← Mobile Testing with WebDriverIO

AWS for Test Automation →

BrowserStack Integration

Table of Contents

Cross-Browser and Cross-Device Testing Strategy

Why Cloud Testing?

Challenges of Local Testing:

- Limited device/browser availability
- Maintenance overhead

- Scalability issues
- No parallel execution

Benefits of Cloud Testing:

- Access to 3000+ real devices
- All browser versions
- No maintenance
- Parallel execution
- Instant scalability
- Videos and screenshots
- Network simulation
- Debugging tools

BrowserStack Overview

BrowserStack provides cloud infrastructure for testing across browsers, operating systems, and mobile devices.

Features:

- Real devices (not emulators)
- Live testing
- Automated testing
- Local testing
- Visual testing
- Accessibility testing
- Performance monitoring

Products:

- **Live**: Manual interactive testing
- **Automate**: Web automation
- **App Automate**: Mobile app automation

Testing Strategy

Browser Coverage:

```
Desktop Browsers:
- Chrome (latest, latest-1, latest-2)
- Firefox (latest, latest-1)
- Safari (latest, latest-1)
- Edge (latest, latest-1)
- Opera (latest)

Mobile Browsers:
- Safari (iOS latest, latest-1)
- Chrome (Android latest, latest-1)
- Samsung Internet
```

Device Coverage:

```
Popular Devices:
- iPhone 15, 14, 13
- Samsung Galaxy S23, S22, S21
- Google Pixel 7, 6
- iPad Pro, Air

Screen Resolutions:
- 1920x1080 (Desktop)
- 1366x768 (Laptop)
- 768x1024 (Tablet)
- 375x667 (Mobile)
```

Test Matrix:

```
Priority 1 (Critical):
- Chrome latest + Windows
- Safari latest + macOS
- Chrome latest + Android
- Safari latest + iOS

Priority 2 (High):
```

```
Priority 3 (Medium):
- All latest-1 versions
- Popular devices
```

Account Setup

1. Sign up:

- Visit browserstack.com
- Create account (free trial available)
- Get credentials

2. Get credentials:

```
 Username: your_username
Access Key: your_access_key
```

3. Store securely:

```
 # .env file (add to .gitignore)
BROWSERSTACK_USERNAME=your_username
BROWSERSTACK_ACCESS_KEY=your_access_key
```

Integrating Playwright with BrowserStack

Installation

Install BrowserStack SDK:

```
 npm install --save-dev @browserstack/playwright-helper
```

```
 # browserstack.yml
auth:
  username: ${BROWSERSTACK_USERNAME}
  access_key: ${BROWSERSTACK_ACCESS_KEY}

browsers:
  - os: Windows
    osVersion: 11
    browserName: Chrome
    browserVersion: latest
  - os: OS X
    osVersion: Ventura
    browserName: Safari
    browserVersion: latest
  - os: Windows
    osVersion: 11
    browserName: Edge
    browserVersion: latest

run_settings:
  timeout: 0
  project: My Project
  build: Build 1
  name: Playwright Test
  buildIdentifier: ${BUILD_NUMBER}
  parallels: 5
```

playwright.config.ts:

```
 import { defineConfig } from '@playwright/test'
import { config as dotenvConfig } from 'dotenv'

dotenvConfig()

export default defineConfig({
  testDir: './tests',
  timeout: 30000,
  retries: 1,
  workers: 5,

  use: {
```

```
    video: 'retain-on-failure',
    trace: 'retain-on-failure',
  },

  projects: [
    {
      name: 'chrome-windows',
      use: {
        browserName: 'chromium',
        channel: 'chrome',
        connectOptions: {
          wsEndpoint: 'wss://cdp.browserstack.com/playwright',
        },
      },
    },
    {
      name: 'safari-mac',
      use: {
        browserName: 'webkit',
        connectOptions: {
          wsEndpoint: 'wss://cdp.browserstack.com/playwright',
        },
      },
    },
  ],
})
```

