靠北 ... 哪裡寫錯

Test -> CI/CD -> DevOps

小明正在做一個網頁，想要把他的成績單秀出來。
他寫了一個成績總和的 function，可是他的程式有錯，
問題是 VSCode 沒有跳錯誤語法上沒有問題，
你能幫他找出哪裡寫錯了？(有三個地方)

```
1   function calcMyScore () {
2     let data = [100, 99, '87', 70]
3     let score = 0
4     for (let i = 1; i < data.length; i++) {
5       score += data[i]
6     }
7
8     return score
9   }
10
11  console.log(calcMyScore)
12
```

罐罐 @JasonXDDD

劉冠林 (罐罐 XD)

麗臺科技 Leadtek：RD

MyProGuide：Web Developer

2015 從事網站開發工作

GitHub:
https://github.com/JasonXDDD
Website:
https://jasonxddd.web.app/

Len_AshBell
JasonXDDD

Edit profile

12 followers · 6 following · ☆ 2

Taiwan
jason1022liu@gmail.com
jasonxddd.web.app

Highlights

Arctic Code Vault Contributor
PRO

Organizations

Longest streak
17 days
Jan 4 — Jan 20

Current streak
1 days
Apr 1 — Apr 1

Show normal chart ▾

今天講什麼？

- 怎麼這裡壞、那裡亂？你需要**測試 (test)**
- 職場上，團隊是怎麼開發的？什麼是 **持續部署 (CI/CD)**？
- **DevOps** 工具推薦
- [實戰] 測試與部署

你答對了嗎？

1. 型別錯誤
2. Array index 錯誤
3. Call Function 錯誤

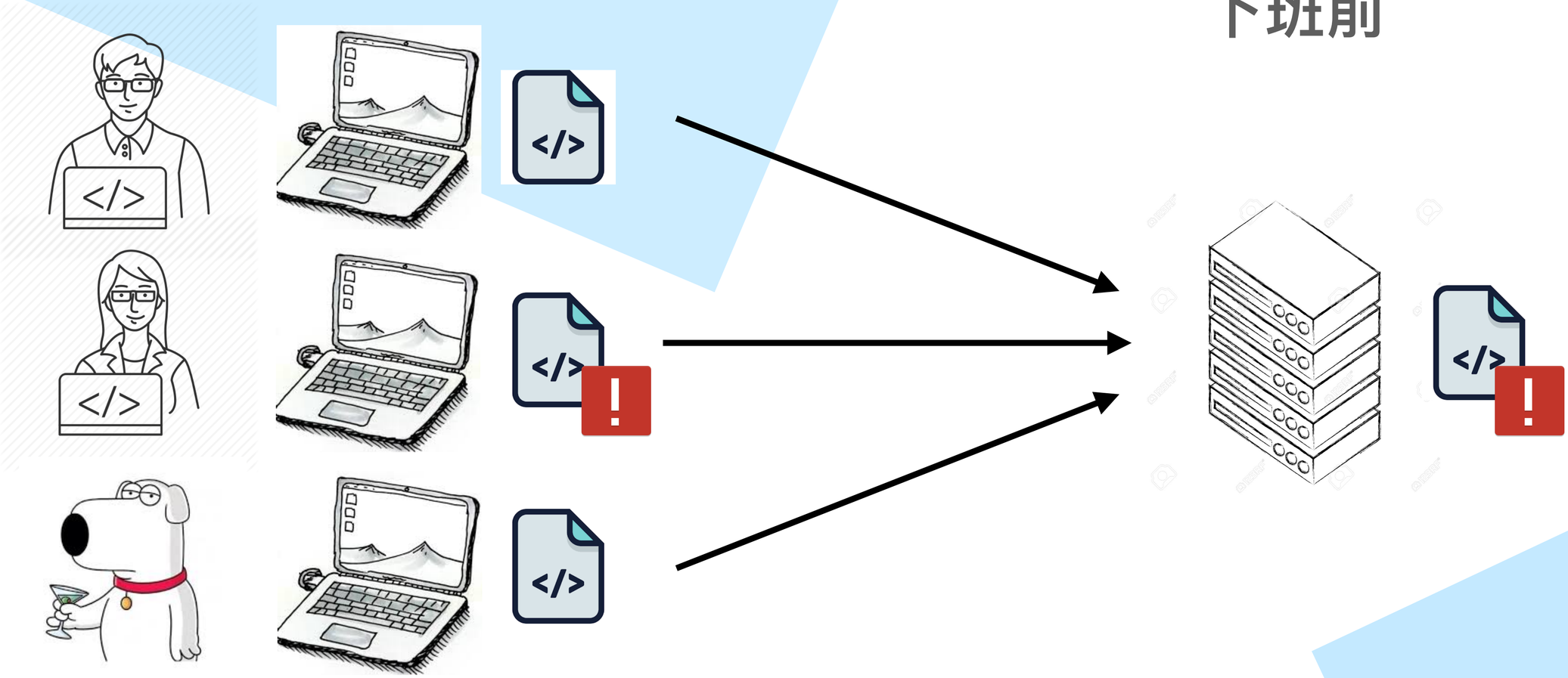
```
function calcMyScore () {
  let data = [100, 99, '87', 70]
  let score = 0
  for (let i = 1; i < data.length; i++) {
    score += data[i]
  }

  return score
}

console.log(calcMyScore)
```

