**Uso de Docker en Pruebas Automatizadas con Python y Selenium**

**1. Introducción a Docker y Docker Compose**

Docker es una herramienta que permite la creación, despliegue y ejecución de aplicaciones en contenedores. Docker Compose, por otro lado, facilita la gestión de múltiples contenedores a través de un archivo de configuración YAML.

**2. Instalación de Docker**

**2.1 Instalación en Windows**

1. Descargar Docker Desktop.
2. Instalar y habilitar WSL 2 si es necesario.
3. Reiniciar y verificar la instalación con:

docker --version

**2.2 Instalación en macOS**

1. Descargar e instalar Docker Desktop.
2. Verificar con:

docker --version

**2.3 Instalación en Linux (Ubuntu/Debian)**

sudo apt update

sudo apt install -y docker.io

sudo systemctl enable --now docker

sudo usermod -aG docker $USER

newgrp docker

Verificar con:

docker --version

docker run hello-world

**3. Diferencias entre Docker y Docker Compose**

|  |  |  |
| --- | --- | --- |
| **Característica** | **Docker** | **Docker Compose** |
| Ejecuta un solo contenedor | ✅ Sí | ❌ No |
| Ejecuta múltiples contenedores | 🚧 Manualmente | ✅ Fácilmente |
| Dependencias entre contenedores | Manual | Automático (depends\_on) |
| Archivo de configuración | No requiere | docker-compose.yml |

**4. Uso de Docker con Selenium**

**4.1 Ejecutar Selenium Standalone con Docker**

docker run -d --name selenium-chrome -p 4444:4444 selenium/standalone-chrome

Luego, en Python:

from selenium import webdriver

options = webdriver.ChromeOptions()

driver = webdriver.Remote(

command\_executor="http://localhost:4444/wd/hub",

options=options

)

driver.get("https://www.google.com")

print(driver.title)

driver.quit()

**4.2 Uso de Docker Compose para Selenium Grid con Múltiples Instancias de Navegadores**

Archivo docker-compose.yml:

version: "3"

services:

selenium-hub:

image: selenium/hub

container\_name: selenium-hub

ports:

- "4444:4444"

chrome:

image: selenium/node-chrome

depends\_on:

- selenium-hub

environment:

- SE\_EVENT\_BUS\_HOST=selenium-hub

- SE\_EVENT\_BUS\_PUBLISH\_PORT=4442

- SE\_EVENT\_BUS\_SUBSCRIBE\_PORT=4443

deploy:

replicas: 2 # Ejecutar 2 instancias de Chrome

firefox:

image: selenium/node-firefox

depends\_on:

- selenium-hub

environment:

- SE\_EVENT\_BUS\_HOST=selenium-hub

- SE\_EVENT\_BUS\_PUBLISH\_PORT=4442

- SE\_EVENT\_BUS\_SUBSCRIBE\_PORT=4443

deploy:

replicas: 2 # Ejecutar 2 instancias de Firefox

Ejecutar con:

docker-compose up -d

**5. Integración con Pytest**

Crear conftest.py:

import pytest

from selenium import webdriver

@pytest.fixture(scope="function")

def browser():

options = webdriver.ChromeOptions()

driver = webdriver.Remote(

command\_executor="http://localhost:4444/wd/hub",

options=options

)

yield driver

driver.quit()

Crear prueba test\_google.py:

def test\_google\_title(browser):

browser.get("https://www.google.com")

assert "Google" in browser.title

Ejecutar:

pytest test\_google.py

**6. Integración en CI/CD con GitHub Actions**

Crear .github/workflows/test.yml:

name: Selenium Tests

on: [push]

jobs:

test:

runs-on: ubuntu-latest

services:

selenium:

image: selenium/standalone-chrome

ports:

- 4444:4444

steps:

- name: Checkout code

uses: actions/checkout@v3

- name: Set up Python

uses: actions/setup-python@v3

with:

python-version: "3.9"

- name: Install dependencies

run: pip install selenium pytest

- name: Run tests

run: pytest

**7. Conclusión**

Docker y Docker Compose facilitan la ejecución de pruebas automatizadas con Selenium, simplificando la configuración de navegadores y evitando problemas de compatibilidad. Usando estas herramientas, se pueden integrar las pruebas en CI/CD de manera eficiente. 🚀