Using BrowserStack with Playwright

Basic test:

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

test('BrowserStack test', async ({ page }) => {
  await page.goto('https://www.browserstack.com')

  const title = await page.title()
  expect(title).toContain('BrowserStack')

  await page.screenshot({ path: 'browserstack-home.png' })
})
```

```
import { chromium } from 'playwright'

const caps = {
  'browser': 'chrome',
  'browser_version': 'latest',
  'os': 'Windows',
  'os_version': '11',
  'name': 'Playwright Test',
  'build': 'Build 1',
  'browserstack.username': process.env.BROWSERSTACK_USERNAME,
  'browserstack.accessKey': process.env.BROWSERSTACK_ACCESS_KEY,
}

const browser = await chromium.connect({
  wsEndpoint: `wss://cdp.browserstack.com/playwright?caps=${encodeURICompo
})

const page = await browser.newPage()
await page.goto('https://example.com')

// Your test code

await browser.close()
```

Session Management

Mark test status:

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

test.afterEach(async ({ page }, testInfo) => {
  const status = testInfo.status === 'passed' ? 'passed' : 'failed'
  const reason = testInfo.error?.message || ''

  await page.evaluate(() => {}, `browserstack_executor: ${JSON.stringify({
    action: 'setSessionStatus',
    arguments: { status, reason }
  })}`)
})
```

Set session name:

```
    arguments: { name: testInfo.title }
  })}`)
})
```

Local Testing

Test apps on localhost or private networks:

Install BrowserStack Local:

```
npm install --save-dev browserstack-local
```

Start local connection:

```
// setup-browserstack-local.ts
import * as browserstack from 'browserstack-local'

export async function startBrowserStackLocal() {
  const bs_local = new browserstack.Local()

  return new Promise((resolve, reject) => {
    bs_local.start({
      key: process.env.BROWSERSTACK_ACCESS_KEY,
      force: 'true',
      onlyAutomate: 'true',
    }, (error: any) => {
      if (error) reject(error)
      else resolve(bs_local)
    })
  })
}

export async function stopBrowserStackLocal(bs_local: any) {
  return new Promise((resolve) => {
    bs_local.stop(() => resolve(true))
  })
}
```

Use in tests:

```
let bsLocal: any

test.beforeAll(async () => {
  bsLocal = await startBrowserStackLocal()
})

test.afterAll(async () => {
  await stopBrowserStackLocal(bsLocal)
})

test('local test', async ({ page }) => {
  await page.goto('http://localhost:3000')
  // Test local application
})
```

Parallel Execution

Configure parallel workers:

```
 // playwright.config.ts
export default defineConfig({
  workers: 5, // Run 5 tests in parallel

  projects: [
    { name: 'chrome-win11' },
    { name: 'safari-monterey' },
    { name: 'edge-win11' },
    { name: 'firefox-win10' },
    { name: 'chrome-mac' },
  ]
})
```

Run tests:

```
npx playwright test --project=chrome-win11 --project=safari-monterey
```

Installation

Install plugin:

```
npm install --save-dev browserstack-cypress-cli
```

Configuration

Create browserstack.json:

```
{
 "auth": {
   "username": "${BROWSERSTACK_USERNAME}",
   "access_key": "${BROWSERSTACK_ACCESS_KEY}"
 },
 "browsers": [
   {
     "browser": "chrome",
     "os": "Windows 11",
     "versions": ["latest", "latest-1"]
   },
   {
     "browser": "firefox",
     "os": "Windows 10",
     "versions": ["latest"]
   },
   {
     "browser": "edge",
     "os": "Windows 11",
     "versions": ["latest"]
   },
   {
     "browser": "safari",
     "os": "OS X Ventura",
     "versions": ["latest"]
   }
 ],
 "run_settings": {
   "cypress_proj_dir": ".",
   "project_name": "My Cypress Project",
```

```
      "@faker-js/faker": "^7.6.0"
    },
    "package_config_options": {
      "video": true,
      "screenshotsFolder": "cypress/screenshots",
      "videosFolder": "cypress/videos"
    }
  },
  "connection_settings": {
    "local": false,
    "local_identifier": null
  },
  "disable_usage_reporting": false
}
```