關鍵字懶人包

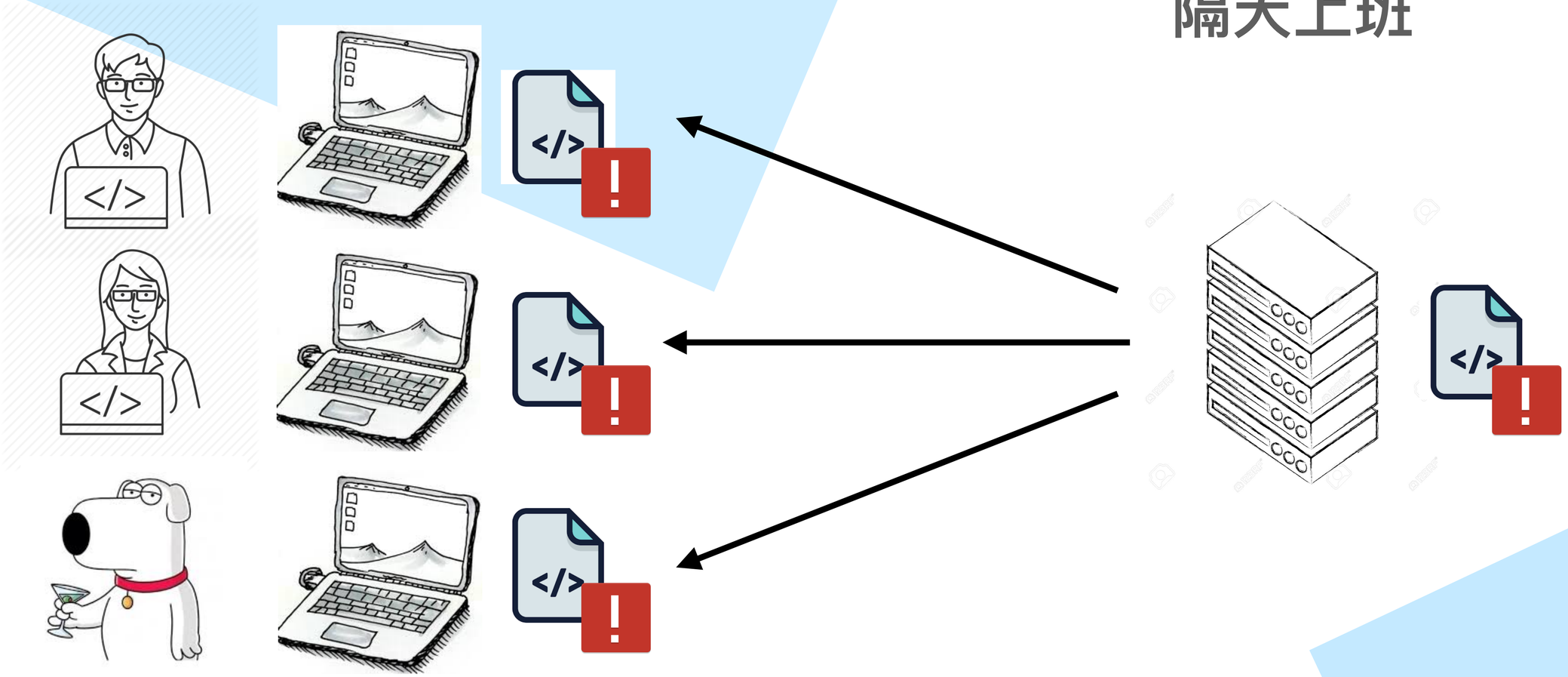
1. JS Test Tool: **Jest**
2. GitHub Deploy Tool: **Travis CI**
3. Website Hosting: **GitHub Page**

誰是 Bug 製造機？

下班前



誰是 Bug 製造機？



一個專案大家同時開發，一個爆炸全部人都爆炸

工程師日常

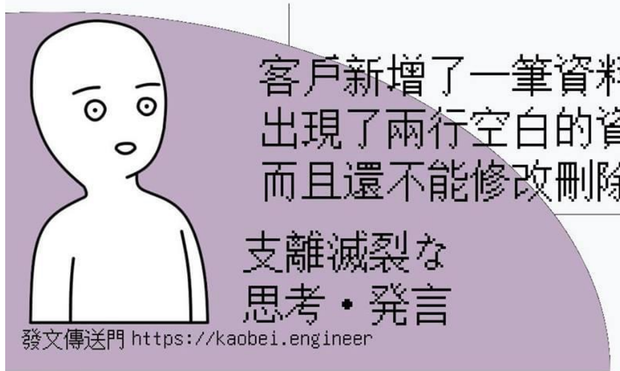
測試工程師新增了資料
一切正常

測試工程師當了時空旅行者新增了未來的資料
一切正常

測試工程師回到了80年代新增資料
一切正常

測試工程師新增了一片白什麼狡都沒有的資料
一切正常

測試工程師試著學user的腦袋做了一堆非常規操作都沒有把系統玩到掛掉
於是測試工程師回報客戶沒有異常



2020-11-16 新增 “資料” OK!

2022-12-23 新增 “資料” OK!

1990-01-23 新增 “資料” OK!

2020-11-16 新增 ”” OK!

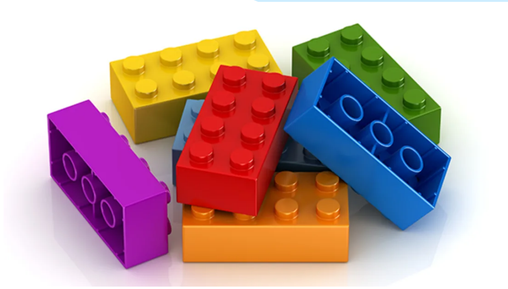
1234-01-23 新增 ”!@#^” OK!

客戶使用
Booommm!

我們需要好好的管理一下

測試，到底在幹嘛？

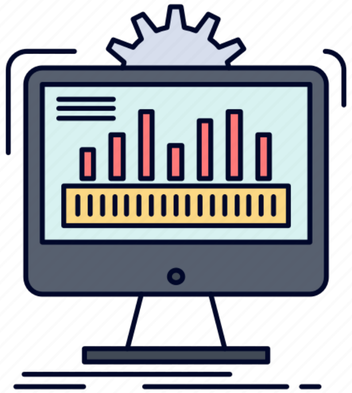
- 透過某些**機制或流程**，保證你的東西是能**動的 (注意不一定是正確的)**



測試你寫的 Function 是對的

測試功能模組正常

測試整體操作上沒問題



測試系統穩定

測試系統安全

測試，到底在幹嘛？

- 透過某些**機制或流程**，保證你的東西是能**動的 (注意不一定是正確的)**

單元測試
Unit Test

測試你寫的 Function 是對的

整合測試
Integration Test

測試功能模組正常

端對端測試
E2E Test

測試整體操作上沒問題

壓力測試

測試系統穩定

漏洞測試

測試系統安全

單元測試：最小單位的測試

- 一個資訊系統最小單位就是 Function, Variable
  也就是針對你寫的 Code 來測試是否正確。
- 數量多、範圍小 = 你要花很多時間來寫

```
const student = require('../js/student')

test('字串比對：Hi, 王小明!', () => {
  const name = '王小明'

  // 期望 Function 回復對的字串
  expect(student.sayHi(name)).toBe('Hi, 王小明!')
})
```

