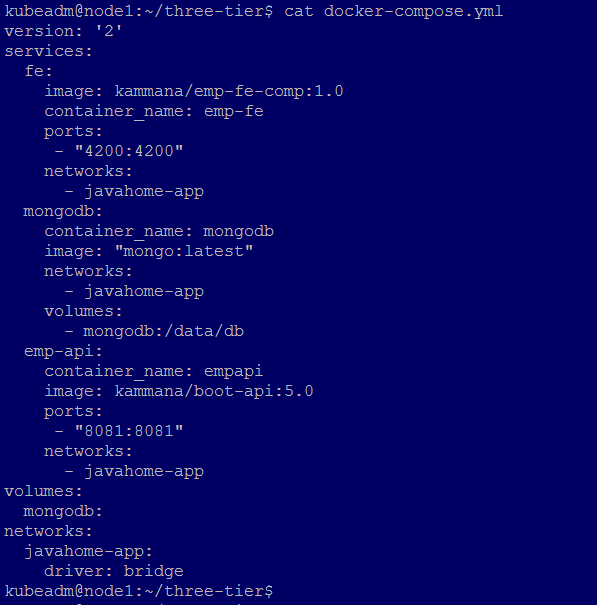
A 3 tier environment is a common setup. Use a tool of your choosing/familiarity creates these resources. Please remember we will not be judged on the outcome but more focusing on the approach, style and reproducibility.

