Retries

Introduction

Test retries are a way to automatically re-run a test when it fails. This is useful when a test is flaky and fails intermittently. Test retries are configured in the configuration file.

Failures

Playwright Test runs tests in worker processes. These processes are OS processes, running independently, orchestrated by the test runner. All workers have identical environments and each starts its own browser.

Consider the following snippet:

```
import { test } from '@playwright/test';

test.describe('suite', () => {
  test.beforeAll(async () => { /* ... */ });
  test('first good', async ({ page }) => { /* ... */ });
  test('second flaky', async ({ page }) => { /* ... */ });
  test('third good', async ({ page }) => { /* ... */ });
  test.afterAll(async () => { /* ... */ });
});
```

When **all tests pass**, they will run in order in the same worker process.

- Worker process starts
  - `beforeAll` hook runs
  - `first good` passes
  - `second flaky` passes
  - `third good` passes
  - `afterAll` hook runs

Should **any test fail**, Playwright Test will discard the entire worker process along with the browser and will start a new one. Testing will continue in the new worker process starting with the next test.

- Worker process #1 starts
  - `beforeAll` hook runs

    - `first good` passes
    - `second flaky` fails
    - `afterAll` hook runs
- Worker process #2 starts
    - `beforeAll` hook runs again
    - `third good` passes
    - `afterAll` hook runs

If you **enable retries**, second worker process will start by retrying the failed test and continue from there.

- Worker process #1 starts
    - `beforeAll` hook runs
    - `first good` passes
    - `second flaky` fails
    - `afterAll` hook runs
- Worker process #2 starts
    - `beforeAll` hook runs again
    - `second flaky` is retried and passes
    - `third good` passes
    - `afterAll` hook runs

This scheme works perfectly for independent tests and guarantees that failing tests can't affect healthy ones.

Retries

Playwright supports **test retries**. When enabled, failing tests will be retried multiple times until they pass, or until the maximum number of retries is reached. By default failing tests are not retried.

```
# Give failing tests 3 retry attempts
npx playwright test --retries=3
```

You can configure retries in the configuration file:

playwright.config.ts
```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  // Give failing tests 3 retry attempts
  retries: 3,
});
```

Playwright Test will categorize tests as follows:

- "passed" - tests that passed on the first run;
- "flaky" - tests that failed on the first run, but passed when retried;
- "failed" - tests that failed on the first run and failed all retries.

```
Running 3 tests using 1 worker

  ✓ example.spec.ts:4:2 › first passes (438ms)
  x example.spec.ts:5:2 › second flaky (691ms)
  ✓ example.spec.ts:5:2 › second flaky (522ms)
  ✓ example.spec.ts:6:2 › third passes (932ms)

  1 flaky
    example.spec.ts:5:2 › second flaky
  2 passed (4s)
```

You can detect retries at runtime with testInfo.retry, which is accessible to any test, hook or fixture. Here is an example that clears some server-side state before a retry.

```
import { test, expect } from '@playwright/test';

test('my test', async ({ page }, testInfo) => {
  if (testInfo.retry)
    await cleanSomeCachesOnTheServer();
  // ...
});
```

You can specify retries for a specific group of tests or a single file with test.describe.configure().

```
import { test, expect } from '@playwright/test';

test.describe(() => {
  // All tests in this describe group will get 2 retry attempts.
  test.describe.configure({ retries: 2 });

  test('test 1', async ({ page }) => {
    // ...
  });

  test('test 2', async ({ page }) => {
    // ...
  });
});
```

Serial mode

Use test.describe.serial() to group dependent tests to ensure they will always run together and in order. If one of the tests fails, all subsequent tests are skipped. All tests in the group are retried together.

Consider the following snippet that uses `test.describe.serial`:

```
import { test } from '@playwright/test';

test.describe.configure({ mode: 'serial' });

test.beforeAll(async () => { /* ... */ });
test('first good', async ({ page }) => { /* ... */ });
test('second flaky', async ({ page }) => { /* ... */ });
test('third good', async ({ page }) => { /* ... */ });
```

When running without retries, all tests after the failure are skipped:

- Worker process #1:
  - `beforeAll` hook runs
  - `first good` passes
  - `second flaky` fails
  - `third good` is skipped entirely

When running with retries, all tests are retried together:

- Worker process #1:
  - `beforeAll` hook runs
  - `first good` passes
  - `second flaky` fails
  - `third good` is skipped
- Worker process #2:
  - `beforeAll` hook runs again
  - `first good` passes again
  - `second flaky` passes
  - `third good` passes

NOTE

It is usually better to make your tests isolated, so they can be efficiently run and retried independently.

Reuse single page between tests

Playwright Test creates an isolated Page object for each test. However, if you'd like to reuse a single Page object between multiple tests, you can create your own in test.beforeAll() and close it in test.afterAll().

- TypeScript
- JavaScript

example.spec.ts

```
import { test, type Page } from '@playwright/test';

test.describe.configure({ mode: 'serial' });

let page: Page;

test.beforeAll(async ({ browser }) => {
  page = await browser.newPage();
});

test.afterAll(async () => {
  await page.close();
});

test('runs first', async () => {
  await page.goto('https://playwright.dev/');
```

```
});

test('runs second', async () => {
  await page.getByText('Get Started').click();
});
```

example.spec.js

```
// @ts-check

const { test } = require('@playwright/test');

test.describe.configure({ mode: 'serial' });

/** @type {import('@playwright/test').Page} */
let page;

test.beforeAll(async ({ browser }) => {
  page = await browser.newPage();
});

test.afterAll(async () => {
  await page.close();
});

test('runs first', async () => {
  await page.goto('https://playwright.dev/');
});

test('runs second', async () => {
  await page.getByText('Get Started').click();
});
```

Previous
Reporters

Next
Sharding

- Introduction
- Failures
- Retries
- Serial mode
- Reuse single page between tests

Learn

- Getting started
- Playwright Training
- Learn Videos
- Feature Videos

Community

- Stack Overflow
- Discord

- Twitter
- LinkedIn

More

- GitHub
- YouTube
- Blog
- Ambassadors