Balavignesh manoharan

10 Python Libraries that every DevOps Engineer should know

Introduction:

In this article, we will learn about python libraries that should be on every DevOps Engineer’s list.

Overview:

- boto3
- paramiko
- docker
- kubernetes
- pyyaml
- requests
- fabric
- pytest
- ansible-runner
- sys

Let’s get started

boto3

The AWS SDK for Python is referred as boto3. Using boto3 one can create, configure and manage AWS services such as Amazon Elastic Compute Cloud (Amazon EC2), Amazon Simple Storage Service (S3).

Prerequisites:

- AWS Account

- Python 3.9 or later
- Install boto3 using `pip install boto3`
- AWS CLI
- Configure using `aws configure`

Example:

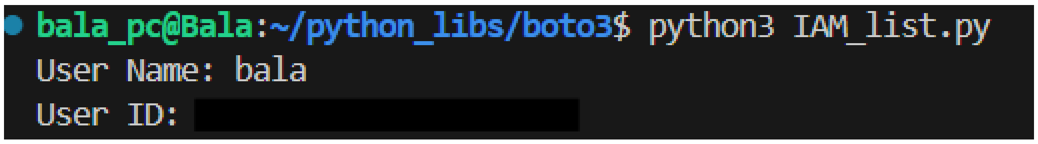
In this library, we will be using two examples:

1. **Listing IAM users**

```
import boto3

iam = boto3.client('iam')
paginator = iam.get_paginator('list_users')
for response in paginator.paginate():
    for user in response['Users']:
        print("User Name:", user['UserName'])
        print("User ID:", user['UserId'])
```

```
python3 IAM_list.py
```

Output:



```
bala_pc@Bala:~/python_libs/boto3$ python3 IAM_list.py
User Name: bala
User ID: [redacted]
```

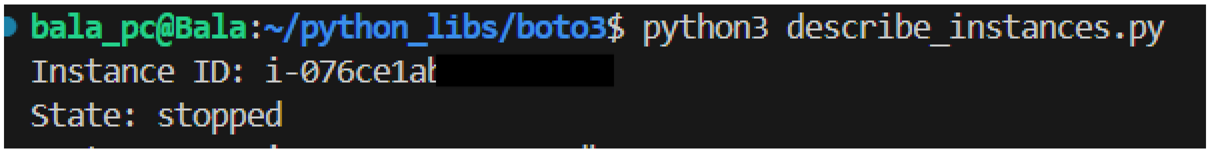
2. **Listing all the Instances**

```
import boto3

ec2 = boto3.client('ec2')

response = ec2.describe_instances()
```

```
for reservation in response['Reservations']:
  for instance in reservation['Instances']:
    print("Instance ID:", instance['InstanceId'])
    print("State:", instance['State']['Name'])

#print(response)
```

```
python3 describe_instance.py
```

Output:



```
bala_pc@Bala:~/python_libs/boto3$ python3 describe_instances.py
Instance ID: i-076ce1ab[redacted]
State: stopped
```

paramiko

Paramiko is a low level Python library used for executing remote shell commands or transferring files securely using SSH.

Prerequisites:

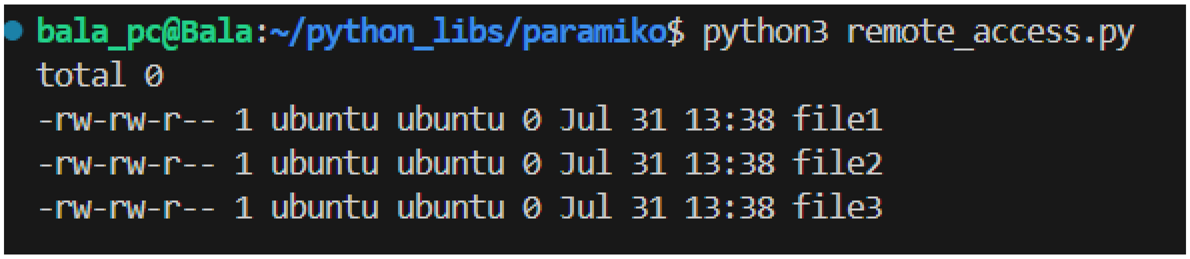
- Install paramiko using pip
- EC2 Instance running

Example:

```
import paramiko

hostname = '13.233.98.222'
port = 22
username = 'ubuntu'
key_file = '/Downloads/new_key.pem'

client = paramiko.SSHClient()
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
client.connect(hostname=hostname, port=port, username=username, key_file
name=key_file)

stdin, stdout, stderr = client.exec_command("ls -l")
print(stdout.read().decode())

client.close()
```

Set the correct path of your private key file and mention the hostname correctly.