cypress.config.js:

```
const { defineConfig } = require('cypress')

module.exports = defineConfig({
  e2e: {
    baseUrl: 'http://localhost:3000',
    specPattern: 'cypress/e2e/**/*.cy.{js,jsx,ts,tsx}',
    supportFile: 'cypress/support/e2e.js',

    setupNodeEvents(on, config) {
      // BrowserStack setup
      return config
    },
  },
})
```

Running Tests on BrowserStack

Initialize:

```
npx browserstack-cypress init
```

Run tests:

Run specific browsers:

```
npx browserstack-cypress run --browser chrome
```

Run with local testing:

```
npx browserstack-cypress run --local
```

BrowserStack Commands

Install Cypress plugin:

```
npm install --save-dev cypress-browserstack
```

cypress/support/e2e.js:

```
require('cypress-browserstack')
```

Mark test status:

```
describe('BrowserStack Tests', () => {
 afterEach(function() {
   cy.browserstack('setSessionStatus', {
     status: this.currentTest.state === 'passed' ? 'passed' : 'failed',
     reason: this.currentTest.err?.message || ''
   })
 })

 it('test example', () => {
   cy.visit('https://example.com')
   cy.get('h1').should('contain', 'Example')
 })
})
```

```
{
  "connection_settings": {
    "local": true,
    "local_identifier": "my-project"
  }
}
```

Run with local:

```
npx browserstack-cypress run --local
```

Parallel Execution

Configure parallels:

```
{
  "run_settings": {
    "parallels": 10
  }
}
```

Run:

```
npx browserstack-cypress run --parallels 10
```

Integrating WebDriverIO with BrowserStack

Installation

Install service:

Configuration

wdio.conf.ts:

```
export const config = {
  user: process.env.BROWSERSTACK_USERNAME,
  key: process.env.BROWSERSTACK_ACCESS_KEY,

  services: [
    ['browserstack', {
      browserstackLocal: false,
      opts: {
        forceLocal: false,
      }
    }]
  ],

  capabilities: [
    {
      browserName: 'Chrome',
      'bstack:options': {
        os: 'Windows',
        osVersion: '11',
        browserVersion: 'latest',
        projectName: 'WebDriverIO Tests',
        buildName: 'Build 1',
        sessionName: 'Chrome Test',
        debug: true,
        networkLogs: true,
        consoleLogs: 'verbose',
        video: true,
      }
    },
    {
      browserName: 'Safari',
      'bstack:options': {
        os: 'OS X',
        osVersion: 'Ventura',
        browserVersion: 'latest',
        projectName: 'WebDriverIO Tests',
        buildName: 'Build 1',
        sessionName: 'Safari Test',
      }
```

```
  maxInstances: 5,
  logLevel: 'info',
  bail: 0,
  waitforTimeout: 10000,
  connectionRetryTimeout: 120000,
  connectionRetryCount: 3,

  framework: 'mocha',
  reporters: ['spec'],

  mochaOpts: {
    ui: 'bdd',
    timeout: 60000
  },

  afterTest: async function(test, context, { error, result, passed }) {
    if (!passed) {
      await browser.execute(
        'browserstack_executor: {"action": "setSessionStatus", "arguments"
        error.message + '"}}'
      )
    } else {
      await browser.execute(
        'browserstack_executor: {"action": "setSessionStatus", "arguments"
      )
    }
  }
}
```