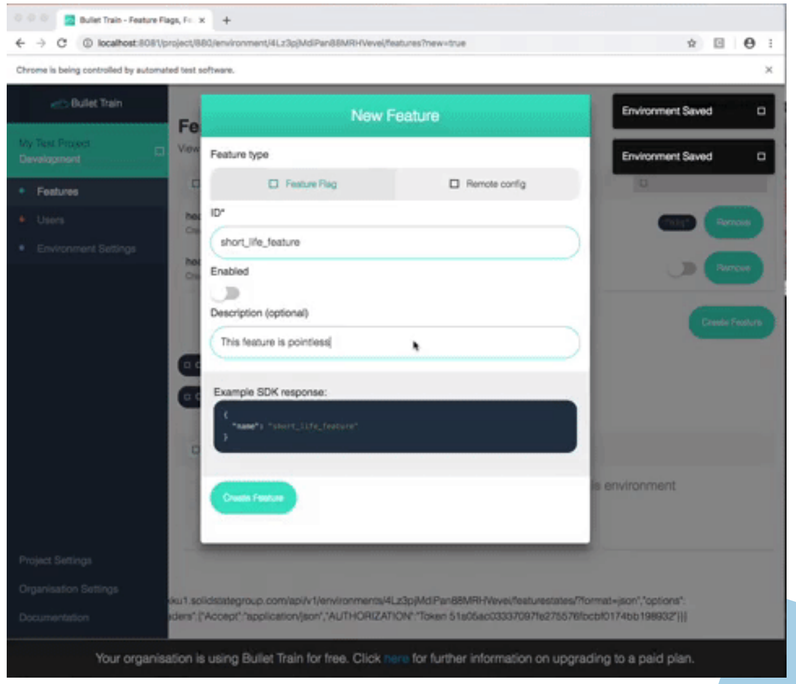
整合測試：A是對的、B是對的、A+B 就錯了

- 把功能兜起來，檢查他是不是正確的，重點在於流程上的檢查
  開發上最容易被忽略的一步
- 需要事先規劃好如何測試、以及測試內容

E2E測試：難用就是錯

- 指的是端點 (end-point) 到端點的測試，以我們來說 E(使用者) - E(系統)
  把網站更新上去之後自己按按看就是在做 E2E。

- 重點在於易用性、操作上的複雜度，也是最重要的測試



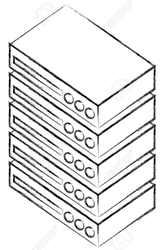
什麼時候該測試？

- 只要有 code 在跑的地方都要測試

開發環境 → 測試環境 → 正式環境

單元測試

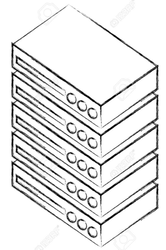
自己寫的 Code 自己維護



單元測試
整合測試
E2E 測試

大家東西結合在一起
看看會不會發生問題

整合測試
E2E 測試
壓力測試
安全測試
...