Output:



```
bala_pc@Bala:~/python_libs/paramiko$ python3 remote_access.py
total 0
-rw-rw-r-- 1 ubuntu ubuntu 0 Jul 31 13:38 file1
-rw-rw-r-- 1 ubuntu ubuntu 0 Jul 31 13:38 file2
-rw-rw-r-- 1 ubuntu ubuntu 0 Jul 31 13:38 file3
```

docker

A Python library for the Docker Engine API. This lets you to do anything the docker command does, but from within the Python apps as run containers. manage containers etc.

Prerequisites:

- Install docker using pip

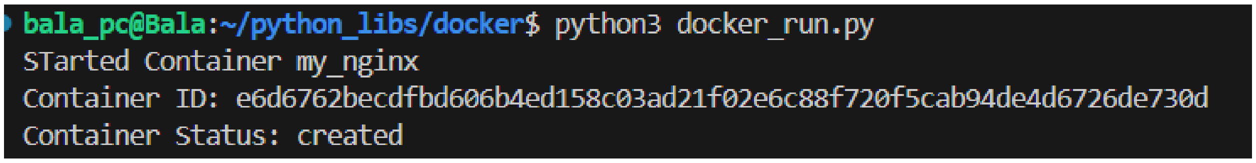
Example:

1. **Run a docker container**

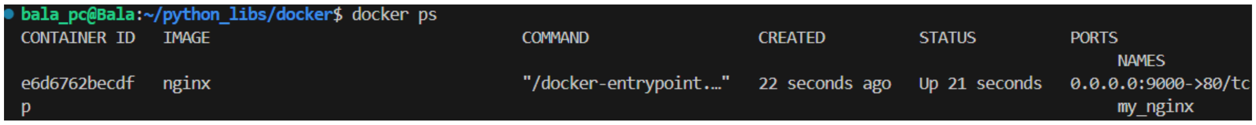
```
import docker

client = docker.from_env()
containers = client.containers.run("nginx", detach=True, name="my_nginx", p
orts={'80/tcp': 9000})
print ("STarted Container", containers.name)
```

```
print("Container ID:", containers.id)
print("Container Status:", containers.status)
```

Output:



```
bala_pc@Bala:~/python_libs/docker$ python3 docker_run.py
STarted Container my_nginx
Container ID: e6d6762becdfbd606b4ed158c03ad21f02e6c88f720f5cab94de4d6726de730d
Container Status: created
```

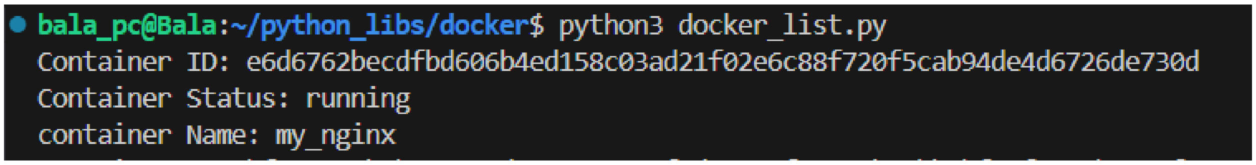


```
bala_pc@Bala:~/python_libs/docker$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS
                                                                                       NAMES
e6d6762becdf   nginx      "/docker-entrypoint.…"   22 seconds ago   Up 21 seconds   0.0.0.0:9000->80/tc
p                                                                                      my_nginx
```

2. **List the containers**

```
import docker

client = docker.from_env()
#container = client.containers.run("ubuntu", detach=True)
containers = client.containers.list()
for container in containers:
    print("Container ID:", container.id)
    print("Container Status:", container.status)
    print("container Name:", container.name)
```

Output:



```
bala_pc@Bala:~/python_libs/docker$ python3 docker_list.py
Container ID: e6d6762becdfbd606b4ed158c03ad21f02e6c88f720f5cab94de4d6726de730d
Container Status: running
container Name: my_nginx
```

Kubernetes

Kubernetes Client library lets you to connect to a cluster, list and manage resources like pods, deployment, namespaces etc.

