BROUGHT TO YOU BY QA WOLF

Mastering the Art of END-TO-END TESTING INFRASTRUCTURE

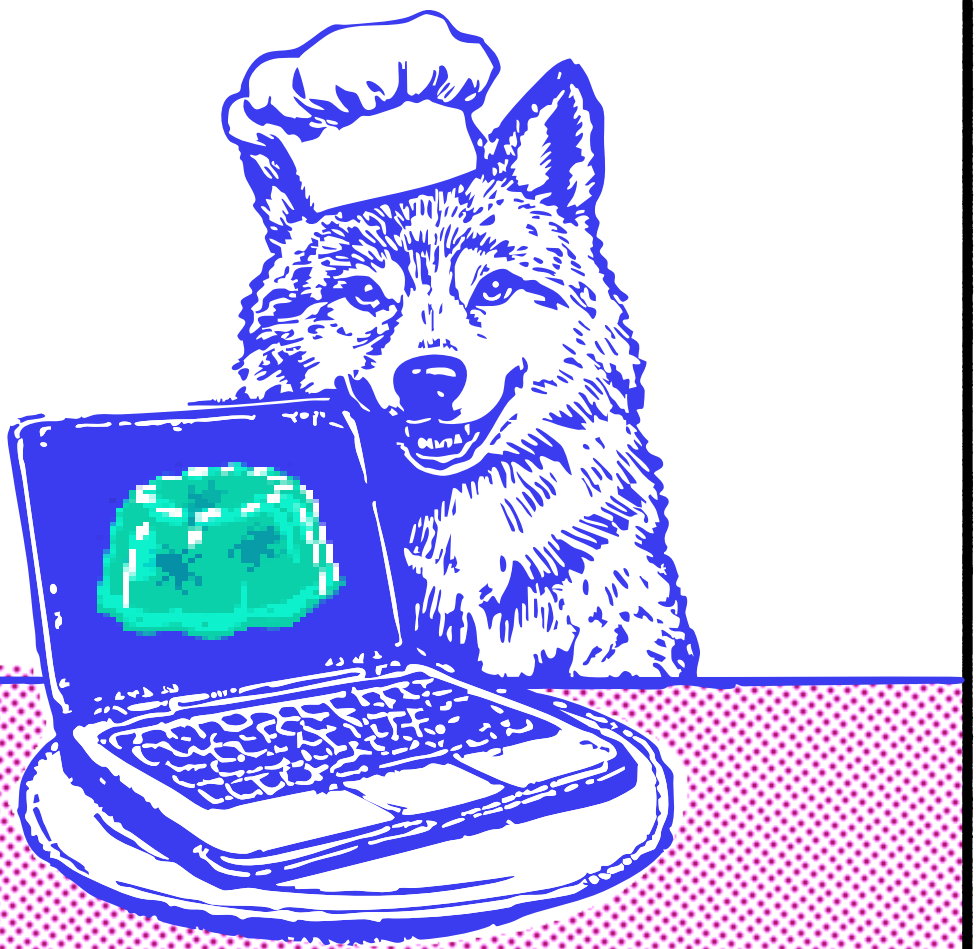


TABLE OF CONTENTS

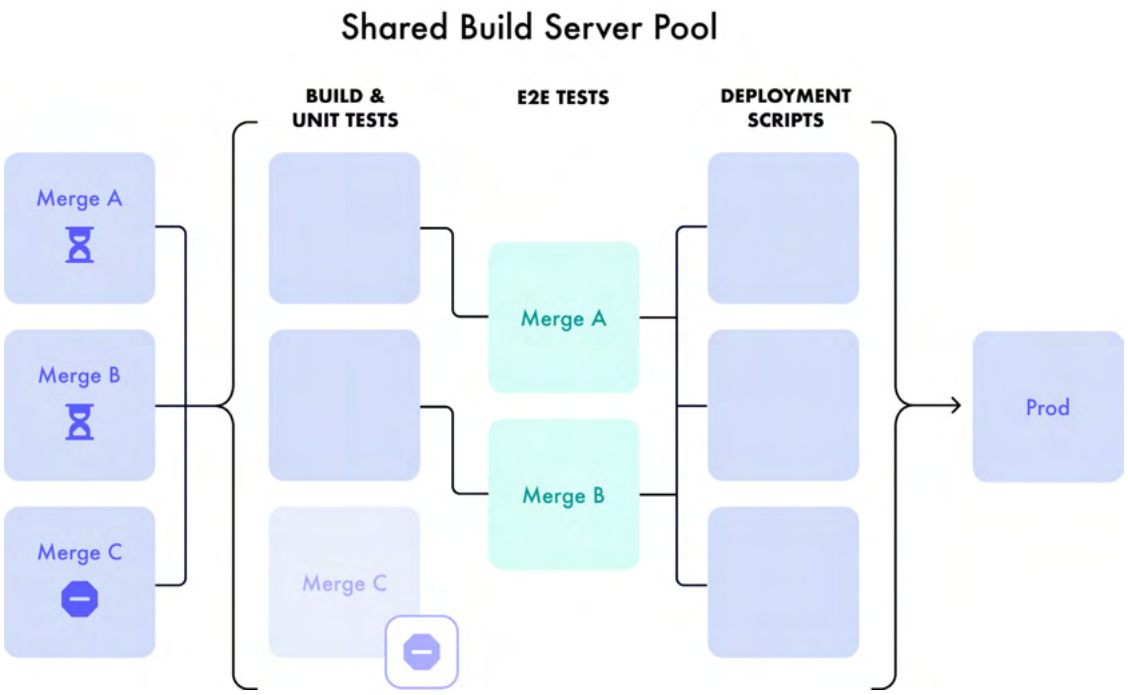
About This Book

ABOUT THIS BOOK

If the hardest part of automated end-to-end testing is investigating failures and maintaining tests as developers make changes to the application, then the second hardest part is building the test execution infrastructure needed to run them.

There is no one-size-fits-all system, and like everything in life, the system you decide to build will reflect the priorities and acceptable trade-offs for your stage of development. In this book of infrastructure recipes, we hope you will find inspiration—and code—for test run infrastructure that meets your needs.

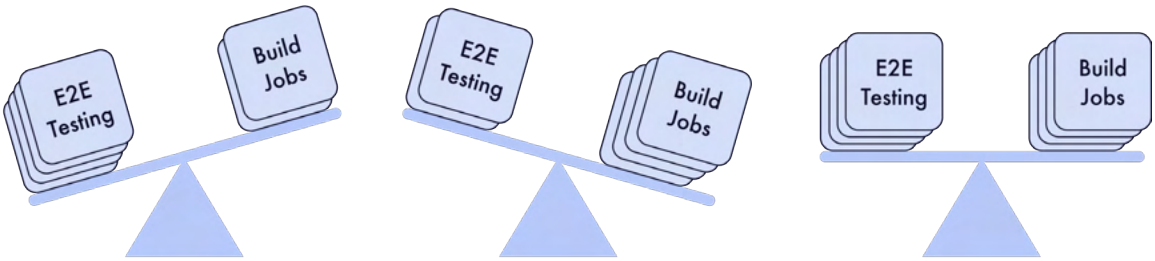
Before we explain how this book is organized and the different ways that testing infrastructure could be assembled, it's best to step back and look at where testing actually takes place: in the build servers of a CI/CD pipeline. When we understand where testing happens we can appreciate why a team's priorities and acceptable trade-offs determine which infrastructure recipe to bake.



Every CI pipeline has its unique quirks but the key bottleneck is the finite number of build servers (aka job runners or nodes) available. These build servers handle everything from compiling code artifacts to environment set up and, of course, testing. Since frequent deployments from multiple developers can monopolize those finite resources, the design of E2E testing infrastructure has to balance the speed of a test suite with the needs of every other deployment task.

WEIGHING THE TRADE OFFS

With a finite number of build servers to handle an ever-increasing number of commits, from an ever-increasing number of developers, who need an ever-increasing number of tests—something has to give. Prioritizing servers for E2E tests increases concurrency and expedites testing times, but leaves fewer servers for other build jobs which can clog up the pipeline. Alternatively, prioritizing the other build jobs leaves fewer servers for E2E tests, which reduces concurrency and extends testing times—which also clogs up the pipeline.



More test concurrency, more build queueing.

Less test concurrency, more test queueing.

More concurrency, less queueing, more money.

The only way to increase the number of builds, the frequency of deployments, and the volume of tests is to add more test runners—which, of course, increases your costs.