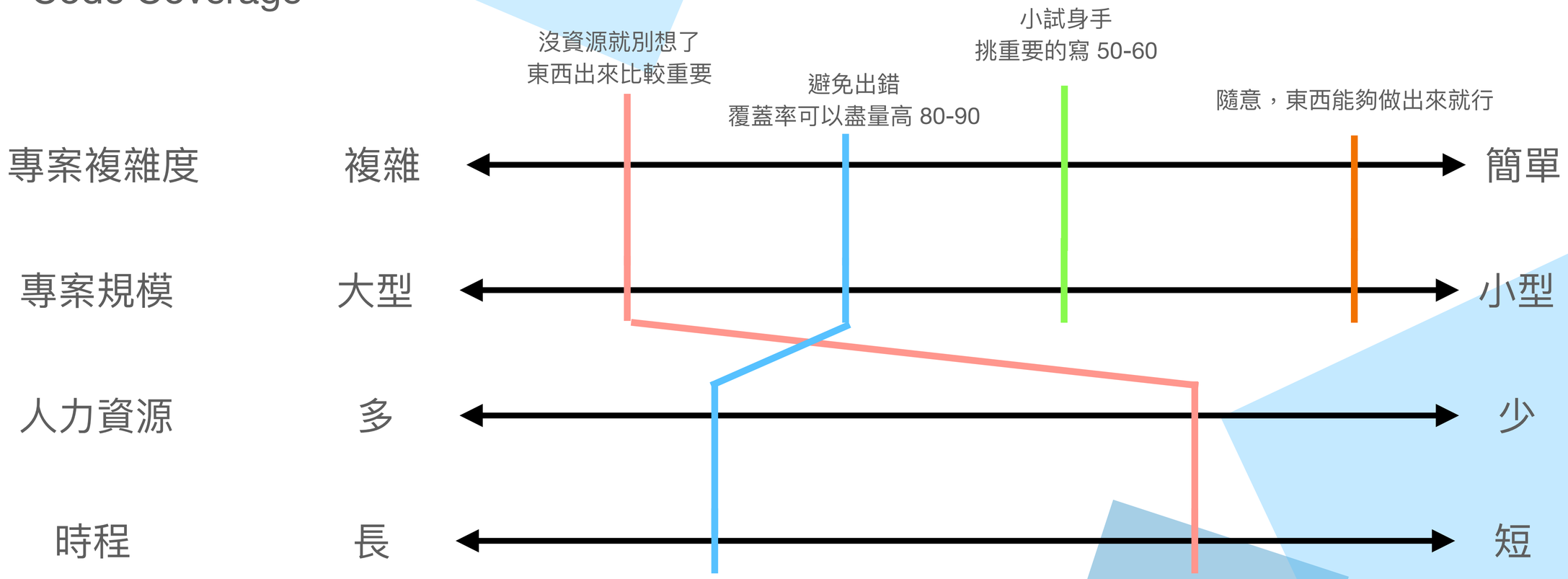
產品要上線了
不能出任何錯誤！

我要全部都寫測試嗎？

- 不，要看專案 (規模、複雜度) 與資源 (人力、時間) 而定。

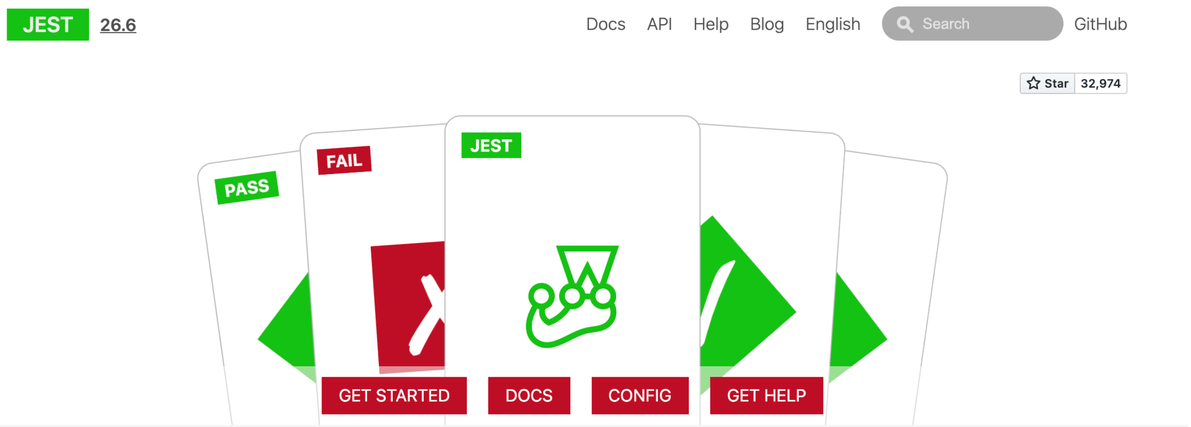
測試覆蓋率 = 測試內容 / 全部項目

Code Coverage



案例：Jest - unit test for JavaScript

- 簡潔明瞭、快速的 JavaScript 測試工具
- 可以支援各大現代框架
- 自動檢查專案的測試檔案、並提供覆蓋率數據

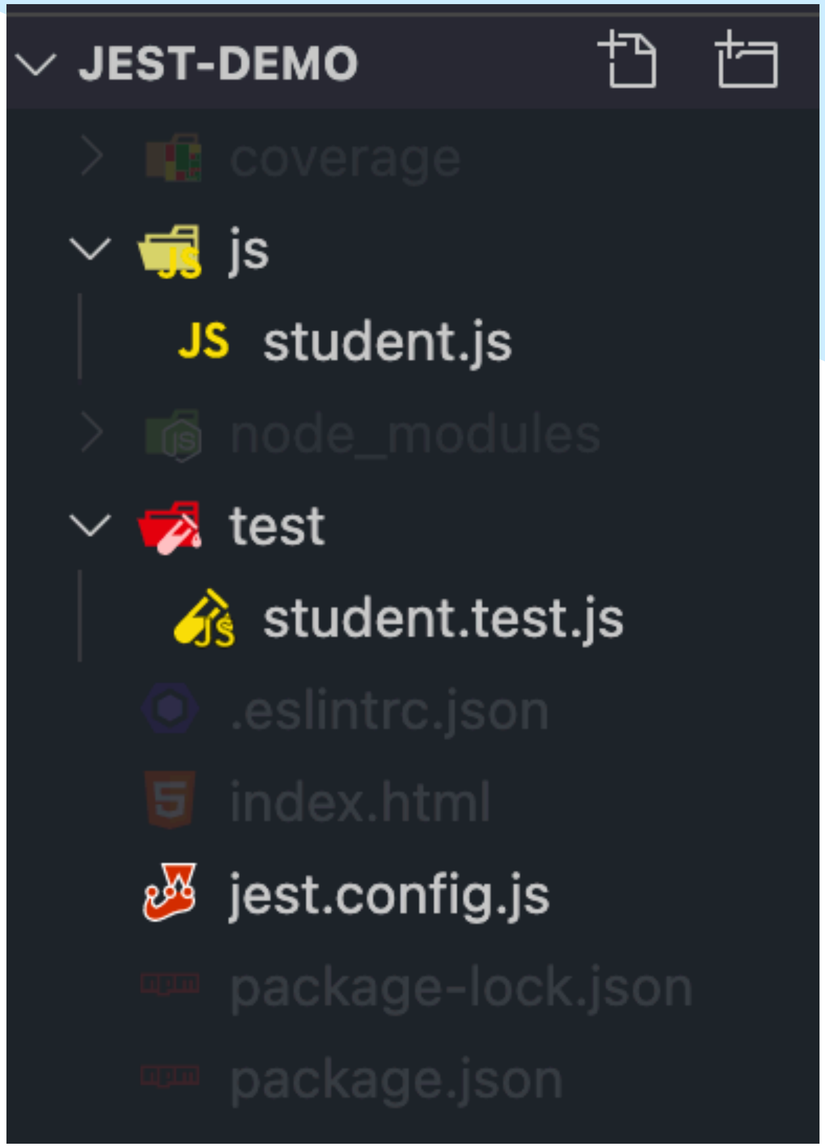



```
 npm run test

> @ test /Users/jasonxd/Documents/CICD-Demo/jest-demo
> jest --coverage

(node:52456) ExperimentalWarning: The fs.promises API is experimental
PASS test/student.test.js (13.054 s)
  ✓ 字串比對：Hi, 王小明！(6 ms)
  ✓ 數字：數學成績大於60 (1 ms)
  ✓ 物件：上學期各科成績都是100分 (3 ms)
  ✓ 正規表達式：符合 email 格式 (1 ms)
  ✓ 陣列包含：3年1班學生包含王小明在內 (1 ms)
  小明上課流程測試
    ✓ 數字：早上10點，進入教室 (1 ms)
    ✓ 陣列包含：檢查手機有沒有在書包裡 (1 ms)
    ✓ 物件：輸入遊戲帳密，並驗證
    ✓ 字串比對：下課，謝謝老師！ (1 ms)

------------|---------|----------|---------|---------|-------------------
File        | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
------------|---------|----------|---------|---------|-------------------
All files   |     100 |      100 |     100 |     100 |
 student.js |     100 |      100 |     100 |     100 |
------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        27.237 s
Ran all test suites.
```

案例：Jest - unit test for JavaScript

- 進到專案
- npm install --save-dev jest
- 為 XXX.js 新增，XXX.test.js
- 寫 test case

案例：Jest - unit test for JavaScript




```
3   ✔test('字串比對：Hi, 王小明!', () ⇒ {
4       const name = '王小明'
5
6       // 期望 Function 回復對的字串
7       expect(student.sayHi(name)).toBe('Hi, 王小明!')
8   })
```

