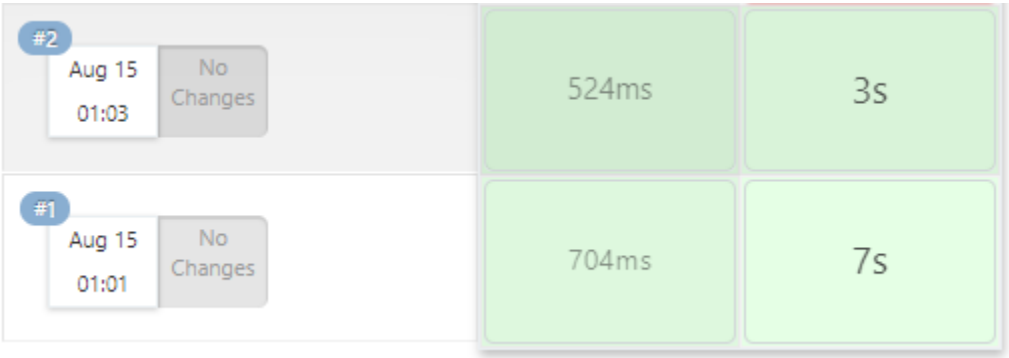
1. To begin this deployment, I needed an EC2. I had an EC2 created to be able to use Jenkins which was installed in a script I placed inside my EC2.
2. And this allowed me to use Jenkins due to the EC2 running a script that installs Jenkins. After that I made my multibranch pipeline.
3. I created my jenkins pipeline and edited it to match what the instructions wanted us to do for creating a multibranch pipeline.

22 lines (21 sloc) | 432 Bytes

```
pipeline {
    agent any
    stages {
      stage('test') {
       steps {
        sh '''#! /bin/bash
       python3 -m venv test3
       source test3/bin/activate
       pip install pip --upgrade
       pip install pytest
       py.test --verbose --junit-xml test-reports/results.xml sources/test_calc.py
       '''
      }
      post {
        always {
          junit 'test-reports/results.xml'
        }
      }
    }
  }
}
```

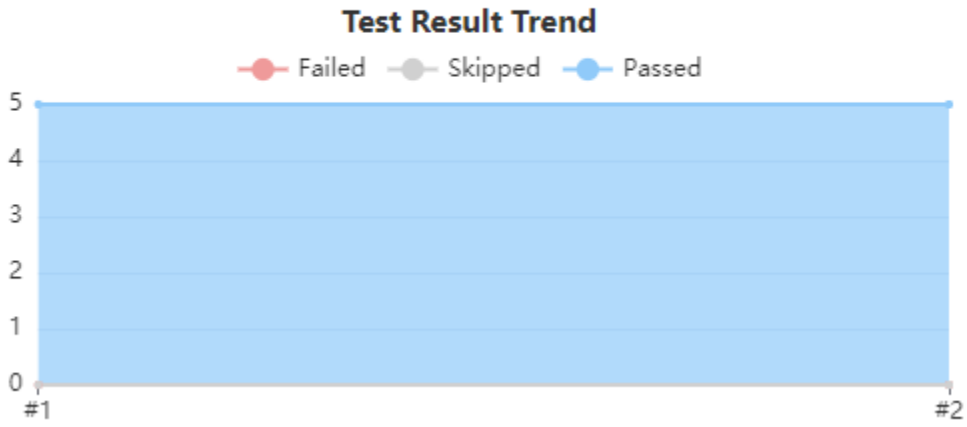
4. Then I went into my Jenkins by going into http://18.118.164.117:8080/ and create my Jenkins multibranch pipeline with a token I had already created before. I connected this multibranch pipeline to my github and my jenkins file.
5. After that I tried testing the pipeline to see if it was working without failure and it did work correctly.

