**Cypress**

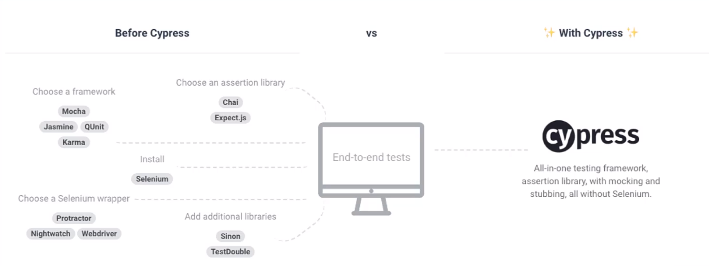
**Introduction:**

Cypress is an open source front end testing tool, build for modern web applications. Cypress is a javascript language binding tool. Cypress is a all in one bundle of many different frameworks like mocha, protractor etc. Cypress is most often compared to Selenium; however, Cypress is both fundamentally and architecturally different. Comparing cypress with selenium is like comparing mobile phone with landline phone, both have similar functionality but uses different architectural style.

We can perform End to end automation testing using Cypress as it has the capability to test both web-based apps testing, API Testing and Database testing.

Most end-to-end testing tools are Selenium based, which is why they all share the same problems. **To make Cypress different, Cypress is built with new architecture from ground up. Whereas Selenium execute remote commands through the network, Cypress runs in the same run-loops as the application.**

Selenium uses Selenium server has a proxy to communication with different browsers via their respective drivers like chromedriver, geckodriver & IE Driver etc. Whereas Cypress runs in the same run-loops as the application so it doesn’t have to relay on this drivers to perform the execution.



This means that when we install Cypress we will get all these wrappers, frameworks & assertion libraries in one framework itself.

**Features of Cypress:**

1. Time Travel: Cypress takes snapshot as your tests run.
2. Real Time Reload: Cypress automatically reloads whenever we make changes in the tests.
3. Spies, stubs and clocks: Verify and control the behavior of functions, server, responses or timers.
4. Consistent results: Cypress architecture doesn’t use Selenium or WebDriver. It provides fast, consistent and reliable tests that are flake-free.
5. Debuggability: Cypress provides debugging functionality by using the chrome dev tool
6. Automatic waiting: Cypress automatically wait for commands and assertions before moving on.
7. Network traffic control: Cypress provided good capabilities to stub network traffic.
8. Screenshot and Videos: In Cypress, we can have screenshot on test failure or videos of entire test suites when run heedlessly.
9. Cross Browser Testing: With Cypress version 4+, cross browser support is added with Edge and Firefox. This feature is still evolving and thus may have some bugs.

These great features come with certain trade-off as mentioned by Cypress. Cypress did support network layer stubbing, but to a small extent. Their future release may have full network layer stubbing.

Similar Cypress team is working on Re-try, unit testing support and Iframe switching, as of now they have small support on Iframe.

[**Cypress Test Runner**](https://www.toolsqa.com/cypress/cypress-test-runner/)**:**

Cypresshas a unique test runner that allows us to see commands as they execute. Additionally, it also shows the real-time run of the application under test.

[**Locator in Cypress**](https://www.toolsqa.com/cypress/cypress-locators/)**:**

Locators are the backbone for all automation frameworks for *Web-based applications*. Additionally, a locator is an identifier that tells any automation tool that *GUI elements* (*say Text Box, Buttons, Check Boxes, etc.*); it needs to operate. Following the same concepts, *Cypress* also uses the locators to identify the UI element for the application under test.

[**Get and Find Command**](https://www.toolsqa.com/cypress/cypress-get-command/)**:**

*Cypress* provides two essential methods *get()* and *find()* to search for the web elements based on the locators. The results for both of these methods are almost identical. But each has its importance and place of implementation.