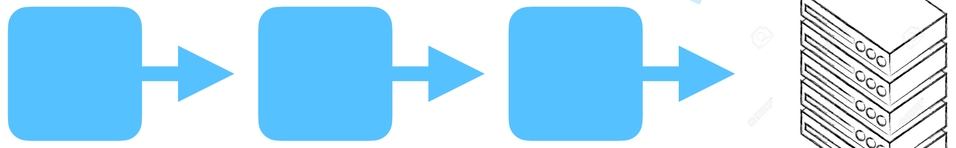
如果能夠與部署流程整合最好！

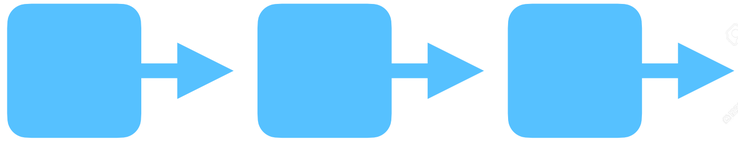
除了測試之外，事實上我們還要做許多事

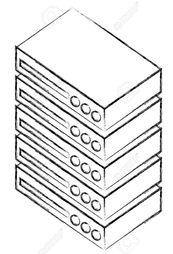
部署流程：把程式碼更新至機器上執行一系列動作的過程

Deploy

開發環境 → 測試環境 → 正式環境







編譯 → **測試** → **部署**

將程式碼進行封裝

執行測試動作

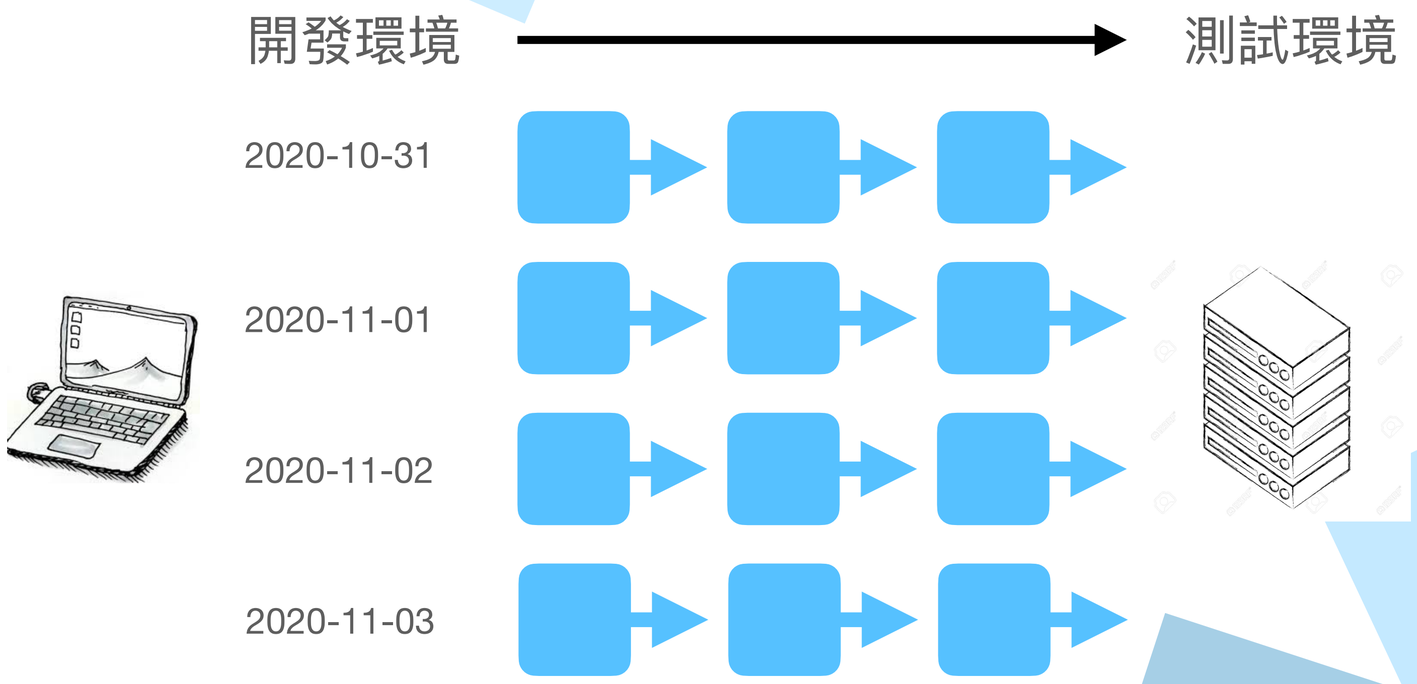
把打包好的程式碼
推到機器上執行

每次更新都持續跑部署流程

持續整合/部署：透過更新觸發持續執行部屬流程

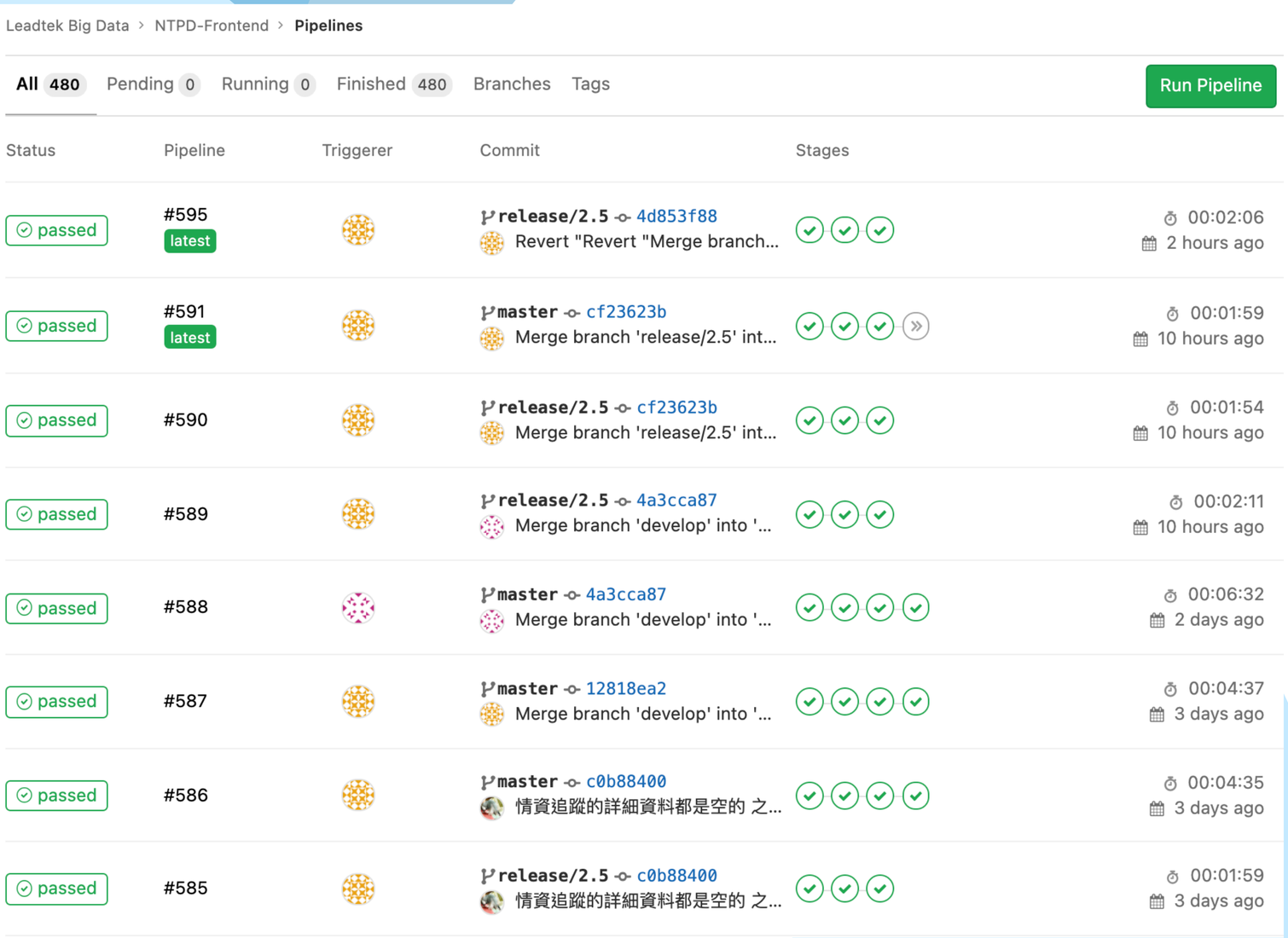
Continue Integration / Deploy (CI/CD)

