# **Parallel Processing/Multithreading for Actual Deployment of Pytest case**

**Generic Pytest Case:**

* **Deploy and validate (test case should be synchronous)**
* **TC: Check status**
* **TC: Validate gcs**
* **TC: Validate Bq**
* **TC: Validate adls**
* **TC: To be explored for scalability (\*\* report specific test cases)**

1. **Try for fixture and utilize in existing class (cons: conftest will be overloaded as it grows, to split conftest will be difficult)**

* Fixtures
* Deploy and return deployment id and its status 🡪
* In pytest class 🡪 test case methods should validate the gcs and bq and adls once deployment is done

\*\*Note\*\* So for individual reports after deployment, test cases should run in synchronous way (if deployment is not success there is not point to go for gcs, bq and adls validation) \*\*

\*\* Same fixtures would be shared among all the py files can contain multiple classes

* Create test files:
* As per category

1. Then files will have multiple classes for individual reports (cons: as it grows will overload same file with lot of class if reports are more for same category)
2. Better approach (to discuss), create test files as per reports and any unique identity like version (check for duplicate reports case might be in release same reports have different versions)

* Outstanding challenges:
* Multithreading/ parallelism in pytest is achieved with xdist module.
* Challenges:

Case: Is it possible to run pytest in parallel mode and assign thread per test\_\*.py file. Say I have a tests folder like:

tests directory

test\_a.py - contains 3 methods

test\_b.py - contains 3 test methods

test\_c.py -contains 3 test methods.

Now can we run test\_a.py, test\_b.py and test\_c.py parallelly with 3 threads. and all the test in each script will run as it is i.e., in synchronous mode.

currently with python -m pytest -n 3, works like: assume test\_a.py is assigned 3 threads: since test\_a.py has 3 test methods, 3 threads will be assigned to 3 methods of test\_a.py, resulting 3 methods of test\_a.py in parallel rest others in synchronous mode.

Any suggestion would be highly appreciated, how we can assign 1 thread to each test file and resulting all test file classes to run in parallel?

Workaround:

To make pytest relalize there is only one test case in a class/test file which will get 1 thread as per method name starts with test\_ key word.

* So as per below code if methods are nested in a test cases in a class i.e. one test method can have 4 methods named other than test\_\*

Code: import unittest

import pytest

from time import sleep

from time import gmtime, strftime

'''

in python -m pytest -n <value>

-n value will refer to number of test in each classes will be schedules at a time

say class a has 3 test and class b has 3 test

-n 6 will result all 6 at once

if -n is 2 only two test of any of the class would run in parallel rest all will be executed in single mode

'''

'''

to reduce number of threads using -n

added the nested mehtods for status, gcs, bq, adls validation in one mehtod

resulting one report specific test cases validation using 1 thread else if we write

separate test cases we need to pass as many number of thread as the test cases

'''

'''

the four validation(status, bq, gcs, adls) for each report will have one test case as pass or fail

'''

class TestCategoryOne(unittest.TestCase):

    #pass report name/version in setUp class get report spec related entry from

    #new test config and use subroutine which deploy and return its status and deployment id

    #(there wont be for loop in deploy subroutine, only for report and version specific)

    @classmethod

    def setUp(cls):

        #pass report name and version(if possible check for version compatibility)

        delay = 5

        sleep(delay)

        cls.result = {

            "status": ['Success'],

            "gcs":['pass'],

            "BQ": ["pass"],

            "adls": ['pass']

        }

        print(f"\nstarted-waiting for {delay} seconds-->")

        print(strftime("%Y-%m-%d %H:%M:%S", gmtime()))

    def test\_report\_abc(self):

        def status\_validation():

            status = self.result['status'][0]

            print(f"status is {status}")

            sleep(2)

            assert status == 'Success'

        def gcs\_validation():

            gcs = self.result['gcs'][0]

            sleep(2)

            assert gcs == 'pass'

        def bq\_validation():

            bq = self.result['BQ'][0]

            print(f"status is {bq}")

            sleep(2)

            assert bq == 'pass'

        def adls\_validation():

            adls = self.result['adls'][0]

            print(f"status is {adls}")

            sleep(2)

            assert adls == 'pass'

        status\_validation()

        gcs\_validation()

        bq\_validation()

        adls\_validation()

class TestCategoryOne\_dash(unittest.TestCase):

    '''

    pass report name/version in setUp class get report spec related entry from

    new test config and use subroutine which deploy and return its status and deployment id

    (there wont be for loop in deploy subroutine, only for report and version specific)

    '''

    @classmethod

    def setUp(cls):

        #pass report name and version(if possible check for version compatibility)

        delay = 5

        sleep(delay)

        cls.result = {

            "status": ['Success'],

            "gcs":['pass'],

            "BQ": ["pass"],

            "adls": ['pass']

        }

        print(f"\nstarted-waiting for {delay} seconds-->")

        print(strftime("%Y-%m-%d %H:%M:%S", gmtime()))

    def test\_status\_validation(self):

        status = self.result['status'][0]

        print(f"status status is {status}")

        sleep(2)

        assert status == 'Success'

    def test\_gcs\_validation(self):

        gcs = self.result['gcs'][0]

        print(f"status gcs is {gcs}")

        sleep(2)

        assert gcs == 'pass'

    def test\_bq\_validation(self):

        bq = self.result['BQ'][0]

        print(f"status bq is {bq}")

        sleep(2)

        assert bq == 'pass'

    def test\_adls\_validation(self):

        adls = self.result['adls'][0]

        print(f"status adls is {adls}")

        sleep(2)

        assert adls == 'pass'

# **Changes to be added to avoid un-intentional run of test cases if multiple category’s test file are present.**

* Add pytest.ini file and configure default pytests directory to be covered.
* Always avoid using [ python -m pytest -s -v] i.e., without mentioning which directory to select for pytest case to run.
* Adding pytest.ini solves the issue by configuring which directory will be select by default in test path.
* **Safe Approach**: Always mention the directory for which test case is to be run/covered

**Ex**: python -m pytest <directory\_name>

* **Always restrict the pytest case run for a specific set of pytest using conftest pytest adoption and mandate the command line argument to validate first and then execute.**

**And add pytest exit case with proper hint.**

**<check code pushed in current branch>**

**pytest.ini configuration/code example:**

[pytest]

# naming convention/patterns to cover test

python\_files=test\_\*

python\_classes=Test\*

python\_functions=test\_\*

# to ignore directories

#norecursedirs =

#        regression\_tests

#        c

# to include directories

testpaths =

    tests

    regression\_tests