

Simple DevOps Pipeline Description

Web application:

At first let's create a simple server using Python, Flask and create index.html file in templates folder to have a form where user can enter some sort of input.

app.py server has a home route that will display the index.html form on GET request and on POST request it will redirect to /greet url with the given input. /greet url simply will display "Hello , username!".

app/app.py

Continuous Integration:

We need to set up a CI pipeline using Github Actions.

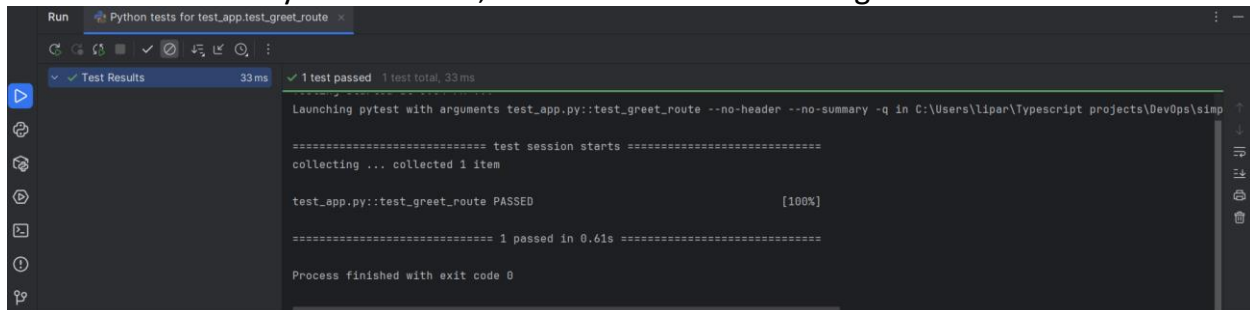
Firstly, it will set up python, install dependencies and then run tests using "pytest". It will work everytime we push or open pull request on main or dev branch.

.github/workflows/python-app.yml

But first we need to have some sort of test script.

So we can create tests folder and inside test_app.py.

If we run it manually in the editor, it should return something like this:

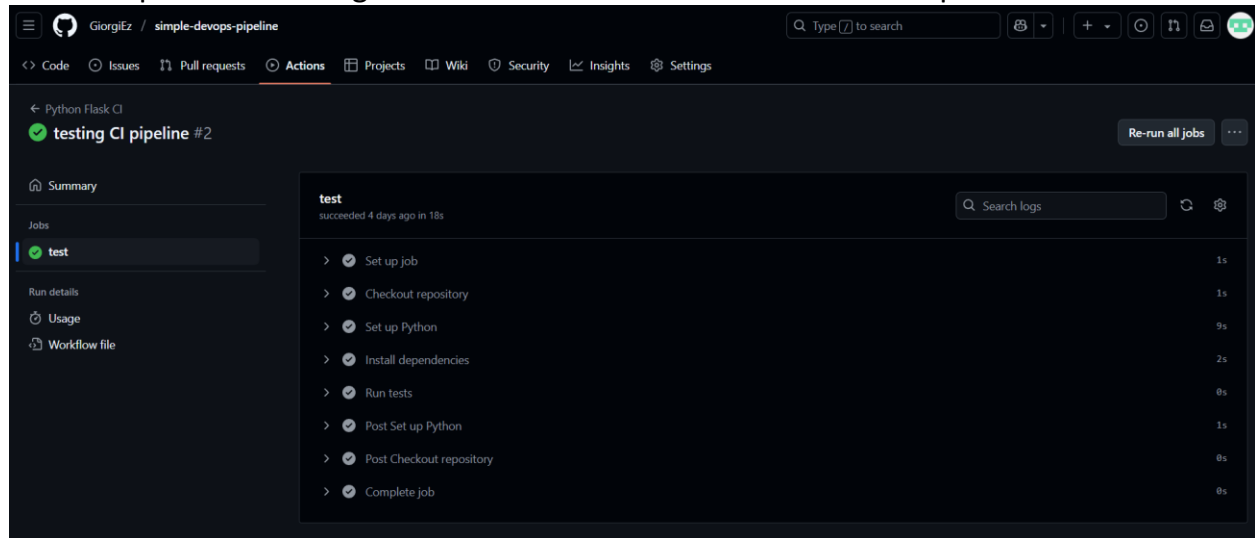
A screenshot of a code editor's terminal window showing the output of a pytest command. The terminal title is "Run Python tests for test_app.test_greet_route". The output indicates that the test passed successfully. The text in the terminal is as follows:

```
Test Results 33ms ✓ 1 test passed 1 test total, 33ms
Launching pytest with arguments test_app.py::test_greet_route --no-header --no-summary -q in C:\Users\lipar\Typescript projects\DevOps\simp
===== test session starts =====
collecting ... collected 1 item

test_app.py::test_greet_route PASSED [100%]

===== 1 passed in 0.61s =====
Process finished with exit code 0
```

Now to make sure the workflow works we need to commit to a dev branch and see if the tests pass. On the github UI we should be able to see that it passed the test.



Continuous Deployment and Infrastructure as Code:

1. Firstly we need to create a production folder where we deploy blue and green servers.
2. Also have a scripts folder, where we store .bat files to run server, stop server or check if server is running.
3. Lastly a terraform folder, where we save 4 different terraform scripts:

1. **blue:** script installs dependencies, copies the app directory inside production/blue folder and runs the blue server on the port 5000 by using the run server batch file.
2. **green:** script installs dependencies, copies the app directory inside production/green folder and runs the green server on the port 5001 by using the run server batch file.
3. **switch_to_green:** uses batch file to stop the blue server running on the port 5000 and runs the green server on port 5000.
4. **switch_to_green:** uses batch file to stop the green server running on the port 5000 and runs the green server on port 5000.

