47. Selenium Grid

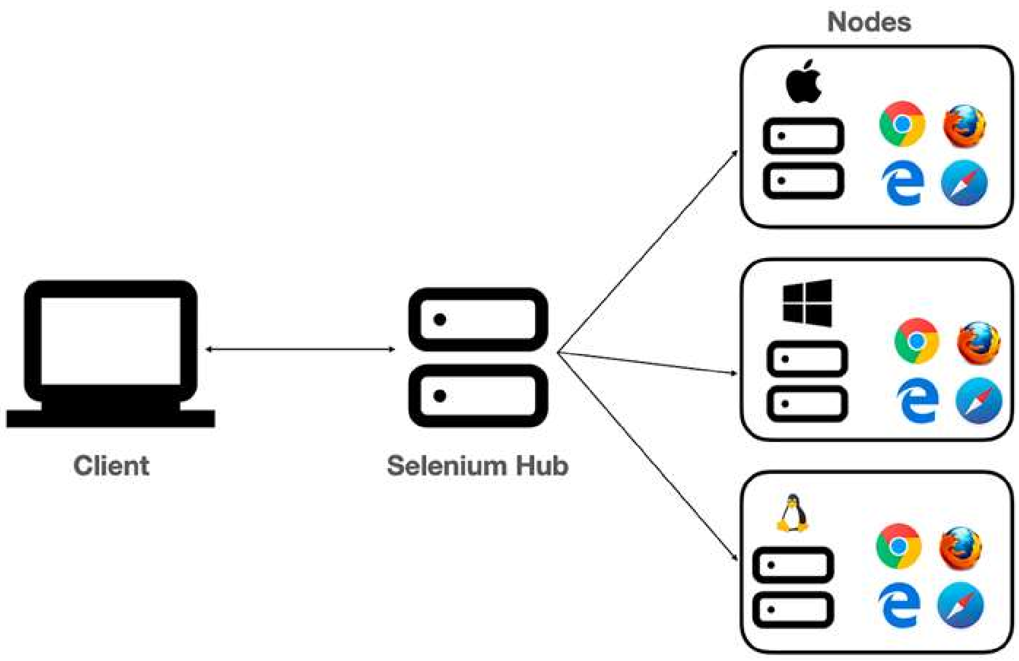
Introduction

Selenium Grid allows the execution of WebDriver scripts on remote machines by routing commands sent by the client to remote browser instances.

- ➔ Provide an easy way to run tests in parallel on multiple machines
- ➔ Allow testing on different browser versions
- ➔ Enable cross platform testing

Selenium Grid Setup

- ➔ Selenium Grid works on Hub and Node concept.
    - ◆ **Hub (From where we have written test cases, executing test cases, controlling entire setup)**
    - ◆ **Node (Remote machine from where we want to execute our Test Cases on which browser and OS)**



- ➔ Hub is setup in the client machine itself.
- ➔ Through grid setup we can attach all Nodes to hub.From hub each and evey node will be controlled.
- ➔ Entire Grid should be connected to single network.In real time in companies we will have one single network where we will setup all..
- ➔ When we run test cases from the hub in the hub itself we will specify which OS and Browser we want to run our Test Cases.
- ➔ Based on that Hub will identify the node with that configuration and sends that test cases.

Note

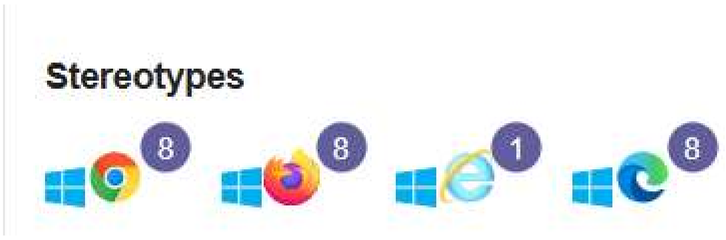
- ➔ In real time environment we can create nodes
    - ◆ By multiple Virtual Mchines since Physical machines are very costly and that too for temporary use only for Testing purposes.
    - ◆ By downloading Docker Images from Docker hub which is a remote repository.By docker images we can create a container which is having all setup.we will make that container as a node.