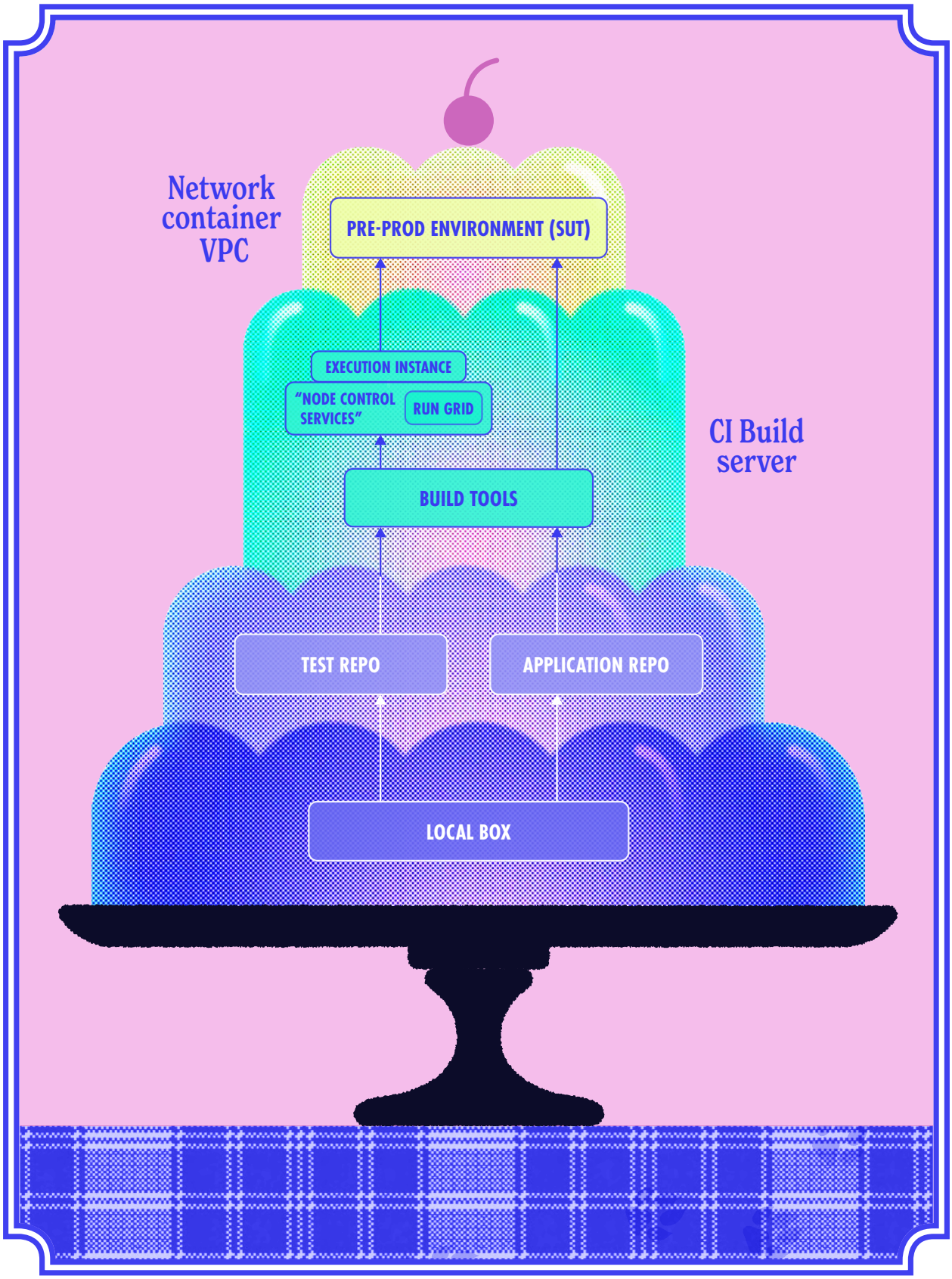
This book contains three recipes for hosted test runners on GitHub Actions. The recipes scale in relation to your concurrency targets, going from one testing server to your entire pool. We also include techniques for balancing the testing load between servers, any of which would pair well with any of the infrastructure designs.

If you've already moved beyond the capacity constraints of hosted runners, reach out and we can discuss the next evolution of your testing infrastructure.

BONS TESTS!

Hosted Test Runners

TESTING WITH GITHUB ACTIONS



TESTING WITH GITHUB ACTIONS

SERVES: 50 tests, once a day (~9 min per test) • **CONCURRENCY:** None • **PREP TIME:** 30–60 min

This is a simple and sweet pipeline for running Playwright-based end-to-end tests on a single GitHub-hosted server. The standard ingredients and easy setup don't require any special hardware, which makes it an ideal option for smaller teams (2–3 developers) looking for a straightforward solution to get started with regression testing.

For this recipe, you'll need GitHub Actions CI/CD platform, the Node.js runtime environment, and our beloved Playwright testing framework built upon gooey, jammy Docker images with Playwright's browsers already baked in. For a little extra spice you can sprinkle in Slack or Teams integration to post notifications when the tests finish. The result is a streamlined test pipeline that runs every time you deploy to master.

This set up will easily handle once-nightly runs of 50 tests, but you might find it underpowered for larger suites and more frequent deployments.

INGREDIENTS

- CI/CD platform, we're using GitHub Actions*
- GitHub Actions YML
- Base image for the run server (Preferably with Playwright and browsers pre-installed: hub.docker.com:microsoft/playwright:v1.51.1-jammy)
- Runtime environment (Node.js is delightful)
- E2E testing framework (use Playwright for fluffiest crumb)
- Playwright config file
- Suite of end-to-end tests
- *Optional:* Messaging platform (Slack, Teams, etc.)

** The DevOps engineer may substitute GitHub Actions with CircleCI or similar products that provide hosted servers.*

INSTRUCTIONS

GO TO CODE INDEX →

GitHub Actions handles the entire CI process through a single YML workflow file that outlines the steps required to install dependencies, launch test execution, and collect results. Here's how you configure the workflow file to execute a pre-built Playwright test suite on a pre-deployed build of your application. At the end of a run, the Action will generate a report of pass/fail test results that's downloadable from the run.

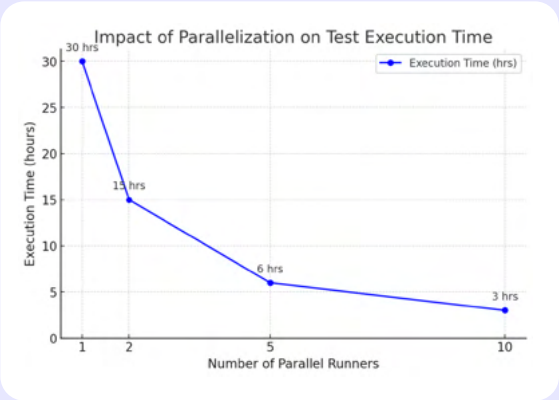
1. Modify your Playwright config file.
    a. Turn off parallel execution. GHA runners are dual-core and underpowered for massive concurrency.
    b. Set the reporter to use HTML.
    c. Set the tests to run in Chromium, Firefox, and WebKit.
2. Pull the test code from the appropriate branch.
3. Define the base image the tests run on.
4. Define test job steps:
    a. Install the runtime environment (Node.js) and its dependencies.
    b. Execute the tests.
    c. Upload result artifacts to GitHub's managed storage.
    d. *Optional:* Trigger Slack notifications.

NOTES FROM THE CHEF!