Evidence of successful test build



#2	Aug 15, 2021 5:03 AM
#1	Aug 15, 2021 5:01 AM

Atom feed for all Atom feed for failures



6. Deployment 3 asked us to add a new component or feature. The feature already had what would happen if there were 2 arguments so I edited my Jenkins file to include a feature that if there are 3 arguments then it would multiply those 3 arguments. And also added what would happen if the arguments were not 2 and 3, the system would exit.

7. In the calc.py I also added a method/function on what would happen if there were three arguments and it would return the multiplication of the three arguments.

   Pic of calc

41 lines (38 sloc) | 1.53 KB

```
'''
The 'calc' library contains the 'add2' function that takes 2 values and adds
them together. If either value is a string (or both of them are) 'add2' ensures
they are both strings, thereby resulting in a concatenated result.
NOTE: If a value submitted to the 'add2' function is a float, it must be done so
in quotes (i.e. as a string).
'''

# If 'value' is not an integer, convert it to a float and failing that, a string.
def conv(value):
    try:
        return int(value)
    except ValueError:
        try:
            return float(value)
        except ValueError:
            return str(value)

# The 'add2' function itself
def add2(arg1, arg2):
    # Convert 'arg1' and 'arg2' to their appropriate types
    arg1conv = conv(arg1)
    arg2conv = conv(arg2)
    # If either 'arg1' or 'arg2' is a string, ensure they're both strings.
    if isinstance(arg1conv, str) or isinstance(arg2conv, str):
        arg1conv = str(arg1conv)
        arg2conv = str(arg2conv)
    return arg1conv + arg2conv

# The 'multiply3' function itself
def multiply3(arg1, arg2, arg3):
    # Convert 'arg1' and 'arg2' to their appropriate types
    arg1conv = conv(arg1)
    arg2conv = conv(arg2)
    arg3conv = conv(arg3)
    # If either 'arg1' or 'arg2 or arg3' is a string, ensure they're both strings.
    if isinstance(arg1conv, str) or isinstance(arg2conv, str) or isinstance(arg3conv, str):
        arg1conv = str(arg1conv)
        arg2conv = str(arg2conv)
        arg3conv = str(arg3conv)
    return arg1conv * arg2conv * arg3conv
```