Standalone Setup (Single machine)

1. Download **selenium-server-4.xx.x.jar** and place it somewhere.
2. Run below command to start Selenium Grid
   a. **java -jar selenium-server-4.xx.x.jar standalone**
3. URL to see sessions: **http://localhost:4444/**

Hub & Node Setup (Multiple machines) or Distributed Setup

1. Download **selenium-server-4.xx.x.jar** and place it somewhere in both (hub & node) the machines.
2. Run below command to make machine as hub **java -jar selenium-server-4.xx.x.jar hub**
3. Run below command to make machine as node
   a. java -jar selenium-server-4.xx.x.jar node --hub http://<hub-ip>:4444
   b. Example
      i. java -jar '/home/kmr/Desktop/selenium-server-4.25.0.jar' node --hub http://192.168.1.6:4444 --selenium-manager true
4. URL to see sessions: **http://localhost:4444/**



Observations

➔ 8 no of sessions we can run parallely in grid environment.
➔ If different nodes are available to same hub it will show in the dashboard of hub

Note

➔ The URL will be IP Address of Hub Machine + Hub Port + /wd/hub
➔ **example**
   ◆ "http://192.168.13.1:4444/wd/hub" **or** "http://localhost:4444/wd/hub"

SeleniumGridDemo.py

```
from selenium import webdriver
hub_url = "http://localhost:4444/wd/hub"
cap = webdriver.ChromeOptions()
cap.add_experimental_option("detach",True)
cap.platform_name = "WIN10"  # Platform name
cap.browser_name = "chrome"  # Browser name
driver = webdriver.Remote(command_executor=hub_url, options=cap)
driver.get("https://www.google.com")
print(driver.title)
driver.quit()
```

conftest.py

```
import pytest
from selenium import webdriver
```

```
@pytest.fixture()
def setup(browser_platform):
  browser,platform = browser_platform
  options = {
     "chrome": webdriver.ChromeOptions,
     "edge": webdriver.EdgeOptions,
     "firefox": webdriver.FirefoxOptions
  }
  if browser not in options:
     raise ValueError(f"Unsupported browser: {browser}")
  platform_mapping = {"windows": "WIN10", "mac": "MAC", "linux": "LINUX"}
  platform_name = platform_mapping.get(platform)
  if not platform_name:
     raise ValueError(f"Unsupported platform: {platform}")
  opt = options[browser]()
  opt.add_experimental_option("detach", True) if browser in ["chrome", "edge"] else None
  opt.platform_name = platform_name
  driver = webdriver.Remote(command_executor="http://localhost:4444/wd/hub", options=opt)
  yield driver
  driver.quit()
```

<u>Hook to add command-line options for browser and OS</u>

```
def pytest_addoption(parser):
  parser.addoption("--browser", default="chrome", choices=["chrome", "edge", "firefox"], help =
"Browser to test")
  parser.addoption("--os", default="windows", choices=["windows", "mac", "linux"], help =
"Operating system to test")
```

<u>Get value from command Line</u>

```
@pytest.fixture()
def browser_platform(request):
  browser = request.config.getoption("--browser")
  platform = request.config.getoption("--os")
  return browser,platform
```

<u>test_Parallel.py</u>

```
class TestTitle:
  def test_title_chrome(self,setup):
     driver = setup
     driver.get("https://www.google.com/")
     assert driver.title == "Google"  # validation
  def test_title_edge(self,setup):
     driver = setup
     driver.get("https://www.google.com/")
     assert driver.title == "Google"  # validation
  def test_title_firefox(self,setup):
      driver = setup
      driver.get("https://www.google.com/")
      assert driver.title == "Google"  # validation
```

Command to Execute

➔ `pytest -s -v --os=windows --browser=chrome -n 3 test_Parallel.py`