Prerequisites:

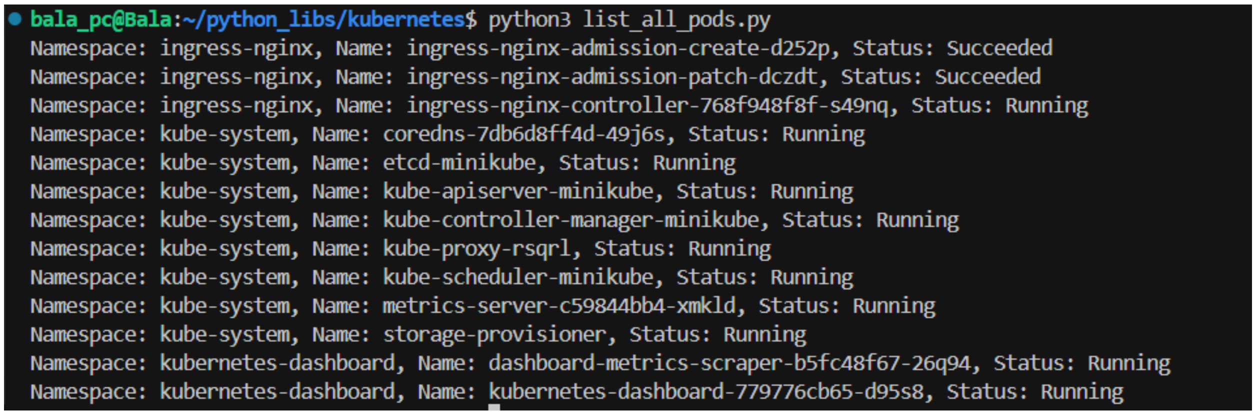
- Install Kubernetes using pip
- Make sure you have a cluster running (local or EKS)
- Kubectl configured

Example:

List all the pods from all the namespaces

```
from kubernetes import client, config

config.load_kube_config()  # Load kube config from default location
v1 = client.CoreV1Api()
pods = v1.list_pod_for_all_namespaces(watch=False)
for pod in pods.items:
    print(f"Namespace: {pod.metadata.namespace}, Name: {pod.metadata.na
me}, Status: {pod.status.phase}")
# This script lists all pods in all namespaces in a Kubernetes cluster.
```

Output:



```
bala_pc@Bala:~/python_libs/kubernetes$ python3 list_all_pods.py
Namespace: ingress-nginx, Name: ingress-nginx-admission-create-d252p, Status: Succeeded
Namespace: ingress-nginx, Name: ingress-nginx-admission-patch-dczdt, Status: Succeeded
Namespace: ingress-nginx, Name: ingress-nginx-controller-768f948f8f-s49nq, Status: Running
Namespace: kube-system, Name: coredns-7db6d8ff4d-49j6s, Status: Running
Namespace: kube-system, Name: etcd-minikube, Status: Running
Namespace: kube-system, Name: kube-apiserver-minikube, Status: Running
Namespace: kube-system, Name: kube-controller-manager-minikube, Status: Running
Namespace: kube-system, Name: kube-proxy-rsqrl, Status: Running
Namespace: kube-system, Name: kube-scheduler-minikube, Status: Running
Namespace: kube-system, Name: metrics-server-c59844bb4-xmkld, Status: Running
Namespace: kube-system, Name: storage-provisioner, Status: Running
Namespace: kubernetes-dashboard, Name: dashboard-metrics-scraper-b5fc48f67-26q94, Status: Running
Namespace: kubernetes-dashboard, Name: kubernetes-dashboard-779776cb65-d95s8, Status: Running
```

pyyaml

PyYAML is a python library which lets you read and write into YAML files. It can validate or update existing YAMLs before applying.