GET TO KNOW TEST PARALLELIZATION

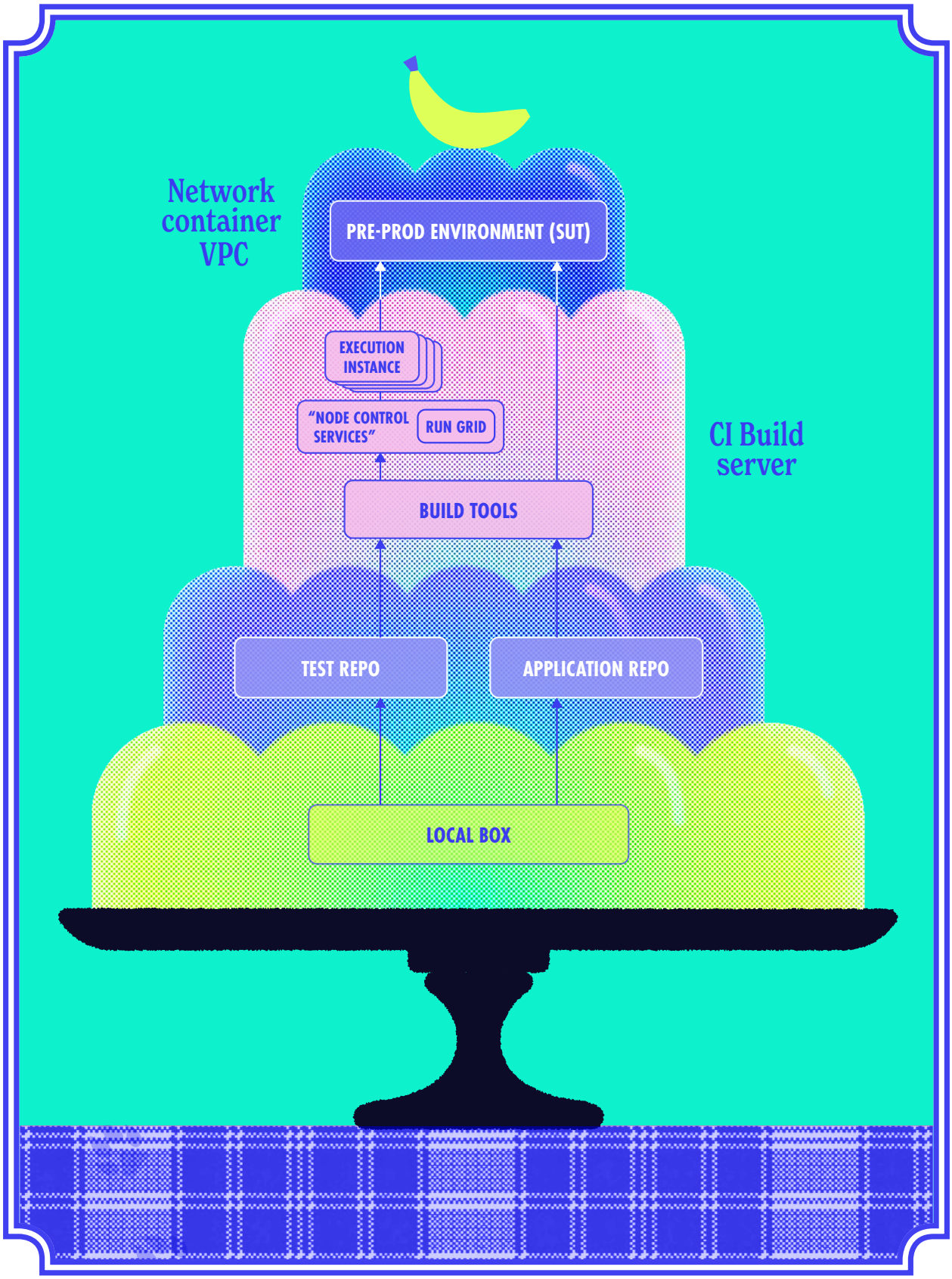
Have you ever realized—halfway through prepping a meal—that you invited way more guests than your kitchen can handle? Suddenly, you're scrambling to cook sides, main dishes, and dessert on a single stovetop. Adding concurrency to your tests is like bringing in more burners: instead of one overloaded runner tackling everything, multiple runners work in harmony, cutting down cooking (or testing) times and keeping the feedback loop nice and hot.

If you notice your CI pipeline is taking so long that your team members are standing around waiting—maybe adjusting schedules or staying late—you'll likely benefit from adding more "burners" in the form of multiple test runners. By dividing tests among parallel jobs, you can whip through heavier workloads more quickly. The result is a shorter feedback loop, easier merges, and, ultimately, a more efficient path to production.



Not every project needs this extra hardware. If your test suite is small or changes infrequently, a single runner might be enough to keep pace. But once you start noticing that each new pull request takes an eternity to verify, it's time to consider spinning up more parallel jobs. Concurrency allows your pipeline to handle a big crowd without making you stand over the stove for hours, waiting on a single pot to boil.

TEST SHARDING WITH GITHUB ACTIONS



TEST SHARDING WITH GITHUB ACTIONS

SERVES: 200 tests, once a day • **CONCURRENCY:** Up to 50 per shard • **PREP TIME:** 60–90 min

Teams that deal with large test suites—upwards of 200 tests—but need results fast love this take on GitHub Actions' classic test execution infrastructure. By splitting the test suite among multiple servers, you can slash the total suite run time.

In our recipe, we use Playwright sharding and hard-code three runners, but you can change the number of runners to fit your needs. While easy enough to set up, this recipe does have drawbacks: The shards will have uneven runtimes, and the runners aren't reserved for testing if another product team is using them.

INGREDIENTS

- CI/CD platform, we're using GitHub Actions*
- GitHub Actions YML
- Base image for the run server (Preferably with Playwright and browsers pre-installed: hub.docker.com:microsoft/playwright:v1.51.1-jammy)
- Runtime environment (Node.js gives a golden brown crust)
- E2E testing framework (use Playwright for balanced sweetness)
- Playwright config file
- Suite of end-to-end tests
- *Optional:* Messaging platform (Slack, Teams, etc.)

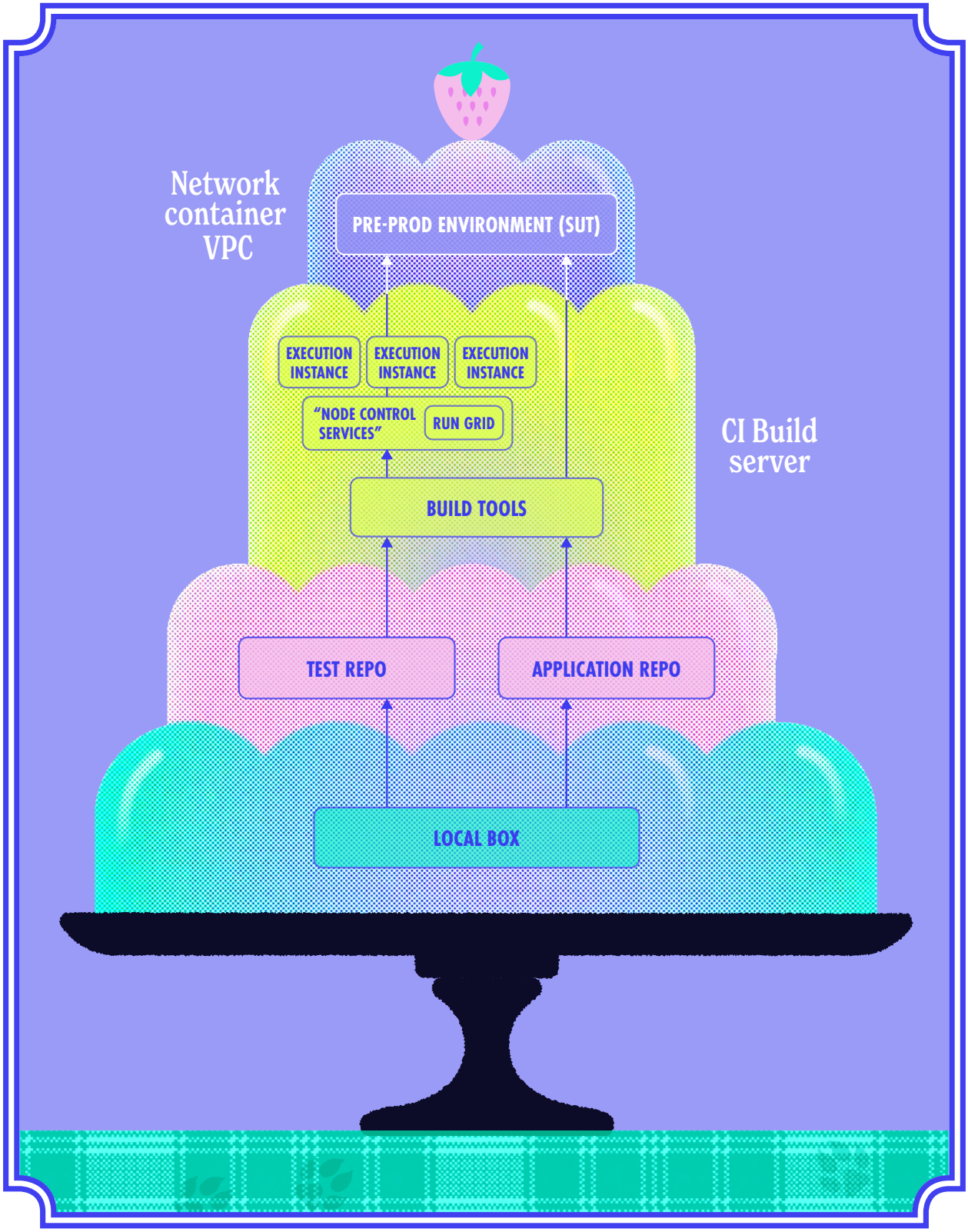
** The DevOps engineer may substitute GitHub Actions with CircleCI or similar products that provide hosted servers.*

INSTRUCTIONS

GO TO CODE INDEX →

1. Modify your Playwright config file.
   a. Turn off parallel execution.
   b. Enable blob reporting format for merging multiple reports.
   c. Set the tests to run in Chromium, Firefox, and WebKit.
2. Pull the test code from the appropriate branch.
3. Define the base image the tests run on.
4. Define controller job steps:
   a. **Install dependencies.** Install Node.js and all required dependencies.
   b. **Create and assign shards to be used in the test job.** Dynamically split test files into shards. (Using 3 shards works well!)
5. Define test job steps:
   a. Associate the controller with the test job. Retrieve the generated test matrix (step 4b).
   b. Install the runtime environment.
   c. Install the test framework.
   d. Execute the tests.
   e. Upload result artifacts to GHA's managed storage.
   f. *Optional:* Trigger Slack notifications.