- CI/CD 主要精神在自動化整合、部署流程，能夠將這些麻煩的過程自動化。



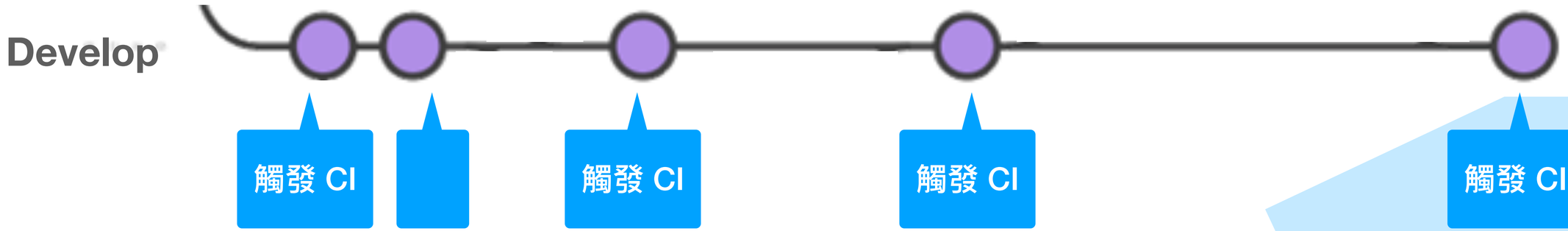
CI 帶來的好處

- 自動化測試與部署流程
- 能夠快速至不同環境
- 在複雜的系統上避免出錯
- 緊密的結合版本控制
- …

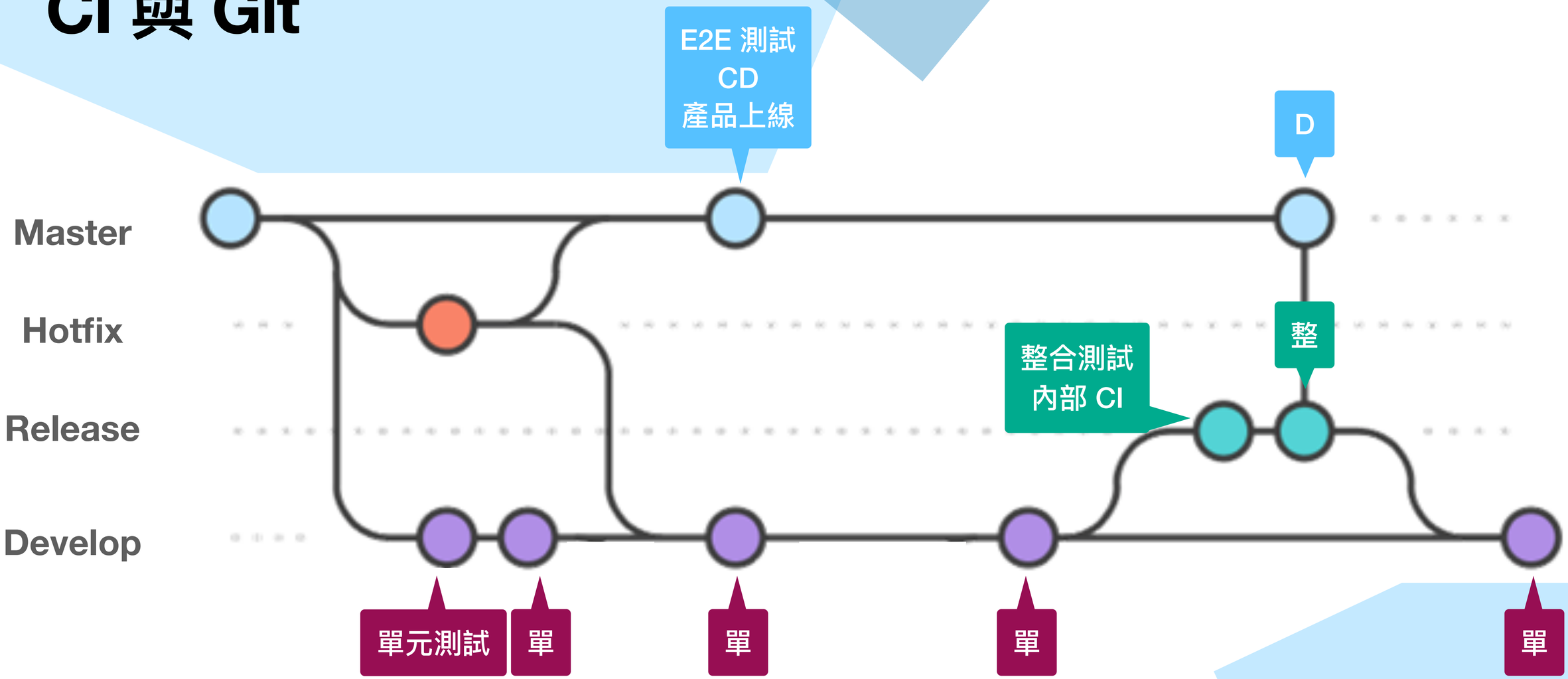


CI 與 Git

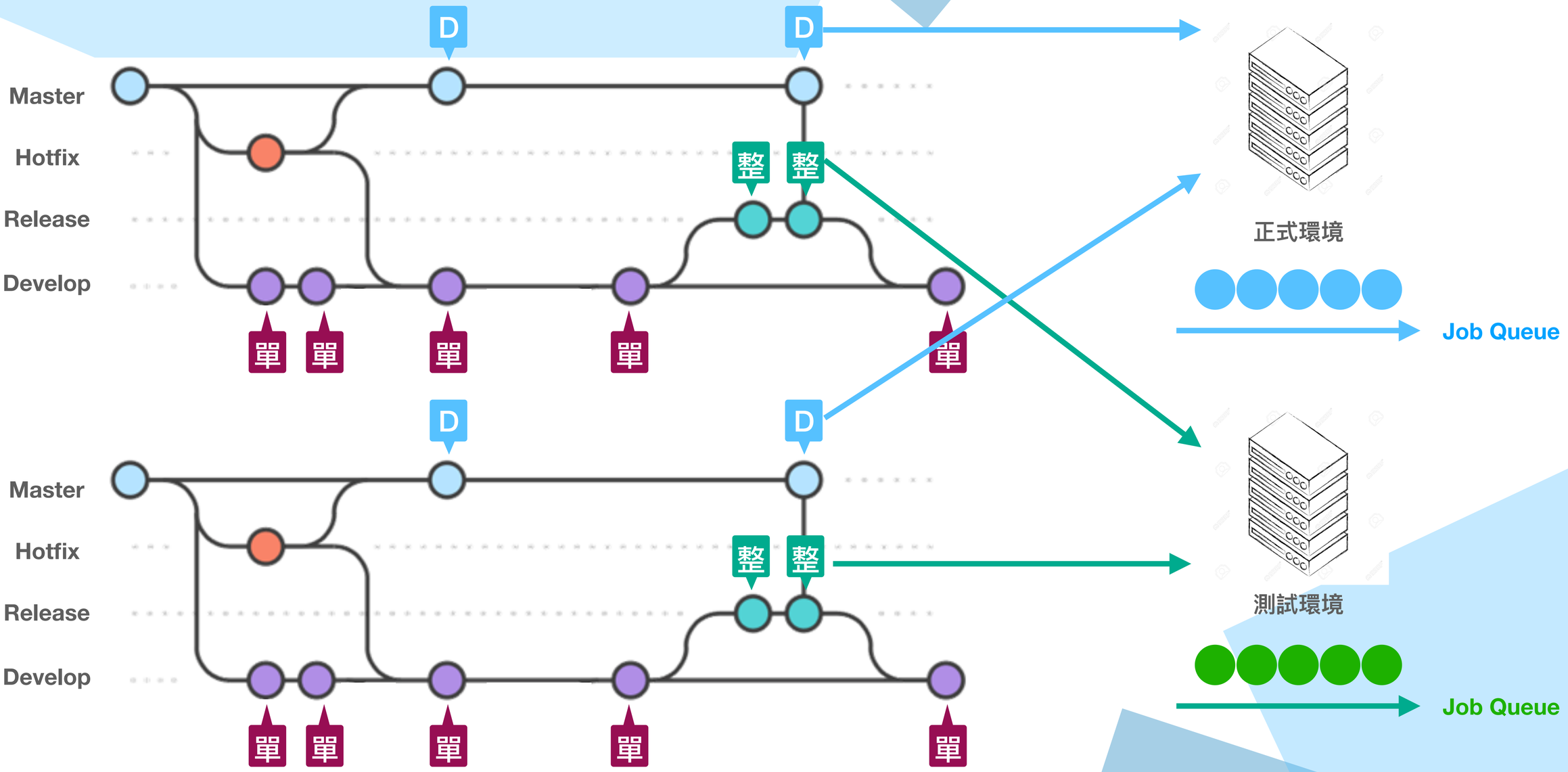
- 版本控制：程式碼的時光機，紀錄所有更改的過程。
- 萬事皆由版控做起，維護版控上的分支與紀錄是非常重要的事情。
