Continuous Integration

Introduction

Playwright tests can be executed in CI environments. We have created sample configurations for common CI providers.

3 steps to get your tests running on CI:

1. **Ensure CI agent can run browsers**: Use our Docker image in Linux agents or install your dependencies using the CLI.
2. **Install Playwright**:

   ```
   pip install playwright
   playwright install --with-deps
   ```

3. **Run your tests**:

   ```
   pytest
   ```

CI configurations

The Command line tools can be used to install all operating system dependencies on GitHub Actions.

GitHub Actions

Check out our GitHub Actions guide for more information on how to run your tests on GitHub.

Docker

We have a pre-built Docker image which can either be used directly, or as a reference to update your existing Docker definitions.

Suggested configuration

1. Using `--ipc=host` is also recommended when using Chromium. Without it Chromium can run out of memory and crash. Learn more about this option in Docker docs.
2. Seeing other weird errors when launching Chromium? Try running your container with `docker run --cap-add=SYS_ADMIN` when developing locally.
3. Using `--init` Docker flag or dumb-init is recommended to avoid special treatment for processes with PID=1. This is a common reason for zombie processes.

Azure Pipelines

For Windows or macOS agents, no additional configuration required, just install Playwright and run your tests.

For Linux agents, you can use our Docker container with Azure Pipelines support running containerized jobs. Alternatively, you can use Command line tools to install all necessary dependencies.

For running the Playwright tests use this pipeline task:

```yaml
trigger:
- main

pool:
  vmImage: ubuntu-latest

steps:
- task: UsePythonVersion@0
  inputs:
    versionSpec: '3.11'
  displayName: 'Use Python'
- script: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
  displayName: 'Install dependencies'
- script: playwright install --with-deps
  displayName: 'Install Playwright browsers'
- script: pytest
  displayName: 'Run Playwright tests'
```

Azure Pipelines (containerized)

```
trigger:
- main

pool:
  vmImage: ubuntu-latest
container: mcr.microsoft.com/playwright/python:v1.38.0-jammy

steps:
- task: UsePythonVersion@0
  inputs:
    versionSpec: '3.11'
  displayName: 'Use Python'

- script: |
    python -m pip install --upgrade pip
    pip install -r requirements.txt
  displayName: 'Install dependencies'
- script: pytest
  displayName: 'Run tests'
```

CircleCI

Running Playwright on CircleCI is very similar to running on GitHub Actions. In order to specify the pre-built Playwright Docker image, simply modify the agent definition with `docker:` in your config like so:

```
executors:
  pw-jammy-development:
    docker:
      - image: mcr.microsoft.com/playwright/python:v1.38.0-jammy
```

Note: When using the docker agent definition, you are specifying the resource class of where playwright runs to the 'medium' tier here. The default behavior of Playwright is to set the number of workers to the detected core count (2 in the case of the medium tier). Overriding the number of workers to greater than this number will cause unnecessary timeouts and failures.

Jenkins

Jenkins supports Docker agents for pipelines. Use the Playwright Docker image to run tests on Jenkins.

```
pipeline {
   agent { docker { image 'mcr.microsoft.com/playwright/python:v1.38.0-jammy' } }
   stages {
      stage('e2e-tests') {
         steps {
            sh 'pip install -r requirements.txt'
            sh 'pytest'
         }
      }
   }
}
```

Bitbucket Pipelines

Bitbucket Pipelines can use public Docker images as build environments. To run Playwright tests on Bitbucket, use our public Docker image (see Dockerfile).

```
image: mcr.microsoft.com/playwright/python:v1.38.0-jammy
```

GitLab CI

To run Playwright tests on GitLab, use our public Docker image (see Dockerfile).

```
stages:
  - test

tests:
  stage: test
  image: mcr.microsoft.com/playwright/python:v1.38.0-jammy
  script:
  ...
```

Caching browsers

Caching browser binaries is not recommended, since the amount of time it takes to restore the cache is comparable to the time it takes to download the binaries. Especially under Linux, operating system dependencies need to be installed, which are not cacheable.

If you still want to cache the browser binaries between CI runs, cache these directories in your CI configuration, against a hash of the Playwright version.

Debugging browser launches

Playwright supports the `DEBUG` environment variable to output debug logs during execution. Setting it to `pw:browser` is helpful while debugging `Error: Failed to launch browser` errors.

```
DEBUG=pw:browser pytest
```

Running headed

By default, Playwright launches browsers in headless mode. This can be changed by passing a flag when the browser is launched.

Sync Async

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
   # Works across chromium, firefox and webkit
   browser = p.chromium.launch(headless=False)
```

On Linux agents, headed execution requires Xvfb to be installed. Our Docker image and GitHub Action have Xvfb pre-installed. To run browsers in headed mode with Xvfb, add `xvfb-run` before the Node.js command.

```
xvfb-run python test.py
```