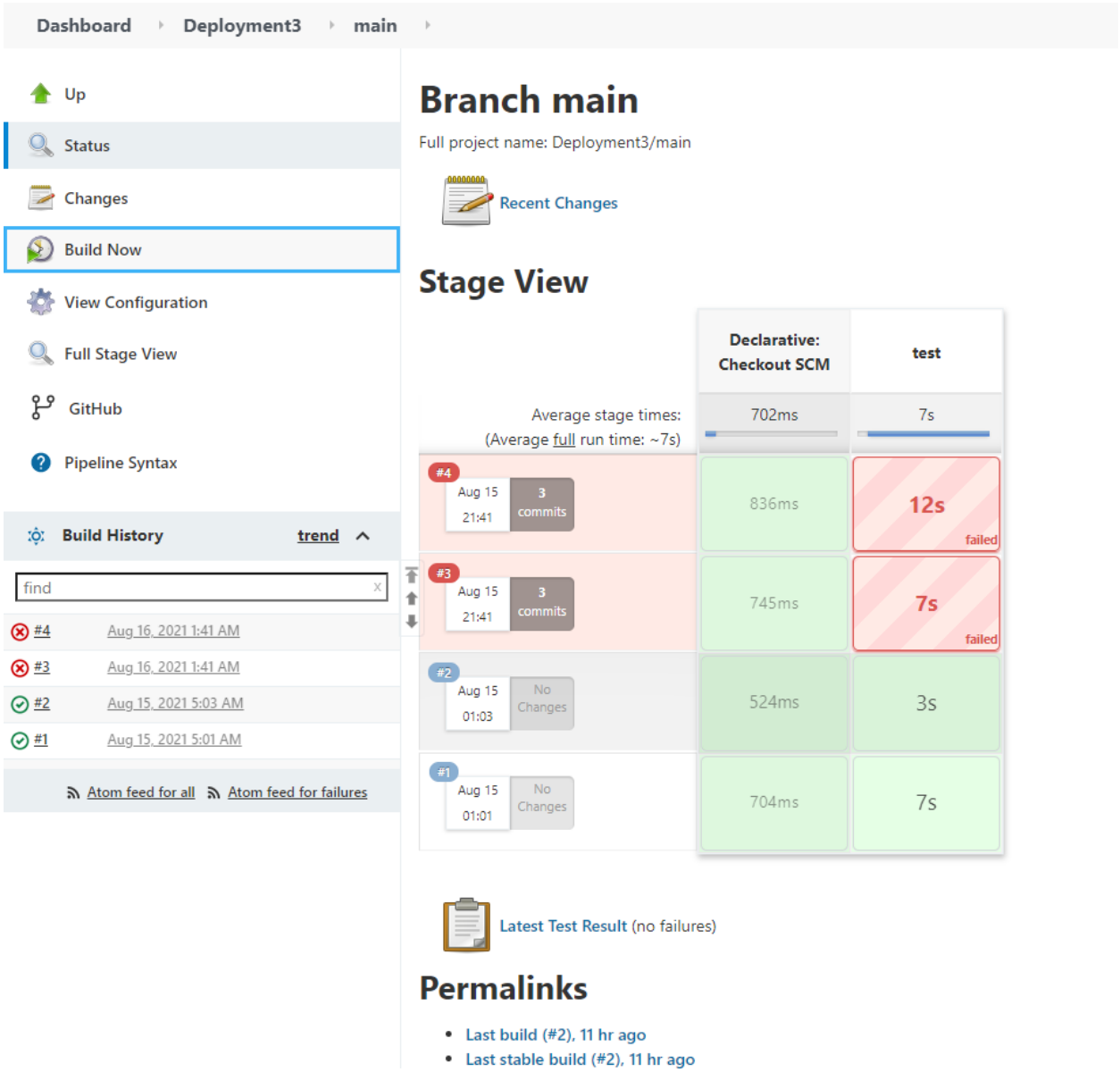
8. I then added the methods to the test_calc.py file to test my code and try multiplying different types of variables. I knew you could multiply integers with each other. And multiplying floats was possible as well. However I decided to test out multiplying strings and also tried multiplying strings with integers.

```
import unittest
import calc

class TestCalc(unittest.TestCase):
    """
    Test the add function from the calc library
    """

    def test_add_integers(self):
        """
        Test that the addition of two integers returns the correct total
        """
        result = calc.add2(1, 2)
        self.assertEqual(result, 3)

    def test_add_floats(self):
        """
        Test that the addition of two floats returns the correct result
        """
        result = calc.add2('10.5', 2)
        self.assertEqual(result, 12.5)

    def test_add_strings(self):
        """
        Test the addition of two strings returns the two strings as one
        concatenated string
        """
        result = calc.add2('abc', 'def')
        self.assertEqual(result, 'abcdef')

    def test_add_string_and_integer(self):
        """
        Test the addition of a string and an integer returns them as one
        concatenated string (in which the integer is converted to a string)
        """
        result = calc.add2('abc', 3)
        self.assertEqual(result, 'abc3')

    def test_add_string_and_number(self):
        """
        Test the addition of a string and a float returns them as one
        concatenated string (in which the float is converted to a string)
        """
        result = calc.add2('abc', '5.5')
```

```
#multiply 3

 def test_am_integers(self):
    """
    Test that the multiplication of two integers returns the correct total
    """
    result = calc.multiply3(1, 2,3)
    self.assertEqual(result, 6)

def test_multiply_floats(self):
    """
    Test that the multiplication of two floats returns the correct result
    """
    result = calc.multiply3('10.5', 2, 11)
    self.assertEqual(result, 231)

def test_multiply_strings(self):
    """
    Test the multiplication of two strings returns the two strings as one
    concatenated string
    """
    result = calc.multiply3('abc', 'def', 'ghi')
    self.assertEqual(result, 'abcdefghi')

def test_multiply_string_and_integer(self):
    """
    Test the multiplication of a string and an integer returns them as one
    concatenated string (in which the integer is converted to a string)
    """
    result = calc.multiply3('abc', 3)
    self.assertEqual(result, 'abc3abc3abc3')
```

9. Testing if multiplying strings and also multiplying a string with an integer failed, so I removed those methods from test_calc.py.