Mobile Testing Configuration

wdio.conf.ts for mobile:

```
capabilities: [
  {
    'bstack:options': {
      deviceName: 'Samsung Galaxy S23',
      platformVersion: '13.0',
      platformName: 'Android',
      app: process.env.BROWSERSTACK_APP_ID,
      projectName: 'Mobile App Tests',
      buildName: 'Android Build 1',
      sessionName: 'Galaxy S23 Test',
```

```
    }
  },
  {
    'bstack:options': {
      deviceName: 'iPhone 15',
      platformVersion: '17',
      platformName: 'iOS',
      app: process.env.BROWSERSTACK_APP_ID,
      projectName: 'Mobile App Tests',
      buildName: 'iOS Build 1',
      sessionName: 'iPhone 15 Test',
    }
  }
]
```

Local Testing

wdio.conf.ts:

```
services: [
  ['browserstack', {
    browserstackLocal: true,
    opts: {
      key: process.env.BROWSERSTACK_ACCESS_KEY,
      forceLocal: true,
      localIdentifier: 'my-project',
    }
  }]
]
```

Custom Executors

Mark session status:

```
it('test example', async () => {
  await browser.url('https://example.com')

  // Your test code

  // Mark as passed
```

```
})
```

Set session name:

```
beforeEach(async function() {
  await browser.execute(
    `browserstack_executor: {"action": "setSessionName", "arguments": {"na
  )
})
```

Parallel Test Execution

Strategy for Parallel Execution

Parallel levels:

```
1. Suite level - Different test suites
2. File level - Different spec files
3. Test level - Individual tests
4. Browser level - Same tests, different browsers
```

Playwright Parallel Execution

playwright.config.ts:

```
export default defineConfig({
  workers: 10, // 10 parallel workers
  fullyParallel: true, // All tests run in parallel

  projects: [
    {
      name: 'chrome-win11',
      use: { /* ... */ },
    },
```

```
    },
    {
      name: 'safari-ventura',
      use: { /* ... */ },
    },
  ]
})
```

Run command:

```
 # All projects in parallel
npx playwright test

# Specific projects
npx playwright test --project=chrome-win11 --project=safari-ventura

# With custom workers
npx playwright test --workers=5
```

Cypress Parallel Execution

browserstack.json:

```
 {
  "run_settings": {
    "parallels": 10
  }
}
```

Run:

```
npx browserstack-cypress run --parallels 10
```

With Cypress Cloud:

```
cypress run --record --parallel --group "Chrome Tests"
```

```
export const config = {
  maxInstances: 10, // Max parallel instances

  capabilities: [
    { browserName: 'chrome' },
    { browserName: 'firefox' },
    { browserName: 'safari' },
    { browserName: 'edge' },
  ],

  specs: [
    './test/specs/**/*.ts'
  ]
}
```

Run:

```
npx wdio run wdio.conf.ts
```

Optimizing Parallel Execution

Best practices:

```
// 1. Independent tests
// Each test should be able to run in isolation

// 2. No shared state
// Avoid global variables or shared data

// 3. Unique test data
// Generate unique data per test
const uniqueEmail = `test-${Date.now()}@example.com`

// 4. Cleanup
afterEach(async () => {
  // Clean up test data
})
```

Calculate optimal workers:

```
 workers = min(
   number_of_tests / average_test_duration,
   browserstack_parallel_limit,
   available_memory / memory_per_worker
 )

 Example:
 - 100 tests
 - Average duration: 30 seconds
 - BrowserStack limit: 5 parallel
 - Optimal workers: 5
```

Debugging Cloud Test Failures

BrowserStack Dashboard

Access dashboard:

1. Login to browserstack.com
2. Go to Automate/App Automate
3. View test sessions
4. Click on session for details

Session details include:

- Video recording
- Screenshots
- Console logs
- Network logs
- Selenium/Appium logs
- Test commands

Video Recording

Enable video:

```
// Playwright
screenshot: 'only-on-failure',
video: 'retain-on-failure',

// Cypress (browserstack.json)
"package_config_options": {
  "video": true
}

// WebDriverIO
'bstack:options': {
  video: true
}
```

