Playwright와 Artillery를 활용한 E2E 테스트 및 부하 테스트

대규모 서비스 개발 및 운영을 위한 실전 엔지니어링 워크숍

목차

1. E2E 테스트와 부하테스트의 중요성

1.1 E2E 테스트의 필요성

- **사용자 관점의 검증**: 실제 사용자 경험과 동일한 시나리오 테스트
- **전체 시스템 통합 검증**: 모든 컴포넌트 간의 상호작용 확인
- **회귀 테스트 자동화**: 새로운 기능 추가 시 기존 기능 영향도 검증

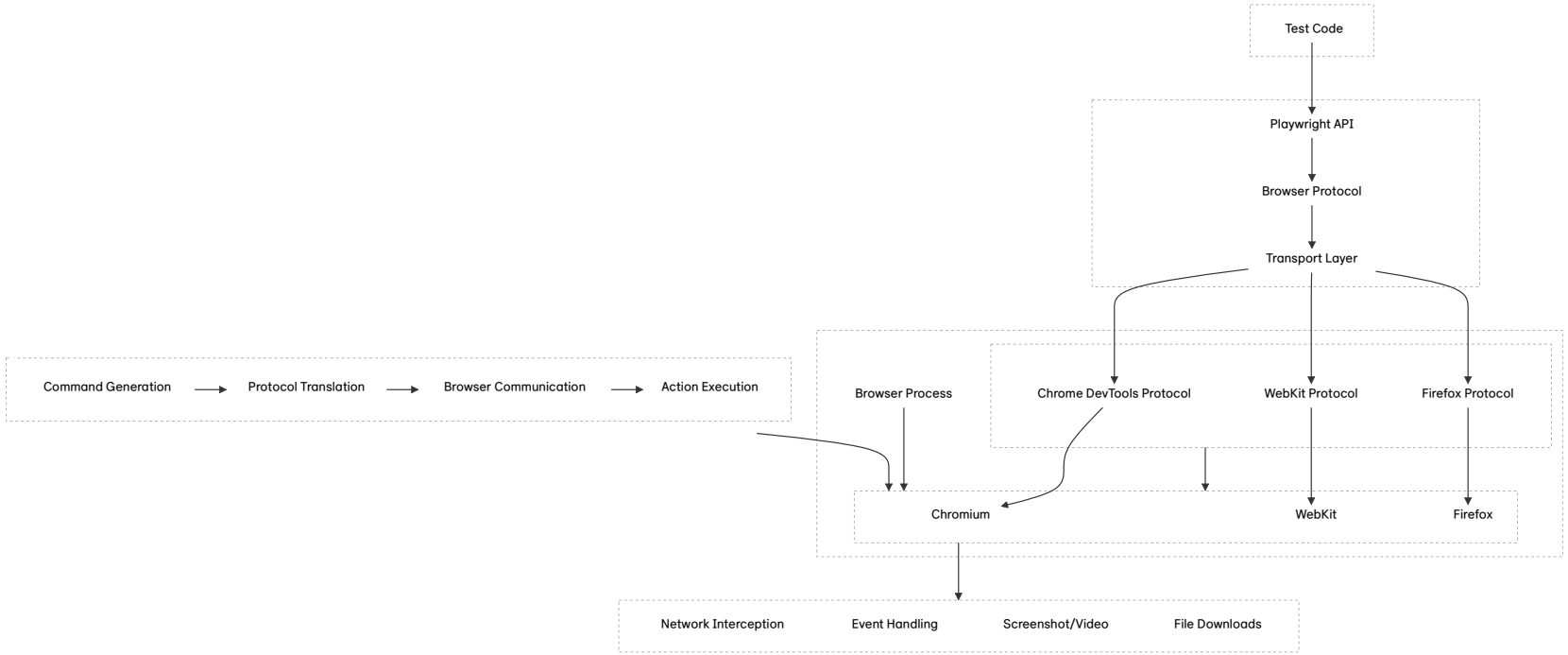
1.2 부하테스트의 중요성

- **시스템 한계 파악**: 최대 처리 용량 및 병목 지점 식별
- **안정성 검증**: 높은 부하 상황에서의 시스템 동작 확인
- **성능 최적화**: 성능 개선 포인트 도출