Screenshot of first fail of test build
failed multiplying strings and also multiplying a string with a integer



To fix this i erased

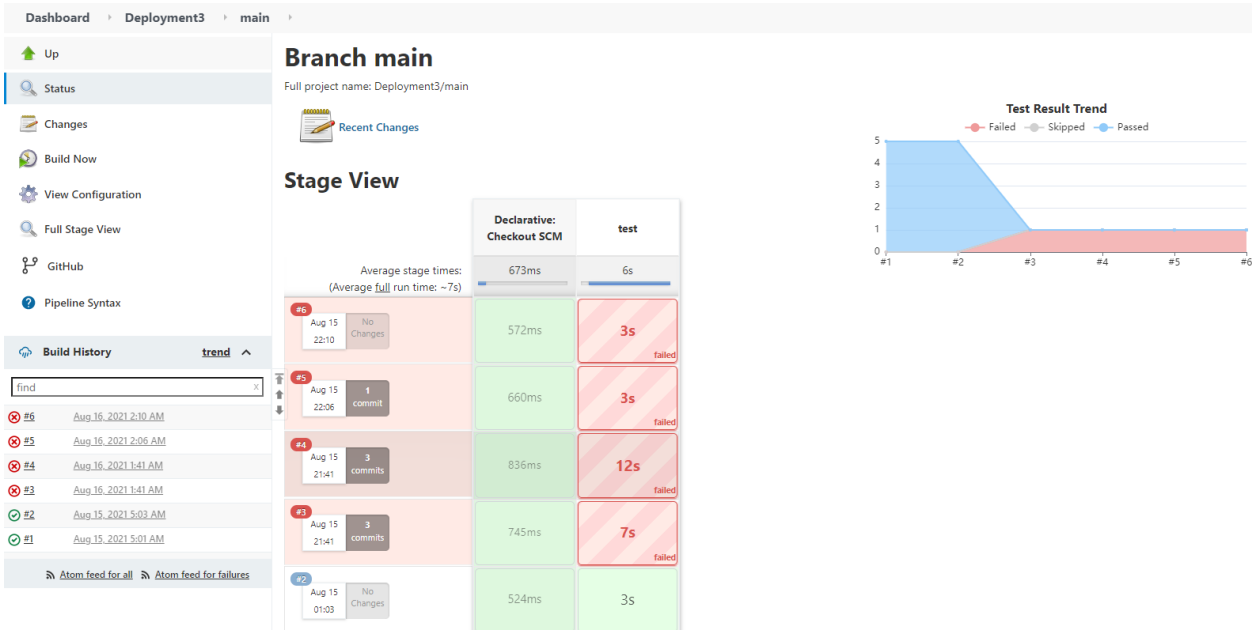
```
62
63      def test_multiply_strings(self):
64          """
65          Test the multiplication of two strings returns the two strings as one
66          concatenated string
67          """
68          result = calc.multiply3('abc', 'def', 'ghi')
69          self.assertEqual(result, 'abcdefghi')
70
71      def test_multiply_string_and_integer(self):
72          """
73          Test the multiplication of a string and an integer returns them as one
74          concatenated string (in which the integer is converted to a string)
75          """
76          result = calc.multiply3('abc', 3)
77          self.assertEqual(result, 'abc3abc3abc3')
78
```

This is what I had left

```
47      #multiply 3
48
49       def test_am_integers(self):
50          """
51          Test that the multiplication of three integers returns the correct total
52          """
53          result = calc.multiply3(1, 2,3)
54          self.assertEqual(result, 6)
55
56      def test_multiply_floats(self):
57          """
58          Test that the multiplication of three floats returns the correct result
59          """
60          result = calc.multiply3('10.5', 2, 11)
61          self.assertEqual(result, 231)
62
63  if __name__ == '__main__':
64      unittest.main()
65
```

10. After removing the multiplying strings method and multiplying a string with a int , it still did not work and the stage view gave me an error.