## Response:

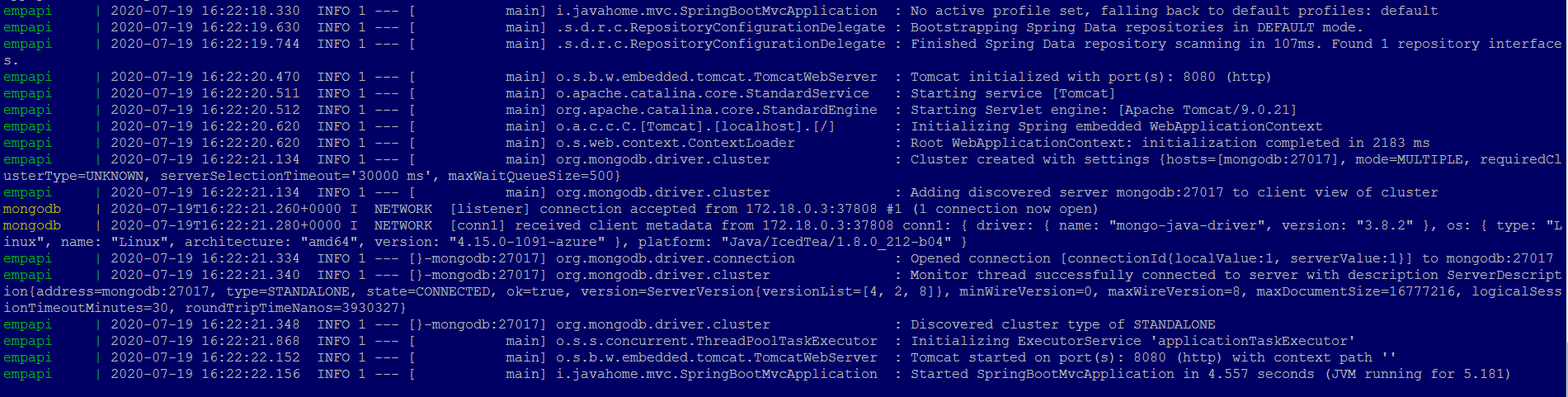
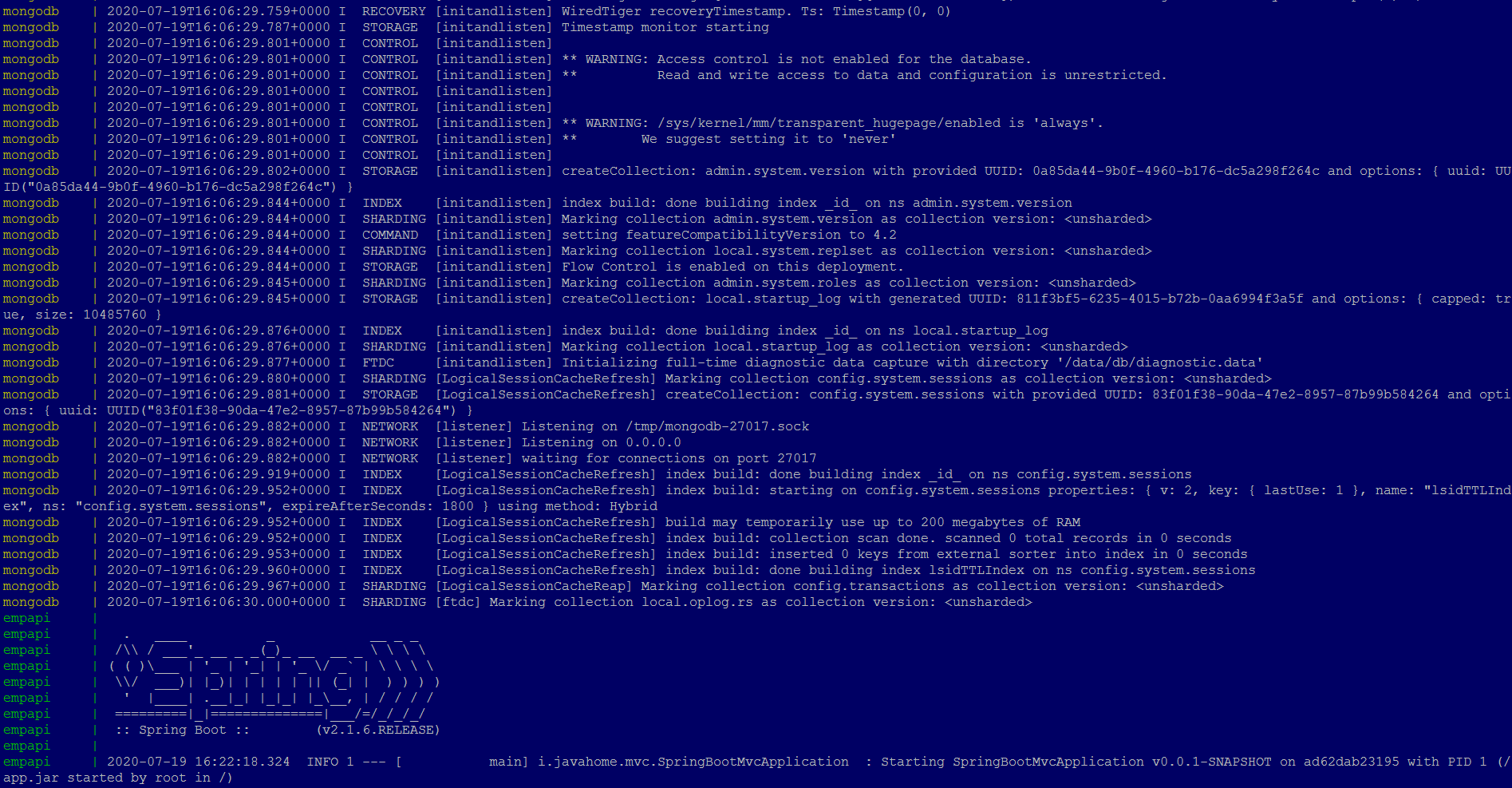
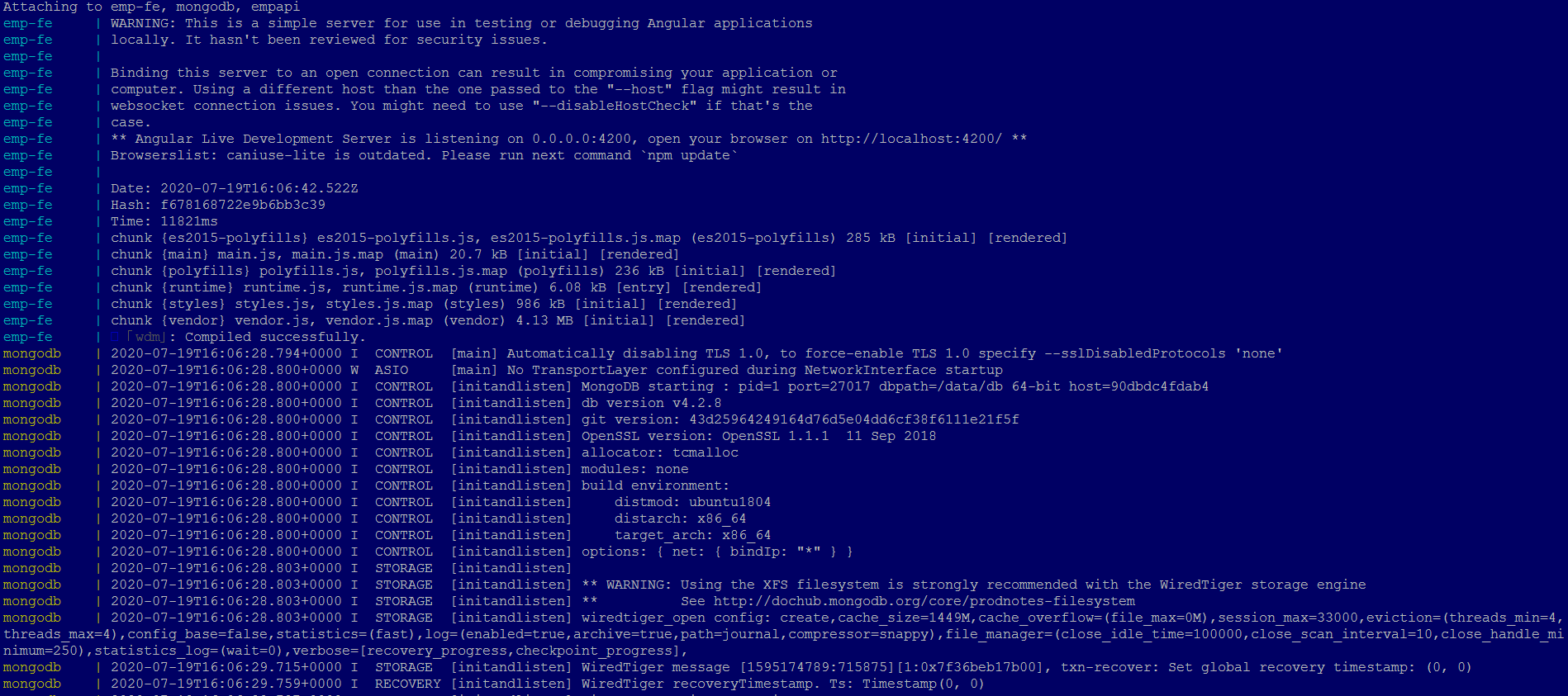
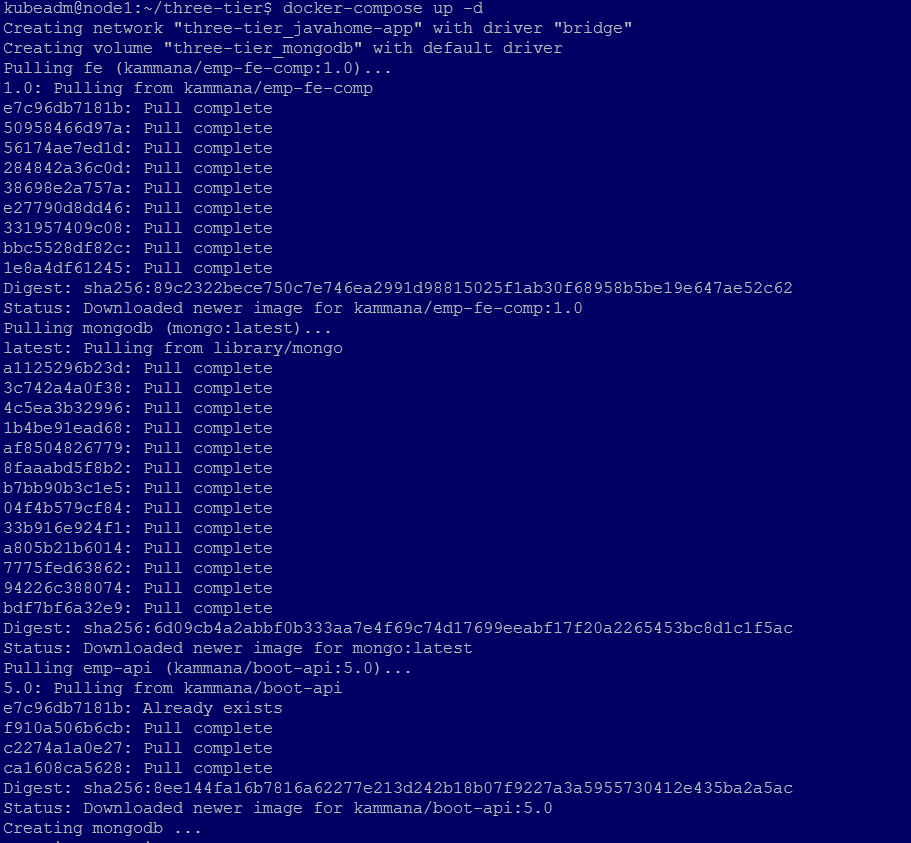
To address the above challenge #1, I am using the docker compose file to create a three tier environment. Since there are three components, we have to spin containers for each of the components. We have to spin the containers in the following way using the docker compose file which creates application with three containers, the docker-compose file is given in the screen shot along with the outputs