2. Playwright와 Artillery 소개

2.1 Playwright의 특징

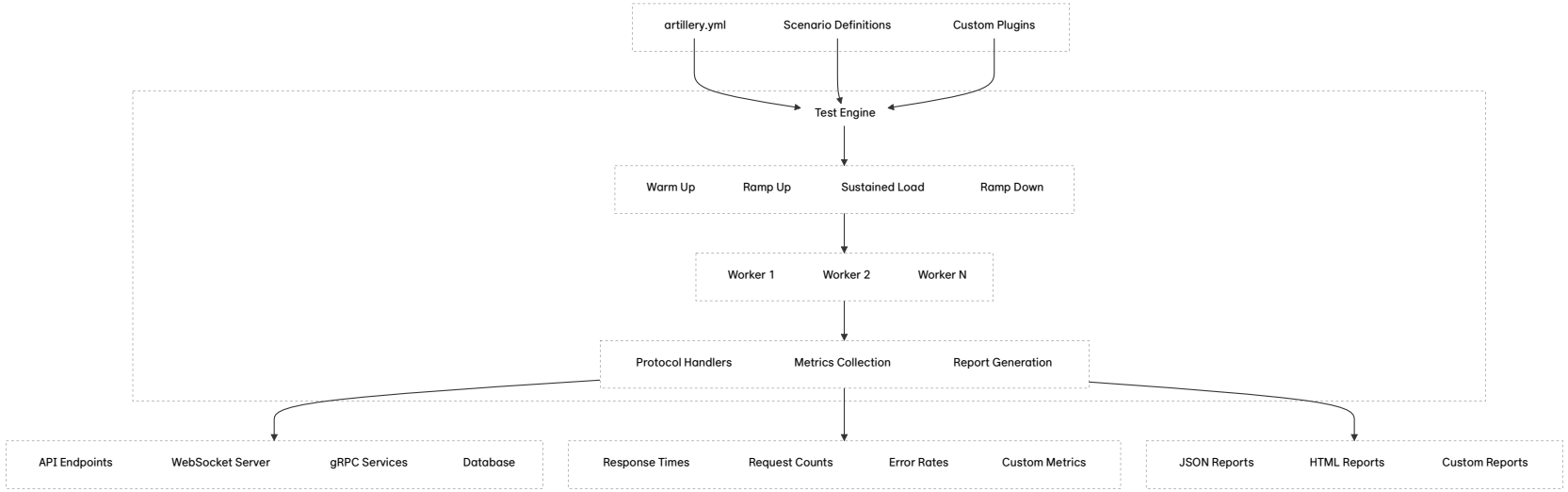
- **크로스 브라우저 지원**
  - Chromium, Firefox, WebKit 엔진 지원
  - 모든 주요 브라우저에서 일관된 테스트 가능
- **강력한 자동화 기능**
  - 네트워크 요청 인터셉트
  - 파일 업로드/다운로드 자동화
  - 모바일 디바이스 에뮬레이션