Prerequisites:

- Install pyyaml using pip

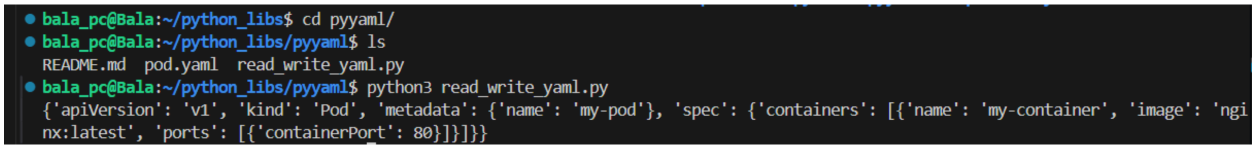
- A kubernetes Pod file (pod.yaml in the repo)

Example:

Read the YAML file

```
import yaml
with open('/home/bala_pc/python_libs/pyyaml/pod.yaml', 'r') as file:
    pod_data = yaml.safe_load(file)
print(pod_data)
# This code reads a YAML file containing pod configuration and prints the par
sed data.
```

Output:



```
● bala_pc@Bala:~/python_libs$ cd pyyaml/
● bala_pc@Bala:~/python_libs/pyyaml$ ls
  README.md  pod.yaml  read_write_yaml.py
● bala_pc@Bala:~/python_libs/pyyaml$ python3 read_write_yaml.py
  {'apiVersion': 'v1', 'kind': 'Pod', 'metadata': {'name': 'my-pod'}, 'spec': {'containers': [{'name': 'my-container', 'image': 'ngi
  nx:latest', 'ports': [{'containerPort': 80}]}]}}
```

requests

Requests is a simple HTTP library for python which allows you to send HTTP requests easily.

Prerequisites:

- Install requests using pip

Example:

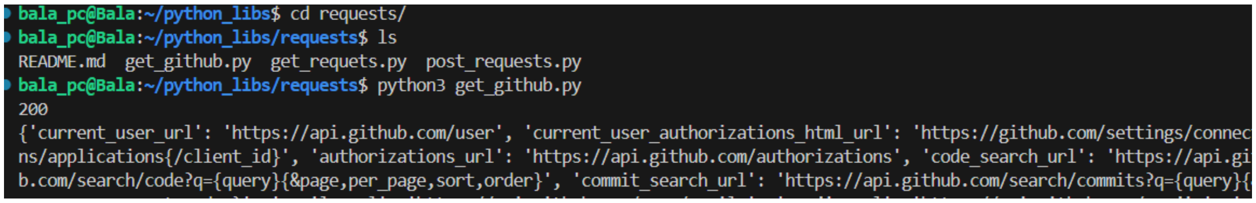
In this case, we have three examples.

1. **Sending GET requests to Github API**

```
import requests

response = requests.get('https://api.github.com')
print(response.status_code)
print(response.json())
```

```
# This code sends a GET request to the GitHub API and prints the status code
and response data.
```

Output:

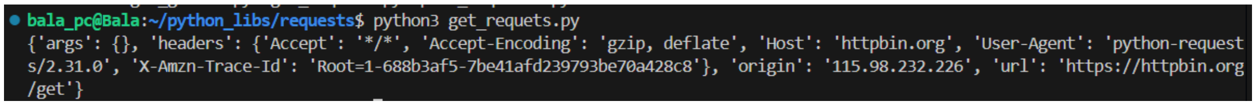


```
bala_pc@Bala:~/python_libs$ cd requests/
bala_pc@Bala:~/python_libs/requests$ ls
README.md  get_github.py  get_requets.py  post_requests.py
bala_pc@Bala:~/python_libs/requests$ python3 get_github.py
200
{'current_user_url': 'https://api.github.com/user', 'current_user_authorizations_html_url': 'https://github.com/settings/connec
ns/applications{/client_id}', 'authorizations_url': 'https://api.github.com/authorizations', 'code_search_url': 'https://api.gi
b.com/search/code?q={query}{&page,per_page,sort,order}', 'commit_search_url': 'https://api.github.com/search/commits?q={query}{
```

2. **Sending GET requests to httpbin.org**

```
import requests

response = requests.get("https://httpbin.org/get")
print(response.json())
```

Output:

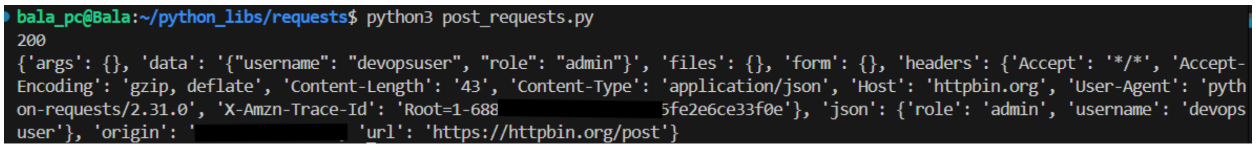


```
bala_pc@Bala:~/python_libs/requests$ python3 get_requets.py
{'args': {}, 'headers': {'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate', 'Host': 'httpbin.org', 'User-Agent': 'python-request
s/2.31.0', 'X-Amzn-Trace-Id': 'Root=1-688b3af5-7be41afd239793be70a428c8'}, 'origin': '115.98.232.226', 'url': 'https://httpbin.org
/get'}
```

3. **Posting a request to httpbin.org**

```
import requests

url = 'https://httpbin.org/post'
data = {'username': 'devopsuser', 'role': 'admin'}

response = requests.post(url, json=data)
print(response.status_code)
print(response.json())
# This code sends a POST request to httpbin.org with JSON data and prints th
e status code and response data.
```

Output:



```
bala_pc@Bala:~/python_libs/requests$ python3 post_requests.py
200
{'args': {}, 'data': '{"username": "devopsuser", "role": "admin"}', 'files': {}, 'form': {}, 'headers': {'Accept': '*/*', 'Accept-
Encoding': 'gzip, deflate', 'Content-Length': '43', 'Content-Type': 'application/json', 'Host': 'httpbin.org', 'User-Agent': 'pyth
on-requests/2.31.0', 'X-Amzn-Trace-Id': 'Root=1-688[redacted]5fe2e6ce33f0e'}, 'json': {'role': 'admin', 'username': 'devops
user'}, 'origin': '[redacted] 'url': 'https://httpbin.org/post'}
```

fabric

Fabric is a high level Python library designed to execute shell commands remotely over SSH, yielding useful Python objects in return.

Prerequisites

- Install fabric using pip
- Running EC2 instance

Example:

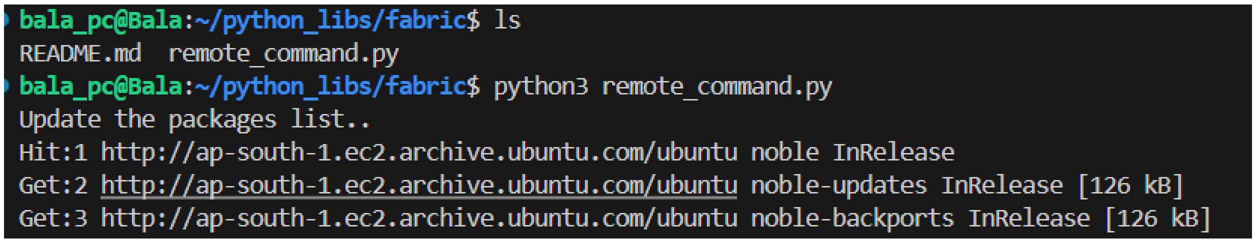
Connect to a remote host and execute commands

```
from fabric import Connection

conn = Connection(
    host="13.127.179.12",
    user="ubuntu",
    connect_kwargs={"key_filename": "/Downloads/new_key.pem"}
)

def setup_nginx():
    print("Update the packages list..")
    conn.run("sudo apt-get update", hide=False)
    print("Install Nginx..")
    conn.run("sudo apt-get install -y nginx", hide=False)
    print("NGINX installed successfully.")

setup_nginx()
# This code connects to a remote host using Fabric and runs a command to g
```

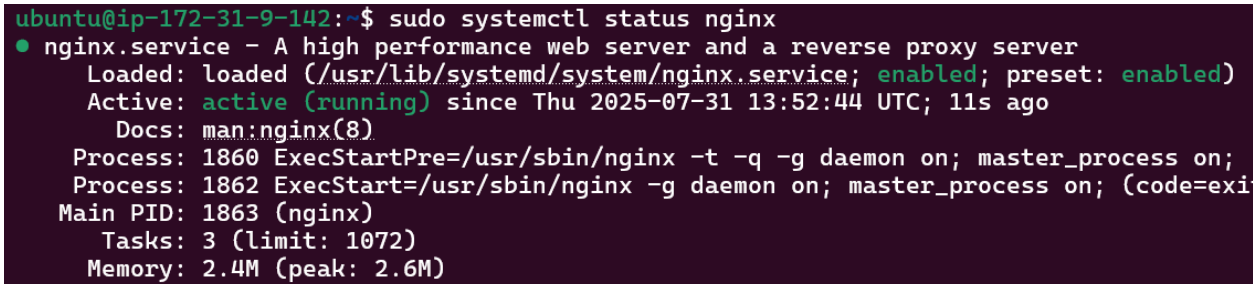
```
et the system name
# and prints the output.
```

Provide your server details accordingly.