1. Container 1 – Frontend
2. Container 2  – Mongo DB
3. Container 3 – API

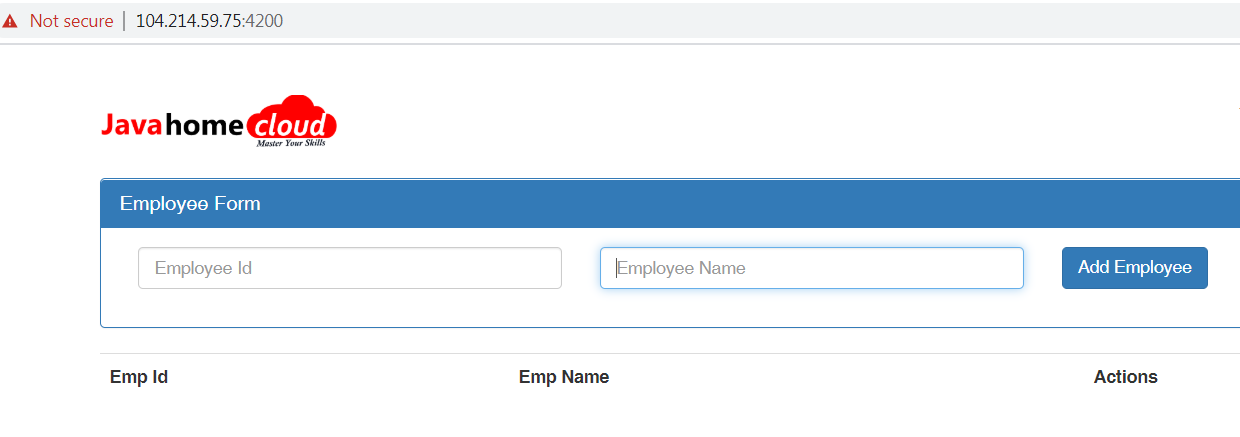
Docker-Compose file in the below



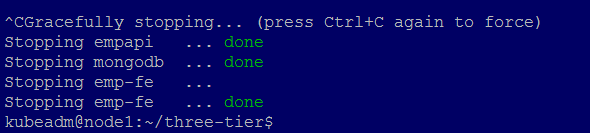
Docker-compose up can be used to build and bring up the three containers, below screenshot is the complete output for the compose file



You can access the application using <http://localhost:4200> or http://<IPAddress>:4200, I am using my docker host ip in the example below 104.214.59.75



Once application is validated, we can stop the containers by pressing ctrl+c



## Challenge #2

We need to write code that will query the Meta data of an instance within aws and provide a json formatted output. The choice of language and implementation is up to you.