2nd screenshot of failed test build (issue was indentation)




```
#multiply 3

 def test_am_integers(self):
    """
    Test that the multiplication of three intege
    """
    result = calc.multiply3(1, 2,3)
    self.assertEqual(result, 6)

def test_multiply_floats(self):
    """
    Test that the multiplication of three floats
```

11. To fix this I clicked on the error and went to console output, and it said I had a error on line 49.

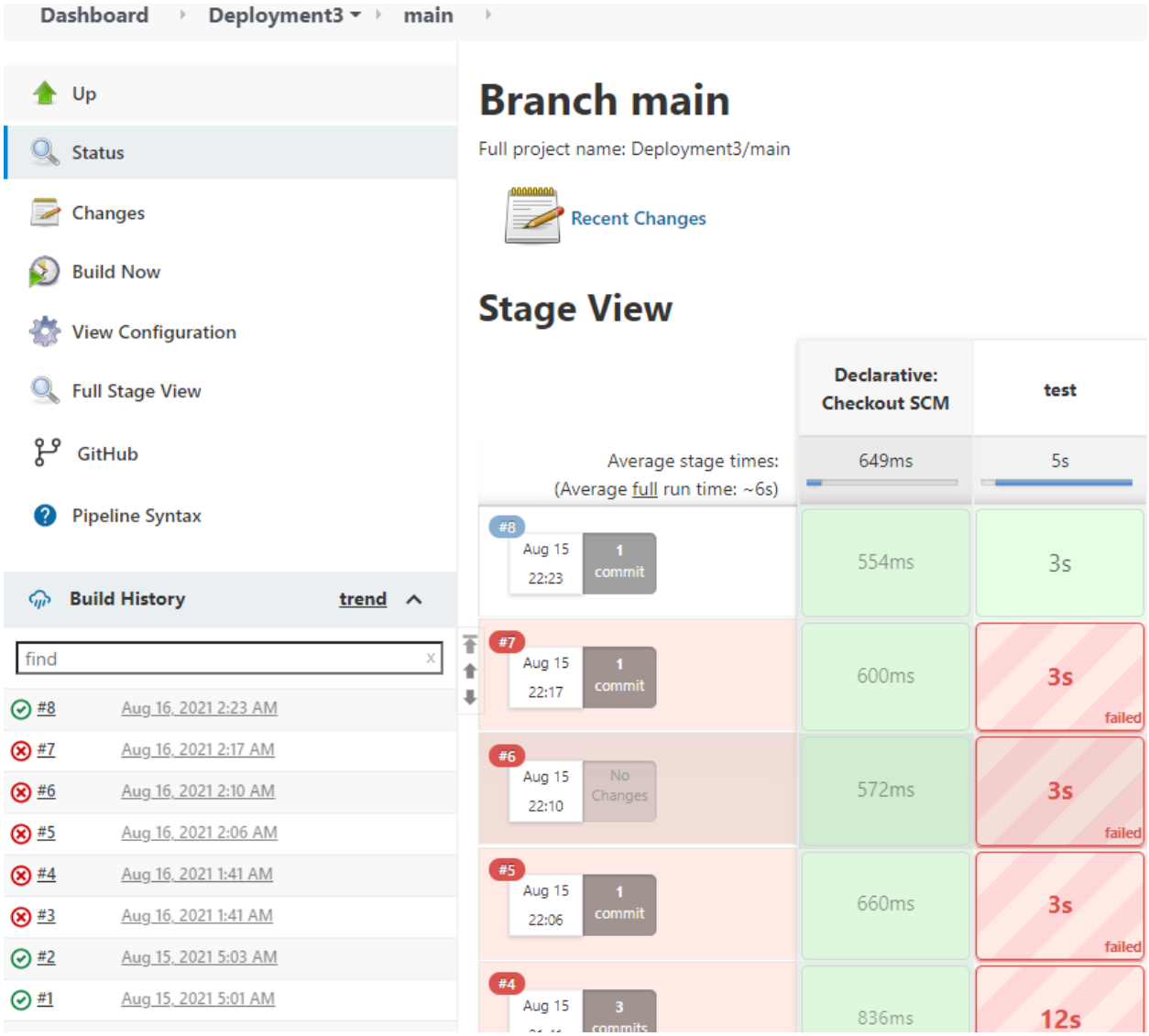
```
#multiply 3

 def test_am_integers(self):
     """
     Test that the multiplication of three integers returns the correct total
     """
     result = calc.multiply3(1, 2, 3)
     self.assertEqual(result, 6)

def test_multiply_floats(self):
     """
     Test that the multiplication of three floats returns the correct result
     """
     result = calc.multiply3('10.5', 2, 11)
     self.assertEqual(result, 231)
```