Output:



```
bala_pc@Bala:~/python_libs/fabric$ ls
README.md  remote_command.py
bala_pc@Bala:~/python_libs/fabric$ python3 remote_command.py
Update the packages list..
Hit:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
```



```
ubuntu@ip-172-31-9-142:~$ sudo systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: enabled)
     Active: active (running) since Thu 2025-07-31 13:52:44 UTC; 11s ago
       Docs: man:nginx(8)
    Process: 1860 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on;
    Process: 1862 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exi
   Main PID: 1863 (nginx)
      Tasks: 3 (limit: 1072)
     Memory: 2.4M (peak: 2.6M)
```

pytest

The pytest framework makes it easy to write small, readable tests and can also scale to support complex functional testing and applications and libraries.

Prerequisites:

- Install pytest using pip

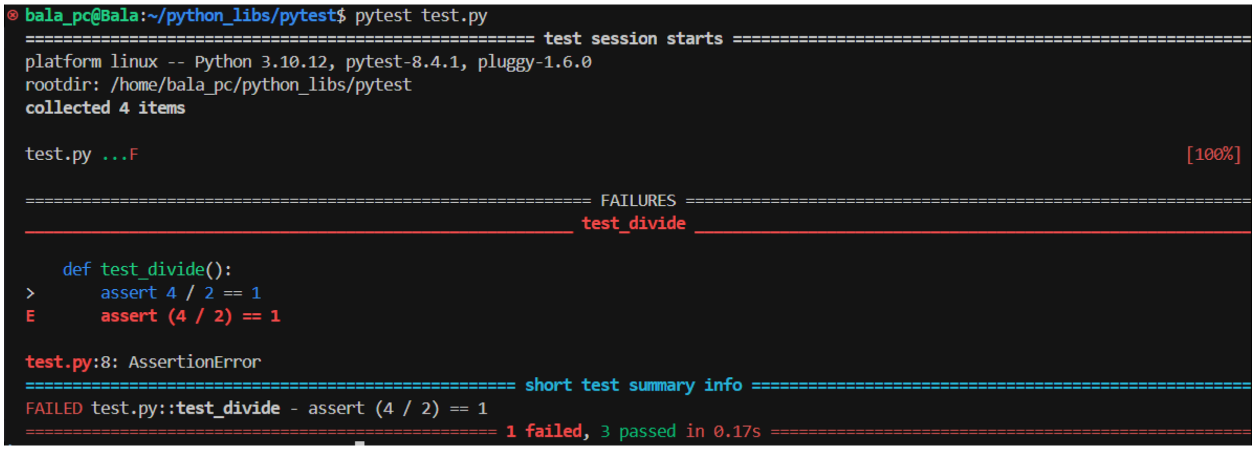
Example:

1. **Writing simple test (Failed test)**

```
def test_add():
    assert 1 + 1 == 2
def test_subtract():
    assert 2 - 1 == 1
def test_multiply():
    assert 2 * 2 == 4
```

```
def test_divide():
    assert 4 / 2 == 1
```

Output:

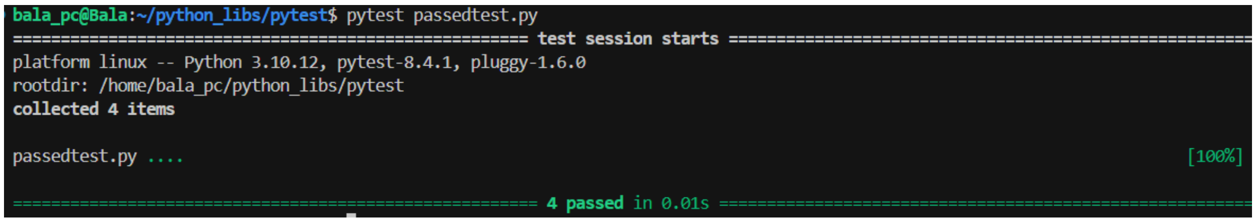


```
bala_pc@Bala:~/python_libs/pytest$ pytest test.py
============================== test session starts ==============================
platform linux -- Python 3.10.12, pytest-8.4.1, pluggy-1.6.0
rootdir: /home/bala_pc/python_libs/pytest
collected 4 items

test.py ...F                                                              [100%]

=================================== FAILURES ====================================
__________________________________ test_divide __________________________________

    def test_divide():
>       assert 4 / 2 == 1
E       assert (4 / 2) == 1

test.py:8: AssertionError
============================ short test summary info ============================
FAILED test.py::test_divide - assert (4 / 2) == 1
========================== 1 failed, 3 passed in 0.17s ==========================
```

2. **Writing simple test (Passed test)**

```
def test_add():
    assert 1 + 1 == 2
def test_subtract():
    assert 2 - 1 == 1
def test_multiply():
    assert 2 * 2 == 4
def test_divide():
    assert 4 / 2 == 2
```

Output:



```
bala_pc@Bala:~/python_libs/pytest$ pytest passedtest.py
============================== test session starts ==============================
platform linux -- Python 3.10.12, pytest-8.4.1, pluggy-1.6.0
rootdir: /home/bala_pc/python_libs/pytest
collected 4 items

passedtest.py ....                                                        [100%]

=============================== 4 passed in 0.01s ===============================
```

ansible-runner

Ansible Runner is a tool and a python library that helps when interfacing with Ansible directly or as part of another system.

Prerequisites:

- Install the ansible-runner library using pip

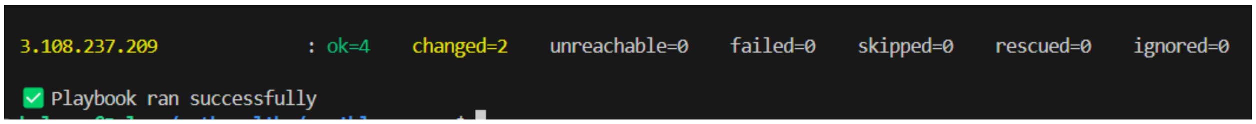
Example:

Connect to the server and install packages

```
import ansible_runner

result = ansible_runner.run(
    private_data_dir='.',
    playbook='/home/python_libs/ansible-runner/project/playbook.yaml',
    inventory='/home/python_libs/ansible-runner/inventory/hosts',
)

print("Status:", result.status)
print("RC:", result.rc)
print("Stdout:\n", result.stdout.read())

if result.rc != 0:
    print("❌ Playbook failed")
else:
    print("✅ Playbook ran successfully")
```

Set correct path for the playbook and inventory accordingly.

Output:



```
3.108.237.209              : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

✅ Playbook ran successfully
```

sys - System-specific parameters and functions

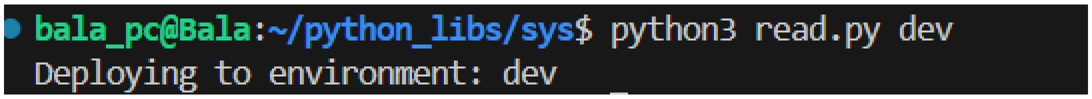
This module provides access to some variables used or maintained by the interpreter and functions that interact strongly with the interpreter.

Example:

Checking for arguments and prints the message

```
import sys

if len(sys.argv) < 2:
    print("Usage: python script.py <env>")
    sys.exit(1)

env = sys.argv[1]
print(f"Deploying to environment: {env}")
# This code checks if an environment argument is provided and prints a mess
age indicating the deployment environment.
```

Output:



```
bala_pc@Bala:~/python_libs/sys$ python3 read.py dev
Deploying to environment: dev
```

Conclusion:

Python has gained significant importance in the DevOps ecosystem due to its ease of use, extensive libraries and adaptability across different platforms. In this article, we have explored some of the most used libraries which will benefit across different use cases. Whether you're automating routine tasks or managing infrastructure Python offers a powerful toolset.

.

.

If you found this post useful, give it a like👍

Repost♻️

Follow @Bala Vignesh for more such posts 💯🚀

Thank You For Reading

Balavignesh manoharan