2. Playwright와 Artillery 소개

2.2 Artillery의 특징

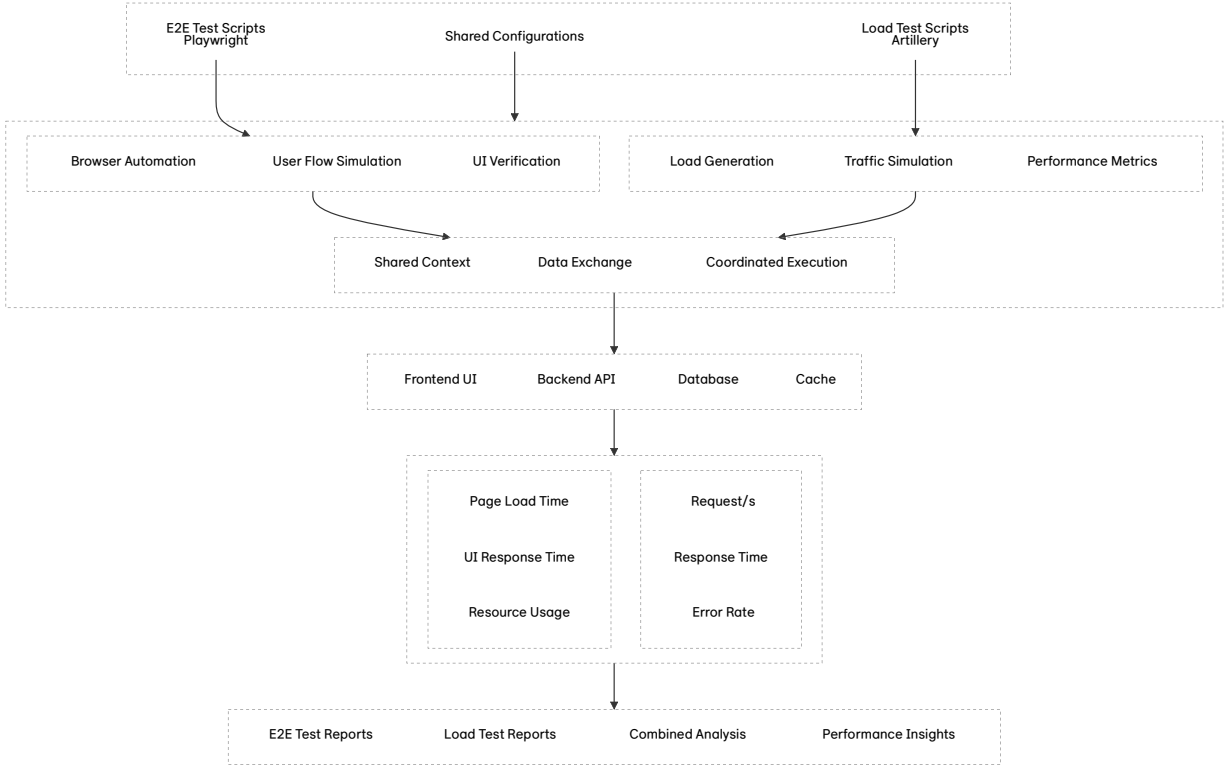
- **확장성 있는 부하 생성**
  - 점진적 부하 증가
  - 다양한 부하 패턴 지원
  - 분산 부하 테스트 가능
- **다양한 프로토콜 지원**
  - HTTP/HTTPS
  - WebSocket
  - Socket.io
  - gRPC



2. Playwright와 Artillery 소개

2.3 두 도구의 결합 효과

- **현실적인 부하 시나리오**
  - 실제 사용자 행동 기반 부하 생성
  - 복잡한 비즈니스 로직 테스트 가능
- **종합적인 성능 분석**
  - 클라이언트 사이드 성능 측정
  - 서버 사이드 부하 모니터링



3. 개발 환경 설정

3.1 Playwright 설치 및 설정

```
# Playwright 설치
npm init playwright@latest

# 브라우저 설치
npx playwright install
```

3. 개발 환경 설정

3.2 Artillery 설치 및 설정

```
# Artillery 전역 설치
npm install -g artillery

# 프로젝트 의존성으로 설치
npm install --save-dev artillery
```

3. 개발 환경 설정

3.3 통합 환경 구성

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

const config: PlaywrightTestConfig = {
  testDir: './tests',
  timeout: 30000,
  use: {
    headless: true,
    viewport: { width: 1280, height: 720 },
    video: 'on-first-retry',
  },
};

export default config;
```

3. 개발 환경 설정

```
# artillery.yml
config:
  target: "http://your-target-service.com"
  phases:
    - duration: 60
      arrivalRate: 5
      rampTo: 50
  processor: "./scenarios.js"
```

4. Playwright 기본 사용법

4.1 요소 선택기(Selectors) 이해

```
// 다양한 선택자 예시
await page.locator('#login-button').click();  // ID 선택자
await page.locator('.menu-item').click();     // 클래스 선택자
await page.getByRole('button').click();       // 역할 기반 선택자
await page.getByText('로그인').click();        // 텍스트 기반 선택자
```

4. Playwright 기본 사용법

4.2 동기/비동기 처리