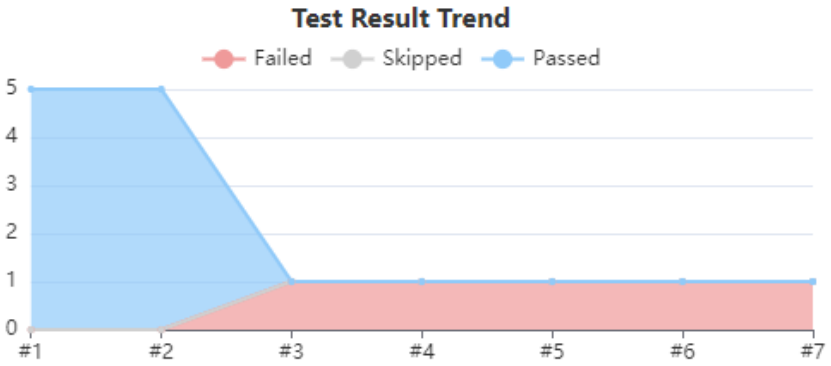
```
<frozen importlib._bootstrap>:677: in _load_unlocked
    ???
test3/lib64/python3.7/site-packages/_pytest/assertion/rewrite.py:161: in exec_module
    source_stat, co = _rewrite_test(fn, self.config)
test3/lib64/python3.7/site-packages/_pytest/assertion/rewrite.py:354: in _rewrite_test
    tree = ast.parse(source, filename=fn_)
/usr/lib64/python3.7/ast.py:35: in parse
    return compile(source, filename, mode, PyCF_ONLY_AST)
E     File "/var/lib/jenkins/workspace/Deployment3_main/sources/test_calc.py", line 49
E       def test_am_integers(self):
E                                  ^
E   IndentationError: unindent does not match any outer indentation level
- generated xml file: /var/lib/jenkins/workspace/Deployment3_main/test-reports/results.xml -
=========================== short test summary info ============================
ERROR sources/test_calc.py
!!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!!!!!!!!!!!!!
=============================== 1 error in 0.20s ===============================
Post stage
[Pipeline] junit
Recording test results
[Checks API] No suitable checks publisher found.
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
ERROR: script returned exit code 2

GitHub has been notified of this commit’s build result

Finished: FAILURE
```

\Line 49 had indentation error

12. To fix this and make sure I had no more indentation errors, I copied my code to visual studio code which had an indentation guide. This guide helped me fix my mistake and make sure there was no other indentation error in any of my code
13. Then I tried to test my build or code again and it worked.

Last successful test build




Test Result Trend
Failed
Skipped
Passed
5
4
3
2
1
0
#1
#2
#3
#4
#5
#6
#7