[**Cypress Asynchronous Nature**](https://www.toolsqa.com/cypress/cypress-asynchronous-nature/)**:**

*Asynchronous programming* is a means of *parallel***programming** whereby, a unit of work runs separately from the main application thread. Additionally, it notifies the calling thread of its completion, failure, or progress. These types of programs are “*non-blocking.”* When you execute something synchronously, you wait for it to finish before moving on to another task.  On the other hand, when you run something asynchronously, you can move on to another task before it ends.

[**Handle Non-Cypress async promises**](https://www.toolsqa.com/cypress/handle-non-cypress-async-promises/)**:**

We know that *Cypress* internally handles the *asynchronous behavior* of its commands and still provides a seamless, sequential, and consistent execution of the test cases. Still, there can be situations, such as combining the Cypress commands with third-party libraries or JavaScript commands, where we have to handle the async promises or commands explicitly.

[**Cypress Assertions**](https://www.toolsqa.com/cypress/cypress-assertions/):

**Assertions**are the validation steps that determine whether the specified step of the automated test case succeeded or not. Cypress uses Chai assertion library as well as Sinon and Jquery for Assertions.

There are two type of waiting mechanism,

1. Implicit assertions – expect()
2. Explicit assertions – Should()

**Data Driven Testing With Cypress:**

Fixtures and Read File are the two different methods available in Cypress to read the data with external file. Cypress reads these file types, but if you specify just the file name without extensions, it will read in the orders as follows,

JSON > JS > Coffee >html > txt > csv > png

**How the Cypress Test Looks like:**

cypress/integration/examples/ea.automation.spec.js

*\*every test file in cypress should be saved with “.spec.js”*

/// <reference types="Cypress" />

// This is testsuite

describe("Testing of EA App", () => {

    // This is testcase

    it("Login Application", () => {

        // Navigating to URL

        cy.visit("http://eaapp.somee.com/");

        // Clicking on Login Button

        cy.contains("Login").click();

        // Verifying URL

        cy.url().should("include", "/Account/Login");

        // Clearing Username field and typing username

        cy.get("#UserName").clear();

        cy.get("#UserName").type("admin");

        // Clearing password field and typing password

        cy.get("#Password").clear();

        cy.get("#Password").type("password");

        // Clicking on login button

        cy.get(".btn").click();

        // Clicking on logout button

        cy.get('#logoutForm > .nav > :nth-child(2) > a').click();

    })

})

**How to execute the Cypress Test from Cypress UI:**

1. Create the file “ea.automation.spec.js” inside “cypress/integration/examples”
2. Copy the above code and paste it in the file.
3. Open terminal in VS Code or any default terminal.
4. Type “npx cypress open”
5. This will open Cypress UI and you can see the newly test case is present in the list of test case.
6. Select “ea.automation.spec.js” and it will automatically open chrome browser and started execution.

**How to execute the Cypress Test from terminal (Headless execution):**

1. Open terminal in VS Code or any default terminal.
2. Type “npx cypress run –-spec cypress\integration\examples\ea.automation.spec.js”
3. This will run the test cases in headless mode.
4. After execution of the test case a new video file of the execution can be found at “cypress\videos\examples\ea.automation.spec.js.mp4”