```
test('비동기 작업 처리 예시', async ({ page }) => {
  // 페이지 로딩 대기
  await page.goto('https://example.com');

  // 요소가 나타날 때까지 대기
  await page.waitForSelector('.dynamic-content');

  // 네트워크 요청 완료 대기
  const responsePromise = page.waitForResponse('**/api/data');
  await page.click('#load-data');
  const response = await responsePromise;
});
```

4. Playwright 기본 사용법

4.3 주요 API 활용

```
test('주요 API 활용 예시', async ({ page }) => {
  // 키보드 입력
  await page.keyboard.type('Hello World');

  // 마우스 조작
  await page.mouse.move(100, 200);
  await page.mouse.click(100, 200);

  // 파일 업로드
  await page.setInputFiles('input[type="file"]', 'path/to/file.jpg');

  // 네트워크 요청 모니터링
  page.on('request', request => {
    console.log(request.url());
  });
});
```

5. Artillery 기본 사용법

5.1 YAML 설정 파일 작성

```yaml
config:
  target: "http://api.example.com"
  http:
    timeout: 10
  phases:
    - duration: 60
      arrivalRate: 5
      name: "Warm up"
    - duration: 120
      arrivalRate: 10
      rampTo: 50
      name: "Ramp up load"
    - duration: 600
      arrivalRate: 50
      name: "Sustained load"
```

5. Artillery 기본 사용법

5.2 시나리오 정의

```yaml
scenarios:
  - name: "사용자 로그인 및 데이터 조회"
    flow:
      - post:
          url: "/auth/login"
          json:
            username: "{{ $processEnvironment.USERNAME }}"
            password: "{{ $processEnvironment.PASSWORD }}"
          capture:
            - json: "$.token"
              as: "authToken"

      - get:
          url: "/api/data"
          headers:
            Authorization: "Bearer {{ authToken }}"
```

5. Artillery 기본 사용법

5.3 부하 패턴 설정

```
config:
  phases:
    # 점진적 부하 증가
    - duration: 300
      arrivalRate: 1
      rampTo: 100

    # 급격한 스파이크 부하
    - duration: 60
      arrivalCount: 1000

    # 지속적 부하
    - duration: 3600
      arrivalRate: 50
```

6. 실전 테스트 시나리오

6.1 로그인 부하 테스트

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

test('대량 로그인 테스트', async ({ page }) => {
  await page.goto('/login');

  await page.fill('#username', 'test@example.com');
  await page.fill('#password', 'password123');

  const responsePromise = page.waitForResponse('**/api/login');
  await page.click('#login-button');

  const response = await responsePromise;
  expect(response.status()).toBe(200);
});
```

6. 실전 테스트 시나리오

```yaml
# artillery-login.yml
scenarios:
  - name: "동시 다중 로그인"
    flow:
      - post:
          url: "/api/login"
          json:
            username: "{{ $randomString() }}@example.com"
            password: "password123"
```

6. 실전 테스트 시나리오

6.2 채팅 메시지 전송 테스트

```
// chat.test.ts
test('채팅 메시지 전송', async ({ page }) => {
  await page.goto('/chat');

  for (let i = 0; i < 100; i++) {
    await page.fill('#message-input', `테스트 메시지 ${i}`);
    await page.click('#send-button');
    await page.waitForSelector(`text=테스트 메시지 ${i}`);
  }
});
```

6. 실전 테스트 시나리오

```
# artillery-chat.yml
scenarios:
  - name: "채팅 부하 테스트"
    engine: "websocket"
    flow:
      - connect: "ws://chat.example.com"
      - think: 1
      - send:
          channel: "message"
          data: "Hello from Artillery #{{ $randomNumber(1, 1000) }}"
```

6. 실전 테스트 시나리오

6.3 파일 업로드/다운로드 테스트

```
// file-operations.test.ts
test('파일 업로드 및 다운로드', async ({ page }) => {
  // 업로드 테스트
  const [fileChooser] = await Promise.all([
    page.waitForEvent('filechooser'),
    page.click('#upload-button')
  ]);
  await fileChooser.setFiles('./test-files/sample.pdf');

  // 다운로드 테스트
  const [download] = await Promise.all([
    page.waitForEvent('download'),
    page.click('#download-link')
  ]);
  await download.saveAs('./downloads/downloaded-file.pdf');
});
```