TESTING WITH ADAPTIVE RUNNER DISTRIBUTION ON GITHUB ACTIONS



TESTING WITH ADAPTIVE RUNNER DISTRIBUTION ON GITHUB ACTIONS

SERVES: 200 tests, once a day • **CONCURRENCY:** Variable by runner availability • **PREP TIME:** 1.5–2hrs

The brilliance of this recipe is that it makes use of every available runner in your pool. The difficulty of this recipe is that it makes use of every available runner in your pool. Rather than hard-coding the number of runners to use as we did in Recipe 2, we query the pool when the suite gets kicked off to see how many runners we can divide our tests into.

The effect is the highest possible concurrency, the fastest test suite, and a warm gooey center, but the SDET or DevOps attempting this recipe will find that it blocks other jobs when the tests are started. We recommend that you prepare a job to kill some or all jobs in progress to free up runner capacity in an emergency. Note the use of GHA composite actions. While not required, it will make your code easier to read and maintain.

INGREDIENTS

- CI/CD platform, we're using GitHub Actions*
- GitHub Actions YML
- Base image for the run server (Preferably with Playwright and browsers pre-installed: hub.docker.com:microsoft/playwright:v1.51.1-jammy)
- Runtime environment (try Node.js for rustic charm)
- E2E testing framework (use Playwright for even bake)
- Playwright config file
- Suite of end-to-end tests
- *Optional:* Messaging platform (Slack, Teams, etc.)

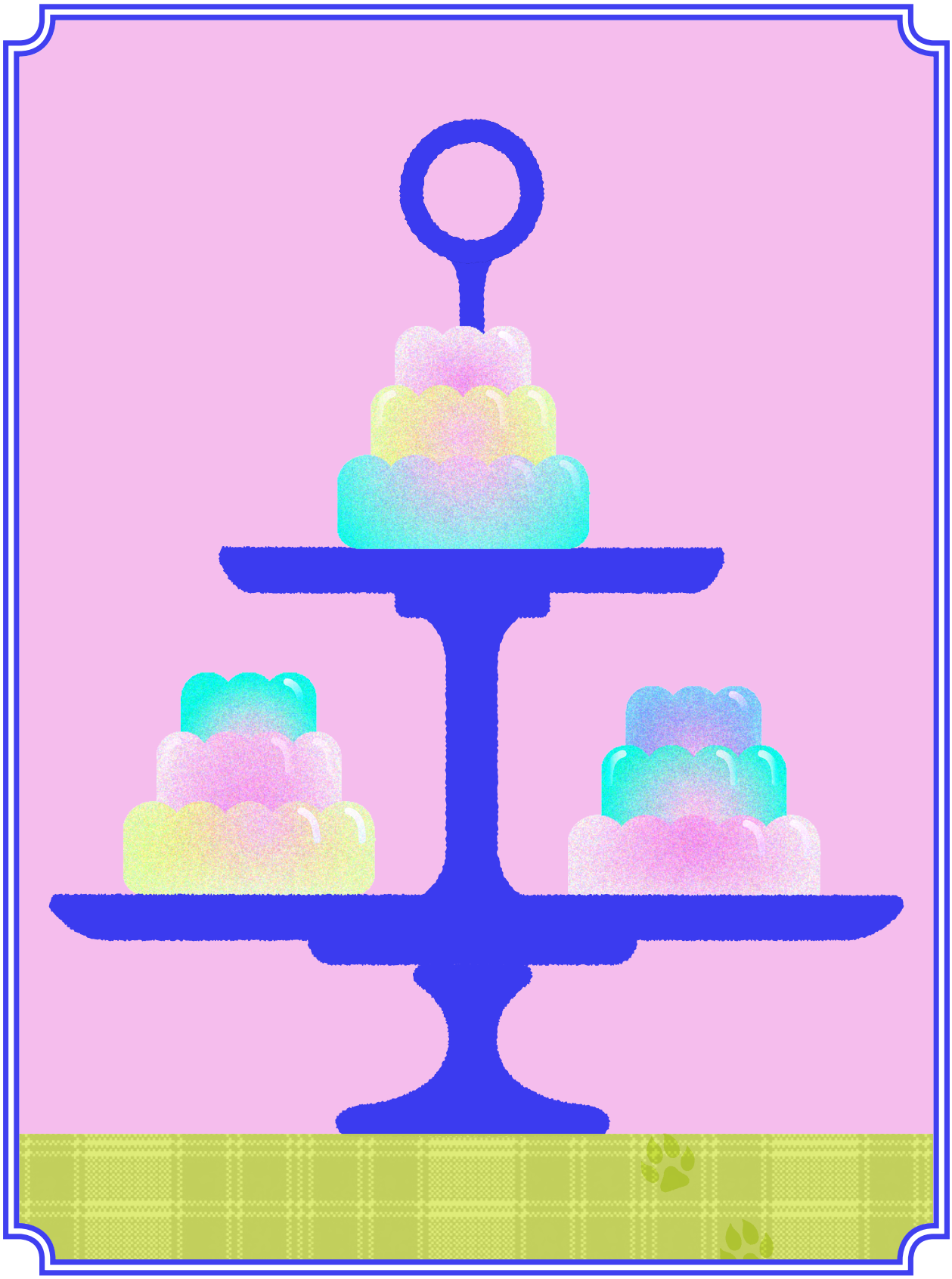
** The DevOps engineer may substitute GitHub Actions with CircleCI or similar products that provide hosted servers.*

INSTRUCTIONS

GO TO CODE INDEX →

1. Modify your Playwright config file.
   a. Turn off parallel execution.
   b. Enable blob reporting format for merging multiple reports.
   c. Set the tests to run in Chromium, Firefox, and WebKit.
2. Start a composite action file.
3. Create five jobs within the composite action file:
   a. 'Check capacity job' outputs the number of available runners.
   b. 'Generate shards job' outputs shard names for each available runner.
   c. 'Test job' executes tests using the list of shards from step 3b as input.
   d. 'Merge reports job' aggregates test results and delivers them to the GHA artifact repo once the test job is finished.
   e. 'Notify Slack job' shares test suites pass/fall status to a specified Slack channel.
4. Start a 'Check capacity job' file.
   a. Define the output for the job as the number of available runners in the runner pool.
   b. Query the GitHub client (`gh`) to get the total number of available runners at run time.
   c. Set a minimum runner count. This function creates at least five shards, no matter how many runners are available.
5. Start a 'generate shards job' file.
   a. Assign each test in the suite to a shard.
6. Start a 'run tests job' file.
   a. Run the tests on the shards defined above.
   b. Generate a pass/fail report for each shard and store it in a shared location to be aggregated with other reports in the next step.
7. Start a 'merge the reports job' file.
   a. Merge the reports generated above into a single report file.
   b. Store the merged report in GHA storage management.
8. Start a 'send Slack notification job' file.
   a. Designate a Slack channel where you want to send the messages.
   b. Send the appropriate message.

TEST DISTRIBUTION ACTIONS 3 WAYS



TEST DISTRIBUTION ACTIONS 3 WAYS

The perfect accompaniment to GitHub Actions test infrastructure is a method for distributing tests among the available runners (shards). Like varieties of scones, there are limitless ways to distribute your tests, but these three stand out for their adaptability to different team needs. You can even combine methods for a truly unique solution!

INGREDIENT

- A GitHub Actions job to distribute tests among available shards (See Page 13)

STRATEGY 1: ROUND ROBIN DISTRIBUTION

GO TO CODE INDEX →

This quick and easy method deals out one test at a time to each runner in the pool, starting from the top until all tests have been assigned. Expect irregular and unpredictable QA cycle times using this method, as the total run-minutes of each shard will vary with the number of runners and the length of the assigned tests.