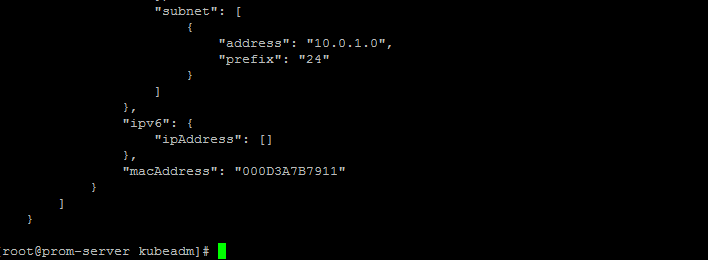
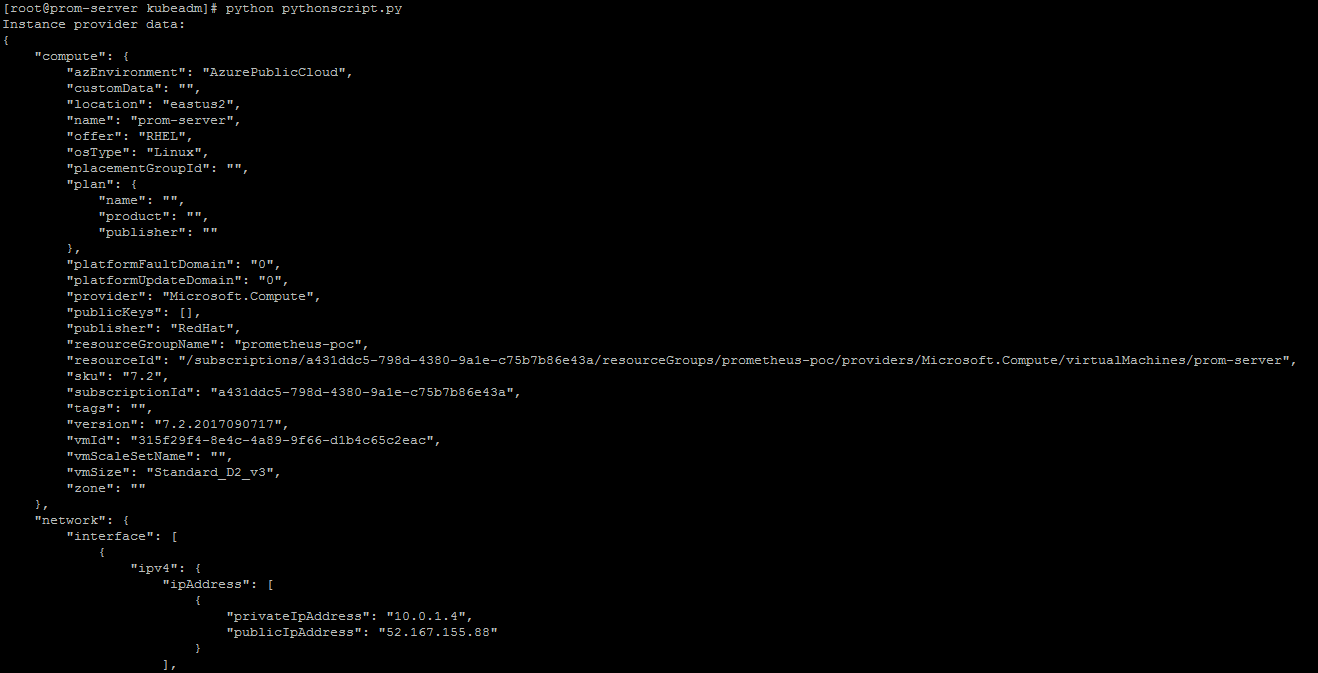
## Response:

I am using the below python script to create the instance in azure cloud, Also which gives the JSON formatted output.

### Please find the script snippet below

|  |
| --- |
| #!/usr/bin/env python3  import base64  import json  import re  import subprocess  import requests  imds\_server\_base\_url = "http://169.254.169.254"  instance\_api\_version = "2019-03-11"  instance\_endpoint = imds\_server\_base\_url + "/metadata/instance?api-version=" + instance\_api\_version  attested\_api\_version = "2019-03-11"  attested\_nonce = "1234576"  attested\_endpoint = imds\_server\_base\_url + "/metadata/attested/document?api-version=" + attested\_api\_version + "&nonce=" + attested\_nonce  def api\_call(endpoint):  headers={'Metadata': 'True'}  json\_obj = requests.get(endpoint, headers=headers).json()  return json\_obj  def main():  # Instance provider API call  instance\_json = api\_call(instance\_endpoint)  print("Instance provider data:")  print(json.dumps(instance\_json, sort\_keys=True, indent=4, separators=(',', ': ')))  if \_\_name\_\_ == "\_\_main\_\_":  main() |

## Output of above script



## Challenge #3

We have a nested object, we would like a function that you pass in the object and a key and get back the value. How this is implemented is up to you.

**Example Inputs**

object = {“a”:{“b”:{“c”:”d”}}}

key = a/b/c

object = {“x”:{“y”:{“z”:”a”}}}

key = x/y/z

value = a

## Response:

Find the below script. Added a function which gives the value, captured the script output in the screenshot below

|  |
| --- |
| import json  #Function that expectes object and key to manipulate key and value  def func1(data,input\_key):  for key,value in data.items():  if input\_key == key:  print(value)  #print (str(key)+'->'+str(value))  if type(value) == type(dict()):  func1(value, input\_key)  elif type(value) == type(list()):  for val in value:  if type(val) == type(str()):  pass  elif type(val) == type(list()):  pass  else:  func1(val)  x = '{"a":{"b":{"c":"d"}}}'  data = json.loads(x)  func1(data,'b') |

## Output of the above script



In the above example if you give the function **func1(data,'a')**  changed the value from b to a

Output will be {b: c: d}