7. 테스트 실행 및 모니터링

7.1 병렬 테스트 실행

```
# Playwright 병렬 실행
npx playwright test --workers=5

# Artillery 병렬 실행
artillery run --split 4 scenario.yml
```

7. 테스트 실행 및 모니터링

7.2 실시간 모니터링

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

test.beforeEach(async ({ page }) => {
  // 성능 메트릭 수집
  page.on('metrics', data => {
    console.log(JSON.stringify(data.metrics));
  });

  // 네트워크 모니터링
  page.on('response', response => {
    console.log(`${response.status()} ${response.url()}`);
  });
});
```

7. 테스트 실행 및 모니터링

7.3 결과 분석 및 리포팅

```
// report-generator.ts
const generateReport = async (results) => {
  const report = {
    summary: {
      total: results.length,
      passed: results.filter(r => r.status === 'passed').length,
      failed: results.filter(r => r.status === 'failed').length
    },
    performance: {
      avgResponseTime: calculateAverage(results.map(r => r.duration)),
      p95ResponseTime: calculatePercentile(results.map(r => r.duration), 95)
    }
  };

  await writeFile('report.json', JSON.stringify(report, null, 2));
};
```

8. CI/CD 파이프라인 통합

8.1 GitHub Actions 연동 (1/2)

```yaml
# .github/workflows/test.yml
name: E2E and Load Tests
on: [push]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Setup Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '16'
```

8. CI/CD 파이프라인 통합

8.1 GitHub Actions 연동 (2/2)

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

      - name: Install Playwright browsers
        run: npx playwright install --with-deps

      - name: Run E2E tests
        run: npx playwright test

      - name: Run Load tests
        run: npx artillery run load-test.yml
```

8. CI/CD 파이프라인 통합

8.2 Jenkins 연동 (1/2)

```
// Jenkinsfile
pipeline {
    agent any

    stages {
        stage('Setup') {
            steps {
                sh 'npm ci'
                sh 'npx playwright install --with-deps'
            }
        }
        stage('E2E Tests') {
            steps {
                sh 'npx playwright test'
            }
        }
```

8. CI/CD 파이프라인 통합

8.2 Jenkins 연동 (2/2)

```
        stage('Load Tests') {
            steps {
                sh 'npx artillery run load-test.yml'
            }
        }
        stage('Report') {
            steps {
                publishHTML([
                    allowMissing: false,
                    alwaysLinkToLastBuild: true,
                    keepAll: true,
                    reportDir: 'playwright-report',
                    reportFiles: 'index.html',
                    reportName: 'E2E Test Report'
                ])
            }
        }
    }
}
```

9. Best Practices

9.1 안정적인 테스트 작성법

- **명확한 선택자 사용**
  - ID나 테스트용 속성 사용
  - CSS 클래스나 XPath 의존성 최소화
- **적절한 대기 전략**
  - `waitForSelector` 활용
  - 명시적 타임아웃 설정
  - 네트워크 요청 완료 대기
- **테스트 격리**
  - 독립적인 테스트 환경
  - 상태 리셋
  - 테스트 간 의존성 제거

9. Best Practices

9.2 성능 최적화 전략

- **리소스 효율적 사용**
  - 브라우저 재사용
  - 병렬 실행 최적화
  - 메모리 누수 방지
- **부하 테스트 설계**
  - 점진적 부하 증가
  - 현실적인 시나리오
  - 적절한 모니터링

9. Best Practices

9.3 문제 해결 가이드

- **일반적인 문제 해결**
  - 스크린샷 및 비디오 활용
  - 상세 로깅 설정
  - 디버거 활용
- **성능 문제 해결**
  - 병목 지점 식별
  - 리소스 사용량 모니터링
  - 최적화 포인트 도출

감사합니다!

Q&A

질문이 있으시면 자유롭게 해주세요.