	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6
Shard A	Test 1 – 3 min	Test 1 – 3 min Test 4 – 9 min	Test 1 – 3 min Test 3 – 2 min Test 5 – 4 min	Test 1 – 3 min Test 5 – 4 min	Test 1 – 3 min Test 2 – 5 min Test 3 – 2 min Test 4 – 9 min Test 5 – 4 min Test 6 – 3 min	Test 1 – 3 min Test 6 – 3 min
Shard B	Test 2 – 5 min	Test 2 – 5 min Test 5 – 4 min	Test 2 – 5 min Test 4 – 9 min Test 6 – 3 min	Test 2 – 5 min Test 6 – 3 min		Test 2 – 5 min
Shard C	Test 3 – 2 min	Test 3 – 2 min Test 6 – 3 min		Test 3 – 2 min		Test 3 – 2 min
Shard D	Test 4 – 9 min			Test 4 – 9 min		Test 4 – 9 min
Shard E	Test 5 – 4 min					Test 5 – 4 min
Shard F	Test 6 – 3 min					
Time elapsed	**9 minutes**	**12 minutes**	**17 minutes**	**9 minutes**	**26 minutes**	**9 minutes**

INSTRUCTIONS

1. Modify your Playwright config file.
   a. Turn off parallel execution.
   b. Enable blob reporting format for merging multiple reports.
   c. Set the tests to run in Chromium, Firefox, and WebKit.

2. Start a composite action file.
3. Create 3 jobs within the composite action file.
    a. 'Generate matrix job' outputs the names of the test for each available runner.
    b. 'Test job' executes the tests using the list of shards from step 3b as input. It also cleans special characters out of the test names so they can be used for generating reports.
    c. 'Merge reports job' aggregates test results and delivers them to the GHA artifact repo once the test job is finished.
4. Start a 'Round Robin" file.
    a. Assign each test in the suite to a shard.
5. Start a 'merge the reports job' file
    a. Merge the reports generated above into a single report file.
    b. Store the merged report in GHA storage management.

STRATEGY 2: DISTRIBUTE BY TEST METADATA

GO TO CODE INDEX →

Distributing tests using their metadata can be helpful in congested pipelines. The QA team could, for example, prioritize P0 and P1 workflows when the pipeline is bustling and run lower-priority tests at off-peak times.

This particular recipe uses the Playwright @-tag but could be adapted to use labels, test titles, or other metadata — chef's choice here. Because this approach depends on manual tagging (that's a feature), your testers have to make sure all the tests are appropriately tagged.

INSTRUCTIONS

1. Modify your Playwright config file.
    a. Turn off parallel execution.
    b. Enable blob reporting format for merging multiple reports.
    c. Set the tests to run in Chromium, Firefox, and WebKit.
2. Start a composite action file.
3. Create two jobs within the composite action file.
    a. 'Test job' executes the tests using the list of shards from step 3b as input. It also cleans special characters out of the test names so they can be used for generating reports.
    b. 'Merge reports job' aggregates test results and delivers them to the GHA artifact repo once the test job is finished.
4. Start a 'merge the reports job' file.
    a. Merge the reports generated above into a single report file.
    b. Store the merged report in GHA storage management.

STRATEGY 3: DISTRIBUTE BY FILE PATHS

GO TO CODE INDEX →

A straightforward approach to manually controlling test distribution is to be careful with what you put where. In our recipe, we use the paths `tests/**` and `tests2/**` to divide our tests into two shards so they can run concurrently because we know they won't interfere with each other. The advantage of this method is that you simply define where a test runs by putting it in the correct path.

INSTRUCTIONS

1. Modify your Playwright config file.
    a. Turn off parallel execution.
    b. Enable blob reporting format for merging multiple reports.
    c. Set the tests to run in Chromium, Firefox, and WebKit.
2. Start a composite action file.
3. Create two jobs within the composite action file.
    a. 'Test job' executes the tests using the list of shards from step 3b as input. It also cleans special characters out of the test names so they can be used for generating reports
    b. 'Merge reports job' aggregates test results and delivers them to the GHA artifact repo once the test job is finished.
4. Start a 'merge the reports job' file.
    a. Merge the reports generated above into a single report file.
    b. Store the merged report in GHA storage management.

Dessert Course

TESTING WITH SELF-HOSTED RUNNERS

Beyond about 50 automated E2E tests (or more than one suite run per day) it starts to make strategic and economic sense to bring your test execution infrastructure in-house. The cost of hosted CI services like GitHub Actions rises linearly with your concurrency needs, but by building your own system, you can scale more efficiently over time, so you can run more tests in parallel and clear bottlenecks that slow down deployments.

We can't provide a one-size-fits-all recipe here, because every organization needs something different. However, there is a group of core components that you need to plan for when embarking on the self-hosted test running adventure.

From a hardware perspective, the key pieces you'll need to bake the MVP version of your first home-cooked test execution environment are as follows:

- Container-based test image (using something like Docker or Podman).
- Managed container orchestration platform (i.e., Kubernetes).
- Container registry.
- Persistent bucket storage for artifacts (i.e., S3).
- Chart-based dependency manager (i.e., Helm).
- CI/CD orchestrator like Argo Workflows.

These days, it makes sense to deploy all of this into a managed cloud environment, such as EKS or GKE, with your organization's networking, security, and permissions frameworks integrated—most likely managed through Infrastructure as Code (IaC) tools like Terraform. With these ingredients, you can distribute tests into individual containers, run them in parallel, and merge results into a centralized report.

Once your infrastructure is in place, operating and maintaining it becomes a whole other challenge. You'll need:

- Hardened security for your clusters.
- Automatic updates and patching.
- Container orchestration troubleshooting.
- Video capture and log storage for debugging.
- Monitoring and alerting to catch problems before they escalate.
- Concurrency limits and sequencing rules to avoid resource contention.
- Automatic retry logic for flaky tests.
- Data cleanup systems.
- Caching strategies for authentication tokens.
- Device management if you're testing hardware.
- PR validation pipelines as your CI/CD practice increases, to test earlier in the cycle, along with ephemeral testing environments.

And all that hardware and software doesn't operate itself. In the short term, you'll need engineers to scale up safely. In the long term, you'll need to make a sustained investment in people to maintain, monitor, and evolve the system as your testing needs grow. Building your own infrastructure is the best way to scale, but it comes with responsibilities that grow just as steadily as your test suite.

Or, you can skip all that.

QA Wolf has already built this entire system—the infrastructure, the scaling, the operational tooling—and when you sign up with us, it's all included (at no extra cost).

Streamlined.
Scalable.
Fully Automatic.

QA WOLF is the MODERN MARVEL of QA INFRASTRUCTURE

Now, for the first time, a complete solution to your software testing headaches. QA Wolf delivers a push-button QA system that's built to scale, fully automatic, and streamlined for speed.

Forget cobbling together fragile test rigs. While the other guys ration test runners and wrangle endless pipelines, QA Wolf runs your entire suite in parallel; no maintenance, no guesswork, no delays.

- Precision-engineered to grow with your team.
- Plug-in ready—works right out of the box.
- Electronic-brain efficiency, without the upkeep.
- It practically thinks for you!

Just set it and forget it—QA Wolf does the work of ten testers, with failsafe performance that's trusted by today's most forward-thinking teams.

You'll wonder how you ever shipped without it.

Code Index

TESTING WITH GITHUB ACTIONS

GITHUB BACK TO RECIPE →

1. Modify your Playwright config file.
   a. Turn off parallel execution. GHA runners are dual-core and underpowered for massive concurrency.
   b. Set the reporter to use HTML.
   c. Set the tests to run in Chromium, Firefox, and WebKit.

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

export default defineConfig({
  testDir: './tests',
  fullyParallel: false,
  forbidOnly: !!process.env.CI,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [['html']],
  use: {
    trace: 'on-first-retry',
  },

  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    },

    {
      name: 'firefox',
      use: { ...devices['Desktop Firefox'] },
    },

    {
      name: 'webkit',
      use: { ...devices['Desktop Safari'] },
    },
  ],
});
```

2. Pull the test code from the appropriate branch.
3. Define the base image the tests run on
4. Define test job steps:
   a. Install the runtime environment (Node.js) and its dependencies.
   b. Execute the tests.
   c. Upload result artifacts to GitHub's managed storage.
   d. *Optional:* Trigger Slack notifications.

