Assignment: Containerizing Python Script and Scaling with Selenium Grid

Objective:

The objective of this assignment is to evaluate your skills in containerization using Docker and scaling a Python script using Selenium Grid. You'll be required to containerize a generalized Python script for website scraping, set up a Selenium Grid environment, and demonstrate the ability to scale the scraper using Docker and AWS.

Instructions:

1. **Study the given sample python script made using selenium python**

   1.1. The script goes on the site **https://quotes.toscrape.com/**
   1.2. It scrapes the quotes on pages 1 to 10.
   1.3. The current script iterates over ten pages sequentially. Your assignment is to modify it to scrape all pages concurrently, rather than using a loop.
   To achieve this, you are advised to utilize Selenium Grid and multiple instances.
   Your objective is to optimize the scraping process and minimize the overall execution time.

2. **Containerize the Python script:**

   2.1. Create a Dockerfile to package the Python script into a Docker container.
   2.2. Ensure the Docker image includes all necessary dependencies and libraries required by the script.
   2.3. Build the Docker image and test it locally to verify that the script runs successfully within the container.

3. **Set up a Selenium Grid environment:**

   3.1. Familiarize yourself with Selenium Grid and its architecture.
   3.2. Design and implement a Selenium Grid infrastructure using Docker.
   3.3. Include one or more Selenium Hub nodes and multiple Selenium nodes (e.g., Chrome or Firefox) in the setup.
   3.4. Ensure the Selenium Grid is properly configured to handle multiple concurrent script executions.

4. **Scale the scraper using Docker and AWS:**

   4.1. Deploy the Docker containerized Python script on AWS.
   4.2. Utilize AWS services (e.g., Amazon EC2, Amazon ECS, or Amazon EKS) to manage and orchestrate Docker containers.

4.3. Implement the necessary infrastructure and configurations to enable horizontal scaling of the scraper across multiple Docker containers.
4.4. Validate that the scraper can effectively distribute the workload across the containers and handle increased traffic.
4.5. Ensure that the data is being accurately stored in your cloud-based MongoDB Atlas.

5. **Documentation and Presentation:**

5.1. Document the steps you followed, including any challenges faced and solutions implemented.
5.2. Provide clear instructions on how to reproduce your setup and deployment process.
5.3. Prepare a presentation summarizing your approach, highlighting the key decisions made, and showcasing the final solution.

Submission Guidelines:

- Share the Dockerfile, relevant configuration files, and any scripts used.
- Provide detailed documentation and instructions on setting up the environment.
- Prepare a presentation (slides, video, or live demo) for showcasing your solution.

Evaluation Criteria:

- Successfully containerized the generalized Python script in a Docker container.
- Correct setup and configuration of the Selenium Grid environment.
- Demonstration of horizontal scaling of the scraper using Docker and AWS.
- Documentation quality, clarity, and completeness.
- Presentation skills and the ability to effectively communicate the approach and results.
- **Able to scrape the 10 pages in the lowest possible time with the help of multiple docker containers using AWS technologies.**

Python Selenium Script:

```python
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options
from pymongo import MongoClient

chrome_options = Options()
chrome_options.add_argument("--disable-gpu")
chrome_options.add_argument("--headless")

driver = webdriver.Chrome(options=chrome_options)
driver.get("http://quotes.toscrape.com")

client = MongoClient('mongodb://localhost:27017/')
db = client['mcs_assignment']
collection = db['quotes']

for page in range(10):
    for i in range(1, 11):
        quote_xpath = f"/html/body/div[1]/div[2]/div[1]/div[{i}]/span[1]"
        author_xpath =
f"/html/body/div[1]/div[2]/div[1]/div[{i}]/span[2]/small"

        quote_element = driver.find_element(By.XPATH, quote_xpath)
        quote_text = quote_element.text

        author_element = driver.find_element(By.XPATH, author_xpath)
        author_name = author_element.text

        print("Quote:", quote_text)
        print("Author:", author_name)
        print()

        document = {
            'quote_id': page * 10 + i,
            'quote': quote_text,
            'author': author_name
        }

        collection.insert_one(document)

    try:
        next_button = driver.find_element(By.XPATH, "//li[@class='next']/a")
        next_button.click()
    except:
        print("No more pages available. Exiting loop.")
        break
driver.quit()
```