\*to run execution via command line but on explicitly on Chrome browser,

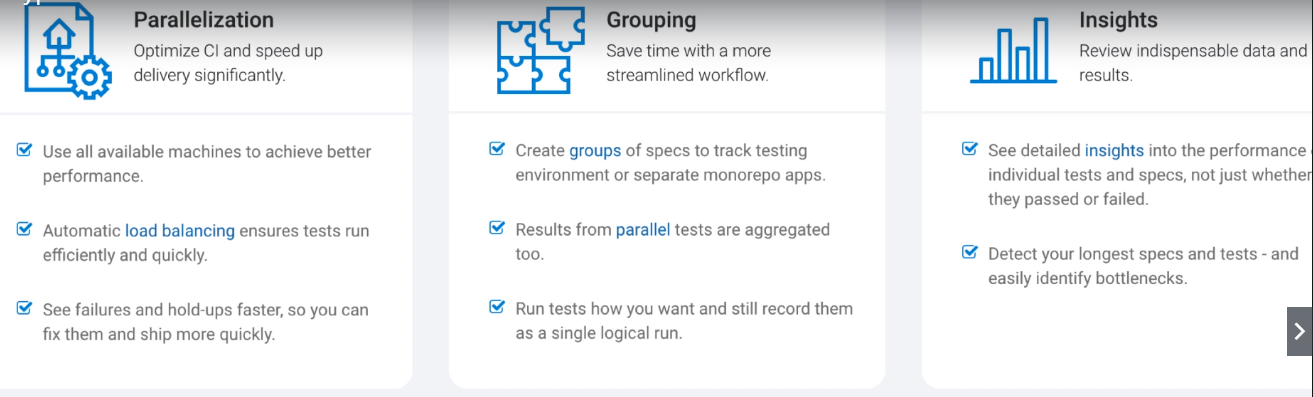
npx cypress run –-spec cypress\integration\examples\ea.automation.spec.js –-browser chrome

**Limitations:**

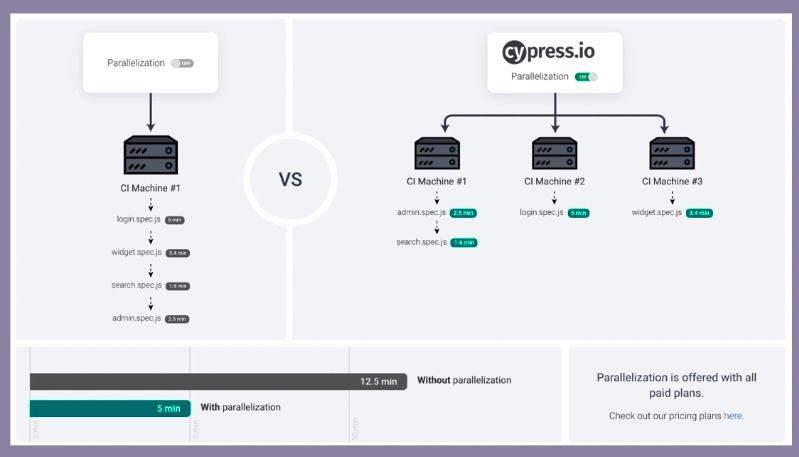
1. We might have to spent more time in debugging the issues as most of the plugins (e.g. Cucumber plugin) are still in developing stating and will get time to fully mature.
2. No support for parallel execution.
3. No iFrame support.
4. Because Cypress runs in the browser, it will never have multi-tabs support.
5. Just like with multiple tabs - Cypress does not support controlling more than 1 open browser at a time.