```
name: Playwright Tests
name: Single Shard Execution
on:
  push:
    branches:
      - master
  workflow_dispatch:
jobs:
  test:
    name: Run Playwright tests
    runs-on: ubuntu-latest #(microsoft/playwright)
    container:
      image: mcr.microsoft.com/playwright:v1.51.1-jammy
    #this needs to be here for Firefox
    env:
      HOME: /root
    steps:
      - name: Notify Slack (Job Success)
        uses: slackapi/slack-github-action@v1.23.0
        with:
          # eng-monitoring-ops
          channel-id: "C03CX7K9JJG"
          slack-message: "Playwright tests have started for
${{ github.repository }} - ${{ github.ref }}"
        env:
          SLACK_BOT_TOKEN: ${{ secrets.SLACK_SEND_TOKEN }}

      - name: Checkout repository
```

```
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 18

      - name: Install dependencies
        run: npm ci

      - name: Run Playwright tests
        run: npx playwright test --config=playwright.config.
recipe1.ts

      - name: Upload test results
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: playwright-report
          path: playwright-report
          retention-days: 1

      - name: Notify Slack (Job Success)
        if: success()
        uses: slackapi/slack-github-action@v1.23.0
        with:
          # eng-monitoring-ops
          channel-id: "C03CX7K9JJG"
          slack-message: "✅ Playwright tests passed for ${{
github.repository }} - ${{ github.ref }}"
        env:
          SLACK_BOT_TOKEN: ${{ secrets.SLACK_SEND_TOKEN }}


      - name: Notify Slack (Job Success)
        if: failure()
        uses: slackapi/slack-github-action@v1.23.0
        with:
          # eng-monitoring-ops
          channel-id: "C03CX7K9JJG"
          slack-message: "❌ Playwright tests failed for ${{
github.repository }} - ${{ github.ref }}. Check logs for
details."
        env:
          SLACK_BOT_TOKEN: ${{ secrets.SLACK_SEND_TOKEN }}
```

TEST SHARDING IN GITHUB ACTIONS

GITHUB BACK TO RECIPE →

1. Modify your Playwright config file.
   a. Turn off parallel execution.
   b. Enable blob reporting format for merging multiple reports.
   c. Set the tests to run in Chromium, Firefox, and WebKit.

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

export default defineConfig({
  testDir: './tests',
  fullyParallel: false,
  forbidOnly: !!process.env.CI,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [['html']],
  use: {
    trace: 'on-first-retry',
  },

  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    },

    {
      name: 'firefox',
      use: { ...devices['Desktop Firefox'] },
    },

    {
      name: 'webkit',
      use: { ...devices['Desktop Safari'] },
    },
  ],
});
```

2. Pull the test code from the appropriate branch.
3. Define the base image the tests run on.
4. Define controller job steps:
   a. **Install dependencies.** Install Node.js and all required dependencies.
   b. **Create and assign shards to be used in the test job.** Dynamically split test files into shards. (Using 3 shards works well!)
5. Define test job steps:
   a. Associate the controller with the test job. Retrieve the generated test matrix (step 4b).

```
 name: Test Sharding
on:
  push:
    branches:
      - master
jobs:
  playwright:
    name: Generate Test Shards
    runs-on: ubuntu-latest
    container:
      image: mcr.microsoft.com/playwright:v1.51.1-jammy
    #this needs to be here for Firefox

    env:
      HOME: /root
    strategy:
      fail-fast: false
      matrix:
        shardIndex: [1, 2, 3]
        shardTotal: [3]
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4
      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 18
      - name: Install dependencies
```

b. Install the runtime environment.
c. Install the test framework.
d. Execute the tests.
e. Upload result artifacts to GHA's managed storage.
f. *Optional:* Trigger Slack notifications.

```
        run: npm ci
      - name: Install Playwright
        run: npx playwright install
      - name: Run assigned tests
        run: npx playwright test --shard=${{ matrix.shardIn-
dex }}/${{ matrix.shardTotal }} --config=playwright.config.
recipe2.ts

      - name: Upload test results
        if: always()
        uses: actions/upload-artifact@v4
        with:
          name: blob-report-${{ matrix.shardIndex }}
          path: blob-report
          retention-days: 1
  merge-reports:
    # Merge reports after playwright-tests, even if some
shards have failed
    if: ${{ !cancelled() }}
    needs: [playwright]

    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4
    - uses: actions/setup-node@v4
      with:
        node-version: 18
    - name: Install dependencies
      run: npm ci

    - name: Download blob reports from GitHub Actions Arti-
facts
      uses: actions/download-artifact@v4
      with:
        path: all-blob-reports
        pattern: blob-report-*
        merge-multiple: true

    - name: Merge into HTML Report
      run: npx playwright merge-reports --reporter html ./
all-blob-reports

    - name: Upload HTML report
      uses: actions/upload-artifact@v4
      with:
        name: html-report--attempt-${{ github.run_attempt }}
        path: playwright-report
        retention-days: 1

    - name: Notify Slack (Job Success)
      if: success()
      uses: slackapi/slack-github-action@v1.23.0
      with:
        # eng-monitoring-ops
        channel-id: "C03CX7K9JJG"
        slack-message: "✅ Playwright tests passed for ${{
github.repository }} - ${{ github.ref }}"
      env:
        SLACK_BOT_TOKEN: ${{ secrets.SLACK_SEND_TOKEN }}


    - name: Notify Slack (Job Success)
      if: failure()
      uses: slackapi/slack-github-action@v1.23.0
      with:
        # eng-monitoring-ops
        channel-id: "C03CX7K9JJG"
        slack-message: "❌ Playwright tests failed for ${{
github.repository }} - ${{ github.ref }}. Check logs for
details."
      env:
        SLACK_BOT_TOKEN: ${{ secrets.SLACK_SEND_TOKEN }}
```

TESTING WITH ADAPTIVE RUNNER DISTRIBUTION ON GITHUB ACTIONS

GITHUB BACK TO RECIPE →

1. Modify your Playwright config file.
   a. Turn off parallel execution.
   b. Enable blob reporting format for merging multiple reports.
   c. Set the tests to run in Chromium, Firefox, and WebKit.

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

export default defineConfig({
  testDir: './tests',
  fullyParallel: false,
  forbidOnly: !!process.env.CI,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [['blob']],
  use: {
    trace: 'on-first-retry',
  },

  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    },

    {
      name: 'firefox',
      use: { ...devices['Desktop Firefox'] },
    },

    {
      name: 'webkit',
      use: { ...devices['Desktop Safari'] },
    },
});
```

2. Start a composite action file.
3. Create five jobs within the composite action file:
   a. 'Check capacity job' outputs the number of available runners.
   b. 'Generate shards job' outputs shard names for each available runner.
   c. 'Test job' executes tests using the list of shards from step 3b as input.
   d. 'Merge reports job' aggregates test results and delivers them to the GHA artifact repo once the test job is finished.
   e. 'Notify Slack job' shares test suites pass/fall status to a specified Slack channel.

```
name: Adaptive Playwright Tests (composite)
on:
  push:
    branches:
      - master

concurrency:
  group: "tests-${{ github.ref }}"
  cancel-in-progress: false

jobs:
  check-capacity:
    runs-on: ubuntu-latest
    outputs:
      capacity: ${{ steps.check-capacity.outputs.available_
runners }}
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Check capacity
        id: check-capacity
        uses: ./.github/actions/check-capacity
        with:
          github-token: ${{ secrets.GITHUB_TOKEN }}

  generate-shard:
    runs-on: ubuntu-latest
    needs: check-capacity
    outputs:
```

```
      shard_list: ${{ steps.set-shards.outputs.shard_list }}
    steps:
      - uses: actions/checkout@v4

      - name: Generate shard list
        id: set-shards
        uses: ./.github/actions/generate-shards
        with:
          total-runners: ${{ needs.check-capacity.outputs.
capacity }}

      - name: Read shard list output
        id: set-output
        run: echo "shard_list=$(cat shard-list.txt)" >>
$GITHUB_OUTPUT

      - uses: actions/upload-artifact@v4
        with:
          name: test-splits
          path: test-splits/

  test:
    name: Run ${{ matrix.shard }} on ${{ matrix.browser }}
    needs: generate-shard
    runs-on: ubuntu-latest
    container:
      image: mcr.microsoft.com/playwright:v1.51.1-jammy
    env:
      HOME: /root
    strategy:
      matrix:
        shard: ${{ fromJson(needs.generate-shard.outputs.
shard_list) }}
        browser: [chromium, firefox, webkit]
    steps:
      - uses: actions/checkout@v4

      - name: Run test shard on browser
        uses: ./.github/actions/run-shard
        with:
          shard: ${{ matrix.shard }}
          browser: ${{ matrix.browser }}

  merge-reports:
    name: Merge Playwright Reports
    needs: test
    if: always()
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Merge reports
        uses: ./.github/actions/merge-reports

  notify-slack:
    runs-on: ubuntu-latest
    needs: test
    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Notify Slack
        uses: ./.github/actions/slack-notify
        with:
          channel-id: "C03CX7K9JJG"
          success-message: "✅ Playwright tests passed for
${{ github.repository }} - ${{ github.ref }}"
          failure-message: "❌ Playwright tests failed for
${{ github.repository }} - ${{ github.ref }}. Check logs for
details."
          slack-token: ${{ secrets.SLACK_SEND_TOKEN }}
```

4. Start a 'Check capacity job' file.
    a. Define the output for the job as the number of available runners in the runner pool.
    b. Query the GitHub client (`gh`) to get the total number of available runners at run time.
    c. Set a minimum runner count. This function creates at least 5 shards, no matter how many runners are available.