- CI/CD 能夠自動化，是根據 Git 上的紀錄來觸發



CI 與 Git

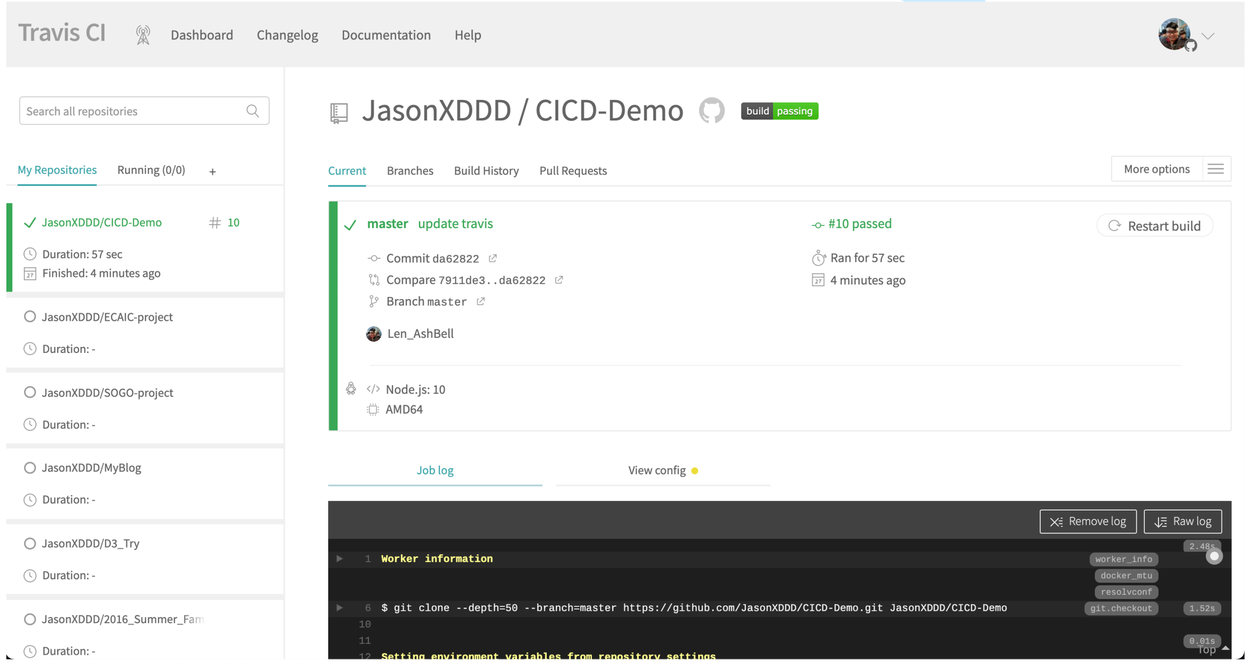

E2E 測試
CD
產品上線
D
Master
Hotfix
整
整合測試
內部 CI
Release
Develop
單元測試
單
單
單
單

CI 與 Git


D
D
Master
Hotfix
Release
Develop
整
整
單
單
單
單
單
正式環境
Job Queue
D
D
Master
Hotfix
Release
Develop
整
整
單
單
單
單
單
測試環境
Job Queue