Download video:

```
# Via API
curl -u "USERNAME:ACCESS_KEY" \
  "https://api.browserstack.com/automate/sessions/SESSION_ID/video.mp4" \
  -o video.mp4
```

Console Logs

Enable console logs:

```
// WebDriverIO
'bstack:options': {
  consoleLogs: 'verbose'
}
```

Access logs:

```
const logs = await browser.getLogs('browser')
console.log(logs)
```

Enable network logs:

```
'bstack:options': {
  networkLogs: true
}
```

Access via dashboard:

- Click on session
- Go to "Network" tab
- View all requests/responses
- Filter by type, status, etc.

Screenshots on Failure

Playwright:

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

test.afterEach(async ({ page }, testInfo) => {
  if (testInfo.status !== 'passed') {
    const screenshot = await page.screenshot()
    await testInfo.attach('screenshot', {
      body: screenshot,
      contentType: 'image/png'
    })
  }
})
```

Cypress:

```
afterEach(function() {
  if (this.currentTest.state === 'failed') {
    cy.screenshot(`FAILED-${this.currentTest.title}`)
  }
})
```

```
afterTest: async function(test, context, { passed }) {
  if (!passed) {
    await browser.saveScreenshot(
      `./screenshots/FAILED-${test.title}.png`
    )
  }
}
```

Debug Mode

Enable debug:

```
'bstack:options': {
  debug: true,
  networkLogs: true,
  consoleLogs: 'verbose'
}
```

Common Issues and Solutions

1. Element not found:

```
// Issue: Element timing
// Solution: Add explicit wait
await page.waitForSelector('.element', { timeout: 10000 })

// Solution: Use retry logic
await browser.waitUntil(async () => {
  return await $('.element').isDisplayed()
}, { timeout: 10000 })
```

2. Flaky tests:

```
// Issue: Race conditions
// Solution: Wait for page load
await page.waitForLoadState('networkidle')

// Solution: Wait for specific condition
```

3. Timeout errors:

```
 // Issue: Default timeout too short
// Solution: Increase timeout
timeout: 60000, // 60 seconds

// Solution: Wait for specific state
await page.waitForSelector('.element', {
  state: 'visible',
  timeout: 30000
})
```

4. Session not found:

```
 // Issue: Session expired
// Solution: Use session keep-alive
'bstack:options': {
  idleTimeout: 300 // 5 minutes
}
```

REST API for Debugging

Get session details:

```
curl -u "USERNAME:ACCESS_KEY" \
  "https://api.browserstack.com/automate/sessions/SESSION_ID.json"
```

Get session logs:

```
curl -u "USERNAME:ACCESS_KEY" \
  "https://api.browserstack.com/automate/sessions/SESSION_ID/logs"
```

Delete session:

```
https://api.browserstack.com/automate/sessions/SESSION_ID.json
```

Practice Exercises

Exercise 1: Multi-Browser Playwright Setup

Task: Configure and run Playwright on multiple browsers

```
// playwright.config.ts
// Configure for:
// - Chrome (Windows 11)
// - Firefox (Windows 10)
// - Safari (macOS Ventura)
// - Edge (Windows 11)

// Create test that:
// 1. Runs on all browsers
// 2. Takes screenshots
// 3. Records video on failure
// 4. Marks session status

// Your implementation
```

Exercise 2: Cypress Parallel Execution

Task: Setup Cypress with BrowserStack parallel testing

```
// browserstack.json
// Configure for:
// - 5 parallel executions
// - Chrome, Firefox, Safari, Edge
// - Video recording enabled
// - Local testing enabled

// Create tests that:
// 1. Run independently
```

```
// Your implementation
```

Exercise 3: Mobile App Testing

Task: Test mobile app on multiple devices