```
name: "Check GitHub Runner Capacity"
description: "Check how many runners are available"
inputs:
  github-token:
    description: "GitHub token to call the API"
    required: true
outputs:
  available_runners:
    description: "Number of available runners"
    value: ${{ steps.check-capacity.outputs.available_run-
ners }}
runs:
  using: "composite"
  steps:
    - id: check-capacity
      run: |
        MAX_RUNNERS=40
        CURRENT_RUNNING=$(gh run list --limit $MAX_RUNNERS
--json status --repo $GITHUB_REPOSITORY | jq '[.[] | se-
lect(.status == "in_progress")] | length')
        AVAILABLE=$((MAX_RUNNERS - CURRENT_RUNNING))
        FINAL_COUNT=$((AVAILABLE > 5 ? AVAILABLE : 5))
        echo "available_runners=$FINAL_COUNT" >> $GITHUB_
OUTPUT
      shell: bash
      env:
        GITHUB_TOKEN: ${{ inputs.github-token }}
```

5. Start a 'generate shards job' file.
    a. Assign each test in the suite to a shard.

```
name: "Generate Playwright Shards"
description: "Split tests and create a JSON array of shard
names"
inputs:
  total-runners:
    description: "Number of runners to split against"
    required: true
outputs:
  shard_list:
    description: "JSON array of test shard filenames"
    value: ${{ steps.emit-shards.outputs.shard_list }}
runs:
  using: "composite"
  steps:
    - run: |
        mkdir -p test-splits
        find tests -name "*.spec.ts" | sort > all-tests.txt
        TOTAL_TESTS=$(wc -l < all-tests.txt)

        TOTAL_RUNNERS="${{ inputs.total-runners }}"
        if [ -z "$TOTAL_RUNNERS" ] || [ "$TOTAL_RUNNERS" -eq
0 ]; then
          echo "❌ TOTAL_RUNNERS is empty or zero"
          exit 1
        fi
        SPLIT_COUNT=$(( TOTAL_TESTS / TOTAL_RUNNERS + 1 ))

        split -l $SPLIT_COUNT all-tests.txt test-splits/
shard_

        shards=$(ls test-splits | sort)
        shard_array=""
        for shard in $shards; do
          shard_array+="\"$shard\","
        done

        shard_list="[${shard_array%,}]"
        echo "$shard_list" > shard-list.txt
      shell: bash
```

```
    - id: emit-shards
      run: echo “shard_list=$(cat shard-list.txt)” >>
$GITHUB_OUTPUT
      shell: bash
```

6. Start a ‘run tests job’ file.
    a. Run the tests on the shards defined above.
    b. Generate a pass/fail report for each shard and store it in a shared location to be aggregated with other reports in the next step.

```
name: “Run Shard”
description: “Run tests for one shard/browser combo and up-
load blob report”
inputs:
  shard:
    required: true
  browser:
    required: true
runs:
  using: “composite”
  steps:
    - uses: actions/checkout@v4

    - uses: actions/setup-node@v4
      with:
        node-version: 18

    - run: npm ci
      shell: bash

    - run: npx playwright install --with-deps
      shell: bash

    - uses: actions/download-artifact@v4
      with:
        name: test-splits
        path: test-splits

    - run: |
        TESTS=$(cat test-splits/${{ inputs.shard }})
        echo “TESTS=$TESTS” >> $GITHUB_ENV
      shell: bash

    - run: |
        echo “ Running ${{ inputs.shard }} on ${{ inputs.
browser }}”
        BROWSER=${{ inputs.browser }}
        SHARD=${{ inputs.shard }}
        npx playwright test $TESTS --project=$BROWSER --con-
fig=playwright.config.recipe3.ts
      shell: bash

    - uses: actions/upload-artifact@v4
      if: always()
      with:
        name: blob-report-${{ inputs.browser }}-${{ inputs.
shard }}
        path: blob-report/
        if-no-files-found: ignore
```

7. Start a 'merge the reports job' file
    a. Merge the reports generated above into a single report file.
    b. Store the merged report in GHA storage management.

```
name: "Merge Blob Reports"
description: "Download all blob reports and generate merged
HTML report"
runs:
  using: "composite"
  steps:
    - uses: actions/checkout@v4

    - uses: actions/setup-node@v4
      with:
        node-version: 18

    - run: npm ci
      shell: bash

    - uses: actions/download-artifact@v4
      with:
        pattern: blob-report-*
        path: blob-report
        merge-multiple: true

    - run: |
        echo "📦 Merging reports..."
        npx playwright merge-reports --reporter html
blob-report
      shell: bash

    - uses: actions/upload-artifact@v4
      with:
        name: merged-playwright-report
        path: playwright-report/
```

8. Start a 'merge the reports job' file
    a. Designate a Slack channel where you want to send the messages.
    b. Send the appropriate message.

```
name: 'Slack Notification on Job Success/Failure'
description: 'Notify Slack on job success or failure'
inputs:
  channel-id:
    description: 'Slack channel ID to send notifications to'
    required: true
    default: 'C03CX7K9JJG'
  success-message:
    description: 'Message to send when the job is success-
ful'
    required: true
    default: '✅ Playwright tests passed for ${{ github.
repository }} - ${{ github.ref }}'
  failure-message:
    description: 'Message to send when the job fails'
    required: true
    default: '❌ Playwright tests failed for ${{ github.re-
pository }} - ${{ github.ref }}. Check logs for details.'
  slack-token:
    description: 'Slack Bot Token'
    required: true

runs:
  using: 'composite'
  steps:
    - name: Notify Slack (Job Success)
      if: success()
      run: ./slack_notify.sh ${{ inputs.channel-id }} ${{
inputs.success-message }} ${{ inputs.slack-token }}

    - name: Notify Slack (Job Failure)
      if: failure()
      run: ./slack_notify.sh ${{ inputs.channel-id }} ${{
inputs.failure-message }} ${{ inputs.slack-token }}
```

TEST DISTRIBUTION ACTIONS THREE WAYS

Strategy 1: Round Robin Distribution

GITHUB BACK TO RECIPE →

1. Modify your Playwright config file.
   a. Turn off parallel execution.
   b. Enable blob reporting format for merging multiple reports.
   c. Set the tests to run in Chromium, Firefox, and WebKit.

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

export default defineConfig({
  testDir: './tests', // Single root directory for Play-
wright to look in
  fullyParallel: false,
  forbidOnly: !!process.env.CI,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [
      ['blob'],
    ],  /* Shared settings for all the projects below. See
https://playwright.dev/docs/api/class-testoptions. */
  use: {
    trace: 'on-first-retry',
    headless: true,
  },

  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    }
  ],
});
```

2. Start a composite action file.
3. Create 3 jobs within the composite action file:
   a. 'Generate matrix job' outputs the names of the test for each available runner.
   b. 'Test job' executes the tests using the list of shards from step 3b as input. It also cleans special characters out of the test names so they can be used for generating reports.
   c. 'Merge reports job' aggregates test results and delivers them to the GHA artifact repo once the test job is finished.

```
name: Dynamic Distribution - Round Robin
on:
  push:
    branches:
      - master

jobs:
  generate-matrix:
    name: Generate Test Distribution
    runs-on: ubuntu-latest
    outputs:
      matrix: ${{ steps.set-matrix.outputs.matrix }}
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4
      - name: Choose Test Distribution Strategy
        id: set-matrix
        uses: ./.github/actions/distribution/round-robin  #
Change this line to switch strategies


  test-runner:
    name: Run Playwright Tests
    needs: generate-matrix
    strategy:
      matrix:
        include: ${{ fromJson(needs.generate-matrix.outputs.
matrix) }}
    runs-on: ubuntu-latest
    container:
      image: mcr.microsoft.com/playwright:v1.51.1-jammy
```

```
    steps:

      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 18

      - name: Install dependencies
        run: npm ci

      - name:  Run Playwright
        id: run
        run: |
          npx playwright test ${{ join(matrix.testFile, ' ')
}}

      # Sanitize the test file name to replace '/' with '-'

      - name: Sanitize artifact name
        id: sanitize
        run: |
          sanitized_name="${{ matrix.testFile[0] }}"
          sanitized_name=$(echo "$sanitized_name" | sed
's/\//-/g')  # Replace / with -
          echo "sanitized_name=$sanitized_name" >> $GITHUB_
ENV  # Save to environment variable

      - name: Upload test results artifact
        uses: actions/upload-artifact@v4
        with:
          name: blob-report-${{ env.sanitized_name }}  # Use
the sanitized name
          path: blob-report/  # Path where Playwright stores
the reports (update as needed)
          if-no-files-found: ignore
          retention-days: 1

  merge-reports:
    name: Merge Playwright Reports
    needs: test-runner
    if: always()
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Merge reports
        uses: ./.github/actions/merge-reports
```