Blue-Green deployment can be simulated in the following way:

1. Blue is a live server
2. We make changes to the development app.py file
3. Run the green.tf file to run the green server on the 5001 port
4. Make sure it works properly.
5. Run switch_to_green.tf file to stop blue running on port 5000 and run green on it.
6. If we want to rollback, we can run switch_to_blue.tf to stop green and run blue.

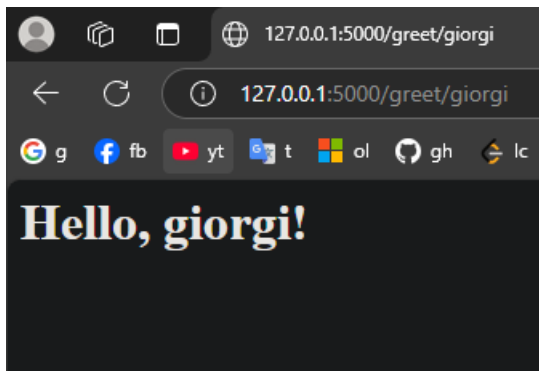
First we run the apply blue and we should see something like this in the console.

```
null_resource.copy_everything_to_blue: Destroying... [id=4459661834979264321]
null_resource.install_dependencies_blue: Destroying... [id=4575669685582367794]
null_resource.start_server_blue: Destroying... [id=3893904780587742565]
null_resource.install_dependencies_blue: Destruction complete after 0s
null_resource.copy_everything_to_blue: Destruction complete after 0s
null_resource.start_server_blue: Destruction complete after 0s
null_resource.start_server_blue: Creating...
null_resource.install_dependencies_blue: Creating...
null_resource.copy_everything_to_blue: Creating...
null_resource.install_dependencies_blue: Provisioning with 'local-exec'...
null_resource.copy_everything_to_blue: Provisioning with 'local-exec'...
null_resource.start_server_blue: Provisioning with 'local-exec'...
null_resource.start_server_blue (local-exec): Executing: ["cmd" "/C" "..\\..\\scripts\\run_server.bat blue 5000"]
null_resource.install_dependencies_blue (local-exec): Executing: ["cmd" "/C" "pip install -r ..\\..\\requirements.txt"]
null_resource.copy_everything_to_blue (local-exec): Executing: ["cmd" "/C" "xcopy /E /Y /I ..\\..\\app ..\\..\\production\\blue"]
null_resource.start_server_blue (local-exec): blue server running on port 5000...
null_resource.copy_everything_to_blue (local-exec): ..\\..\\app\\app.py
null_resource.copy_everything_to_blue (local-exec): ..\\..\\app\\__init__.py
null_resource.copy_everything_to_blue (local-exec): ..\\..\\app\\__pycache__\\app.cpython-312.pyc
null_resource.copy_everything_to_blue (local-exec): ..\\..\\app\\__pycache__\\__init__.cpython-312.pyc
null_resource.copy_everything_to_blue (local-exec): 4 File(s) copied
null_resource.copy_everything_to_blue: Creation complete after 1s [id=2976897540442284368]
null_resource.start_server_blue (local-exec): INFO:waitress:Serving on http://127.0.0.1:5000
```

In the browser we should see this



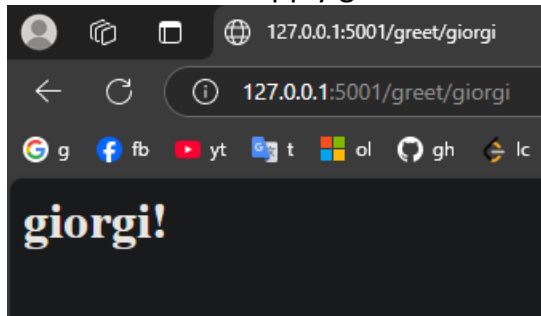
and if we enter some name this:



Now lets make change to the app/app.py file, lets remove "Hello" from greet function:

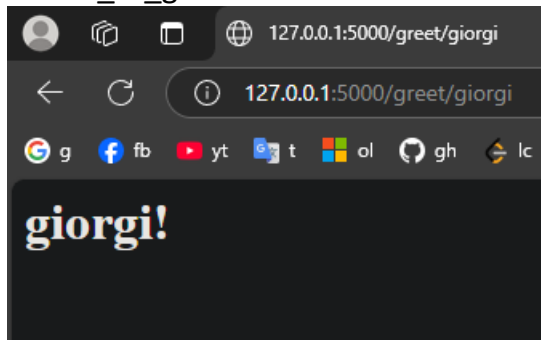
```
def greet(username):
    return f<h1>{username}!</h1>
```

And now lets run apply green command:



As we can see green server is running on port 5001 and returns just the entered name.

Now we want to run this update on the main port 5000, we need to run the apply switch_to_green terraform command:



As we can see by just restarting the browser the updated green server is running on the port 5000 and blue server has been stopped.

If we want to roll back the changes we can simply run the apply switch_to_blue to run the old blue server on the main port again.

Monitoring Health Check:

We can check if the server is up by using the health_check.bat file that takes port as the input and checks if the status is 200, the result is written in the health_log.txt file.

We need to cd into scripts folder from the main directory and run:

`.\health_check.bat 5000`

to check if the server is running on the port 5000. we should see something like this.