```
 // wdio.conf.ts
// Configure for:
// - Samsung Galaxy S23 (Android 13)
// - iPhone 15 (iOS 17)
// - Google Pixel 7 (Android 13)

// Create tests that:
// 1. Test app on all devices
// 2. Handle device-specific UI
// 3. Test both orientations
// 4. Capture screenshots on failure

// Your implementation
```

Exercise 4: Local Testing Setup

Task: Test local application on cloud browsers

```
 // Setup local testing for:
// - Playwright
// - Cypress
// - WebDriverIO

// Test localhost:3000 application
// Verify local tunnel works
// Run tests on cloud browsers
// Debug any connection issues

// Your implementation
```

Exercise 5: Comprehensive Test Suite

```
// Create test suite that:
// 1. Tests on 5 browsers
// 2. Runs 10 tests in parallel
// 3. Includes web and mobile
// 4. Captures all debugging info
// 5. Reports to dashboard
// 6. Marks all session statuses

// Tests should cover:
// - Login flow
// - Form submission
// - Navigation
// - Data validation
// - Error handling

// Your implementation
```

Exercise 6: Debugging Practice

Task: Debug failing cloud tests

```
// Given a failing test:
// 1. Review video recording
// 2. Check console logs
// 3. Analyze network logs
// 4. Review screenshots
// 5. Identify root cause
// 6. Fix the issue
// 7. Verify fix on cloud

// Document your debugging process

// Your implementation
```

Exercise 7: Performance Testing

Task: Measure parallel execution performance

```
// Measure and compare:
// 1. Sequential execution time
```

```
// Calculate:
// - Time saved
// - Optimal worker count
// - Cost efficiency
// - Resource usage

// Create report with findings

// Your implementation
```

Summary

In Day 11, you mastered:

✅ Cloud Testing Strategy

- Cross-browser coverage
- Device selection
- Test matrix design
- Priority planning

✅ Playwright + BrowserStack

- Configuration setup
- Capabilities management
- Session control
- Local testing
- Parallel execution

✅ Cypress + BrowserStack

- Plugin installation
- JSON configuration
- CLI usage

✅ WebDriverIO + BrowserStack

- Service integration
- Mobile capabilities
- Custom executors
- Debug options

✅ Parallel Execution

- Strategy design
- Worker optimization
- Independent tests
- Resource management

✅ Debugging

- Dashboard navigation
- Video analysis
- Log inspection
- Network debugging
- Screenshot capture
- API usage

Key Takeaways

- **Cloud testing** eliminates device maintenance
- **BrowserStack** provides comprehensive coverage
- **Parallel execution** dramatically reduces test time
- **Proper configuration** is critical for success
- **Debugging tools** are essential for troubleshooting
- **Session management** provides test traceability

2. Use data-testid for stable selectors
3. Enable all debugging features
4. Mark session status always
5. Implement retry logic
6. Use local testing for staging
7. Monitor BrowserStack usage/costs
8. Keep credentials secure
9. Use build identifiers
10. Review failed sessions immediately

Integration Checklist

- [ ] BrowserStack account setup
- [ ] Credentials stored securely
- [ ] Configuration files created
- [ ] Tests run locally first
- [ ] Capabilities configured correctly
- [ ] Parallel execution tested
- [ ] Local testing working
- [ ] Session status marked
- [ ] Debugging enabled
- [ ] Dashboard accessible

Cost Optimization

Tips to reduce costs:

- Use parallel wisely (don't over-parallelize)
- Set appropriate timeouts
- Clean up sessions
- Use local testing for dev

- Delete old builds
- Optimize test duration

Next Steps

- **Day 12**: AWS for Test Automation
- **Day 13**: CI/CD & Test Reporting
- **Day 14**: AI in Test Automation
- **Day 15**: Framework Design & Capstone

Only 4 more days to complete the program! 🎯

End of Day 11 Documentation

← Mobile Testing with WebDriverIO

AWS for Test Automation →