4. Start a 'Round Robin" file.
    a. Assign each test in the suite to a shard.

```
name: Matrix-Based Test Distribution
description: Splits Playwright tests into shards using
GitHub Actions matrix

inputs: {}

outputs:
  matrix:
    description: "Test distribution matrix"
    value: ${{ steps.set-matrix.outputs.matrix }}

runs:
  using: "composite"
  steps:
    - name: Generate and split test list into shards
      id: set-matrix
      shell: bash
      run: |
```

```
        NUM_SHARDS=4

        TEST_FILES=$(find tests -type f -name "*.spec.ts" |
jq -R -s -c 'split("\n")[:-1]')
        if [ -z "$TEST_FILES" ] || [ "$TEST_FILES" = "[]" ];
then
          echo "⚠️ No test files found"
          echo "matrix=[]" >> "$GITHUB_OUTPUT"
          exit 0
        fi

        SHARD_MATRIX=$(echo "$TEST_FILES" | jq -c --argjson
n "$NUM_SHARDS" '
          . as $all
          | ($all | length) as $total
          | [range(0; $n) |
              $all[
                ((. * $total) / $n | floor)
                :
                (((. + 1) * $total) / $n | floor)
              ]
            ]
        ')
        CLEANED_MATRIX=$(echo "$SHARD_MATRIX" | jq -c '[.[]
| select(length > 0)] | map({ testFile: . })')
        printf "matrix=%s" "$CLEANED_MATRIX" >> "$GITHUB_
OUTPUT"
        echo 'cat GITHUB_OUTPUT'
        cat $GITHUB_OUTPUT

    - name: Debug matrix
      shell: bash
      run: |
        echo " steps.set-matrix.outputs.matrix "
        echo "${{ steps.set-matrix.outputs.matrix }}"
```

5. Start a 'merge the reports job' file
   a. Merge the reports generated above into a single report file.
   b. Store the merged report in GHA storage management.

```
name: "Merge Blob Reports"
description: "Download all blob reports and generate merged
HTML report"
runs:
  using: "composite"
  steps:
    - uses: actions/checkout@v4

    - uses: actions/setup-node@v4
      with:
        node-version: 18

    - run: npm ci
      shell: bash

    - uses: actions/download-artifact@v4
      with:
        pattern: blob-report-*
        path: blob-report
        merge-multiple: true

    - run: |
        echo "📦 Merging reports..."
        npx playwright merge-reports --reporter html
blob-report
      shell: bash

    - uses: actions/upload-artifact@v4
      with:
        name: merged-playwright-report
        path: playwright-report/
```

TEST DISTRIBUTION ACTIONS THREE WAYS

Strategy 2: Distribute by Test Metadata

GITHUB BACK TO RECIPE →

1. Modify your Playwright config file.
   a. Turn off parallel execution.
   b. Enable blob reporting format for merging multiple reports.
   c. Set the tests to run in Chromium, Firefox, and WebKit.

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

export default defineConfig({
  testDir: './tests', // Single root directory for Play-
wright to look in
  fullyParallel: false,
  forbidOnly: !!process.env.CI,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [
      ['blob'],
    ],  /* Shared settings for all the projects below. See
https://playwright.dev/docs/api/class-testoptions. */
  use: {
    trace: 'on-first-retry',
    headless: true,
  },

  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    }
  ],
});
```

2. Start a composite action file.
3. Create two jobs within the composite action file:
   a. 'Test job' executes the tests using the list of shards from step 3b as input. It also cleans special characters out of the test names so they can be used for generating reports.
   b. 'Merge reports job' aggregates test results and delivers them to the GHA artifact repo once the test job is finished.

```
name: Dynamic Distribution - Tags
on:
  push:
    branches:
      - master


jobs:

  test-runner:
    name: Run Playwright Tests
    strategy:
      matrix:
        items: ["@smoke", "@regression", "@e2e"]  # Matrix
should be an object with a key 'tags'
    runs-on: ubuntu-latest
    container:
      image: mcr.microsoft.com/playwright:v1.51.1-jammy
    steps:

      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 18

      - name: Install dependencies
        run: npm ci
```

```
      - name: Run Playwright Tests for Tag
        id: run
        run: |
          echo “Running Playwright tests for tag: ${{ ma-
trix.items }}”
          npx playwright test --grep “${{ matrix.items }}”
# Corrected to use matrix.tags

      - name: Upload test results artifact
        uses: actions/upload-artifact@v4
        with:
          name: blob-report-${{ matrix.items }}  # Use the
sanitized name
          path: blob-report/  # Path where Playwright stores
the reports (update as needed)
          if-no-files-found: ignore
          retention-days: 1

  merge-reports:
    name: Merge Playwright Reports
    needs: test-runner
    if: always()
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Merge reports
        uses: ./.github/actions/merge-reports
```

4. Start a ‘merge the reports job’ file
    a. Merge the reports generated above into a single report file.
    b. Store the merged report in GHA storage management.

```
name: “Merge Blob Reports”
description: “Download all blob reports and generate merged
HTML report”
runs:
  using: “composite”
  steps:
    - uses: actions/checkout@v4

    - uses: actions/setup-node@v4
      with:
        node-version: 18

    - run: npm ci
      shell: bash

    - uses: actions/download-artifact@v4
      with:
        pattern: blob-report-*
        path: blob-report
        merge-multiple: true

    - run: |
        echo “📦 Merging reports...”
        npx playwright merge-reports --reporter html
blob-report
      shell: bash

    - uses: actions/upload-artifact@v4
      with:
        name: merged-playwright-report
        path: playwright-report/
```

TEST DISTRIBUTION ACTIONS THREE WAYS

Strategy 3: Distribute by File Paths

GITHUB BACK TO RECIPE →

1. Modify your Playwright config file.
   a. Turn off parallel execution.
   b. Enable blob reporting format for merging multiple reports.
   c. Set the tests to run in Chromium, Firefox, and WebKit.

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

export default defineConfig({
  testDir: './tests', // Single root directory for Play-
wright to look in
 workers: 2, // Use 4 worker processes

  fullyParallel: false,
  forbidOnly: !!process.env.CI,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [
      ['blob'],
    ],  /* Shared settings for all the projects below. See
https://playwright.dev/docs/api/class-testoptions. */
  use: {
    trace: 'on-first-retry',
    headless: true,
  },

  projects: [
    {
      name: 'chromium',
      use: { ...devices['Desktop Chrome'] },
    }
  ],
});
```

2. Start a composite action file.
3. Create two jobs within the composite action file:
   a. 'Test job' executes the tests using the list of shards from step 3b as input. It also cleans special characters out of the test names so they can be used for generating reports.
   b. 'Merge reports job' aggregates test results and delivers them to the GHA artifact repo once the test job is finished.

```
name: Dynamic Distribution - Paths
on:
  push:
    branches:
      - master

jobs:

  test-runner:
    name: Run Playwright Tests
    strategy:
      matrix:
        items: ["tests", "tests2"]  # Matrix should be an
object with a key 'tags'
    runs-on: ubuntu-latest
    container:
      image: mcr.microsoft.com/playwright:v1.51.1-jammy
    steps:

      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: 18
```

```
      - name: Install dependencies
        run: npm ci

      - name: Run Playwright Tests for path
        id: run
        run: |
          npx playwright test ./${{ matrix.items }}  # Cor-
rected to use matrix.tags

      - name: Upload test results artifact
        uses: actions/upload-artifact@v4
        with:
          name: blob-report-${{ matrix.items }}  # Use the
sanitized name
          path: blob-report/  # Path where Playwright stores
the reports (update as needed)
          if-no-files-found: ignore
          retention-days: 1

  merge-reports:
    name: Merge Playwright Reports
    needs: test-runner
    if: always()
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Merge reports
        uses: ./.github/actions/merge-reports
```

4. Start a 'merge the reports job' file
    a. Merge the reports generated above into a single report file.
    b. Store the merged report in GHA storage management.

```
name: "Merge Blob Reports"
description: "Download all blob reports and generate merged
HTML report"
runs:
  using: "composite"
  steps:
    - uses: actions/checkout@v4

    - uses: actions/setup-node@v4
      with:
        node-version: 18

    - run: npm ci
      shell: bash

    - uses: actions/download-artifact@v4
      with:
        pattern: blob-report-*
        path: blob-report
        merge-multiple: true

    - run: |
        echo "📦 Merging reports..."
        npx playwright merge-reports --reporter html
blob-report
      shell: bash

    - uses: actions/upload-artifact@v4
      with:
        name: merged-playwright-report
        path: playwright-report/
```

WWW.QAWOLF.COM