**Cypress Dashboard:**

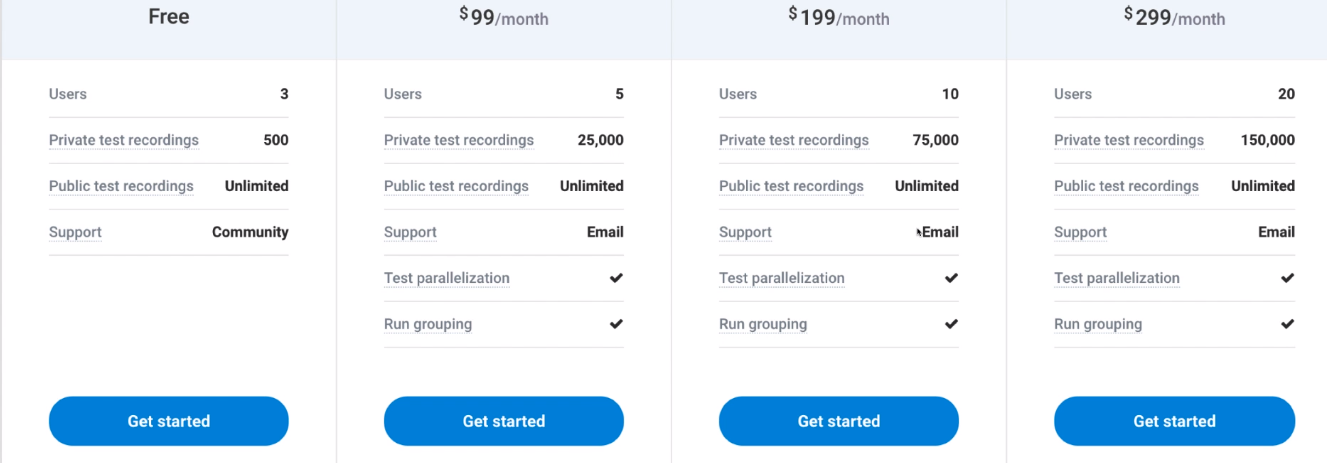
The cypress dashboard services is an option web based companion to the Test Runner. Cypress Dashboard service have various features with which Cypress actually makes money. This is the only feature which enables the parallelization support for Cypress.



How Parallelization works in Cypress:



Costing:



Parallel Execution via Cypress Dashboard:

<https://docs.cypress.io/guides/guides/parallelization.html#Turning-on-parallelization>

**Visual Testing with Cypress & Percy:**

Visual Testing is the automated process of detecting and reviewing visual UI changes. Teams are replacing manual testing with automation to ensure their website and application always look exactly as intended.

Visual Changes in UI or even verifying any cosmetic alignment in application UI was harder, Percy makes the integration smoother and easier.



**Changing browser behavior with Cypress**

If we would like to modify the behavior of browser. This can be done by browser events in Cypress. Cypress emits a series of events as it runs in browser. These events are useful not only to control the application’s behavior, but also helpful in debugging purposes.

Some actions are:

1. Maximizing the window
2. Incognito mode
3. Set chrome extension etc.

**Cypress Reporting:**

Cypress is built on top of Mocha, that means any reporter build for Mocha can be used with Cypress. Below are the list of built in Mocha reporters,

1. Spec
2. XUnit
3. JSON
4. Json Stream
5. JUnit
6. HTML report etc

Cypress executes each spec in isolation. This leads to Mocha generating a separate test report for each spec. Thus, there’s no out-of-box solution to generate one mochawesome report for all specs.

But in order to make all spec as one file, we need to install one more package which is mochawesome-merge.

Steps to generate report:

1. Update **“cypress.json”** file with below:

"reporter": "mochawesome",

    "reporterOptions": {

        "reportDir": "cypress/report/mochawesome-report",

        "overwrite": false,

        "html": false,

        "json": true,

        "timestamp": "mmddyyyy\_HHMMss"

    }

1. Add below data to package.json:

{

    "scripts": {

        "test": "cypress run --headless"

    },

    "devDependencies": {

        "@percy/cypress": "^2.3.1",

        "cypress": "^4.11.0",

        "cypress-dark": "^1.7.14",

        "cypress-file-upload": "^4.0.7",

        "mocha": "5.2.0",

        "mochawesome": "4.1.0",

        "mochawesome-merge": "4.0.3",

        "cypress-multi-reporters": "1.4.0",

        "mochawesome-report-generator": "5.1.0"

    },

    "dependencies": {

        "@reportportal/agent-js-cypress": "^5.0.1"

    }

}

1. Create a folder in cypress/report.
2. Run the cypress execution using below command:

***npx cypress run***

* this will generate json files for each specs

1. Now run below command to merge all the json files in one json file,

***npx mochawesome-merge cypress/report/mochawesome-report/mochawesome\*.json>mochareports\report.json***

* We have to create folder “mochareports”

1. Now run below command to generate HTML report,

***npx marge mochareports\report.json -f report -o mochareport***