案例：Travis CI - CI for GitHub

- 專為 GitHub 而做的 CI，支援 GitHub 功能
- 簡易快速上手
- 建立流程容易

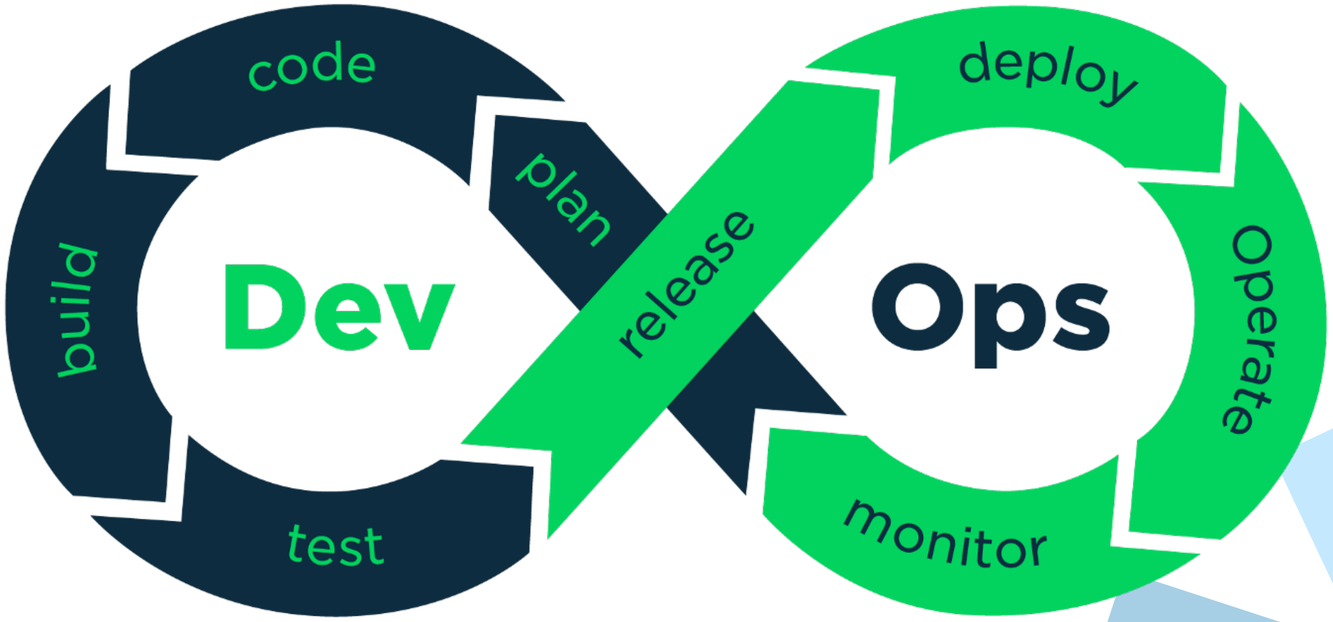


Travis CI

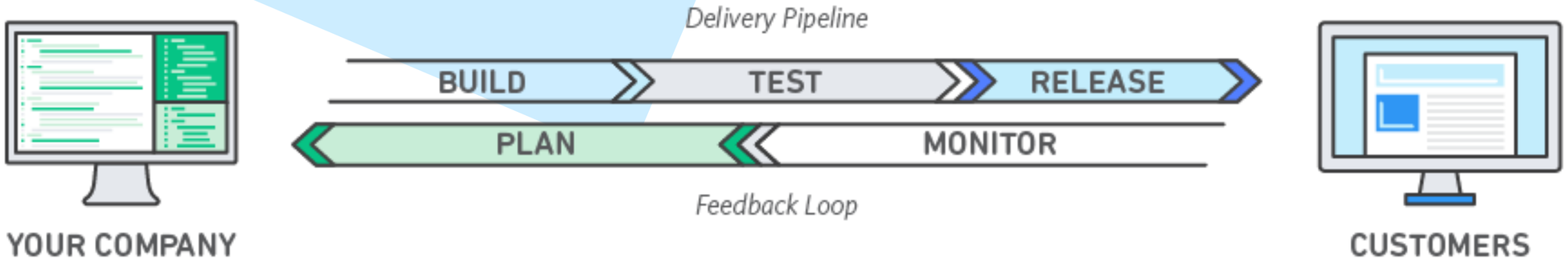
一切就緒，動起來吧！

與各團隊溝通

- 專案會有好幾個團隊共同完成，彼此負責各種任務：RD, QA, OP ...
- 專案會一直長大，需求也會變更，團隊間的溝通往往是最麻煩的
- 避免產生過多的溝通成本，說話必須有個共同語言與工具

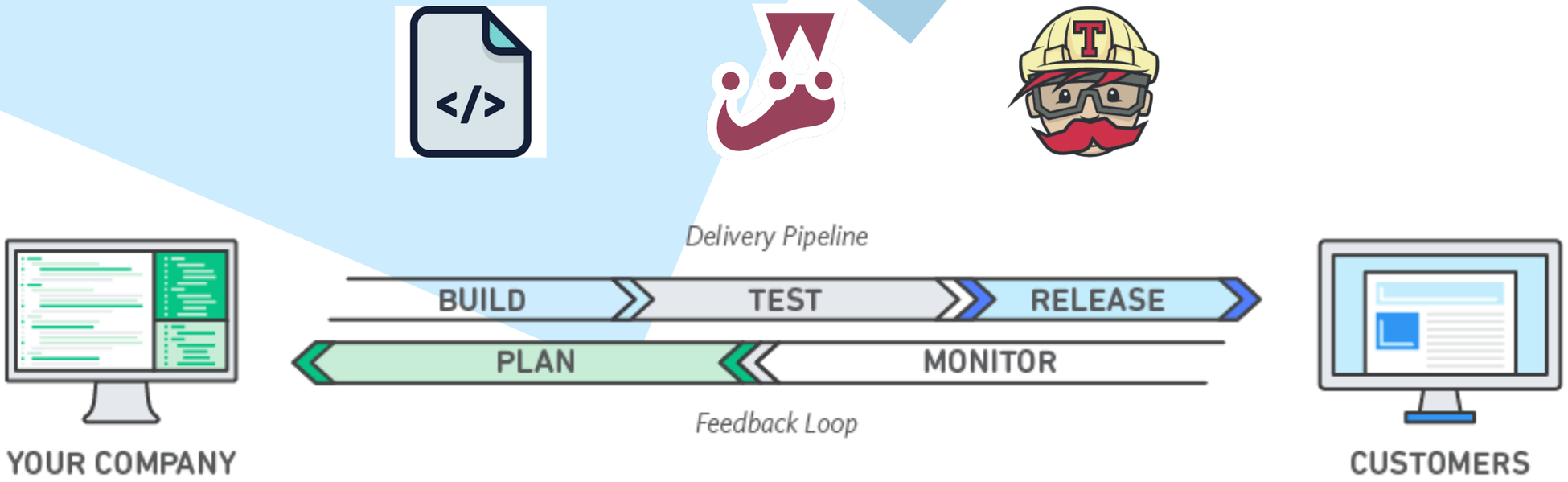


與各團隊溝通



- Dev：把功能更新到專案上
- Ops：反映顧客的狀況並規劃將來的產品方向

與各團隊溝通



Trello

SENTRY

Jira Software

Prometheus

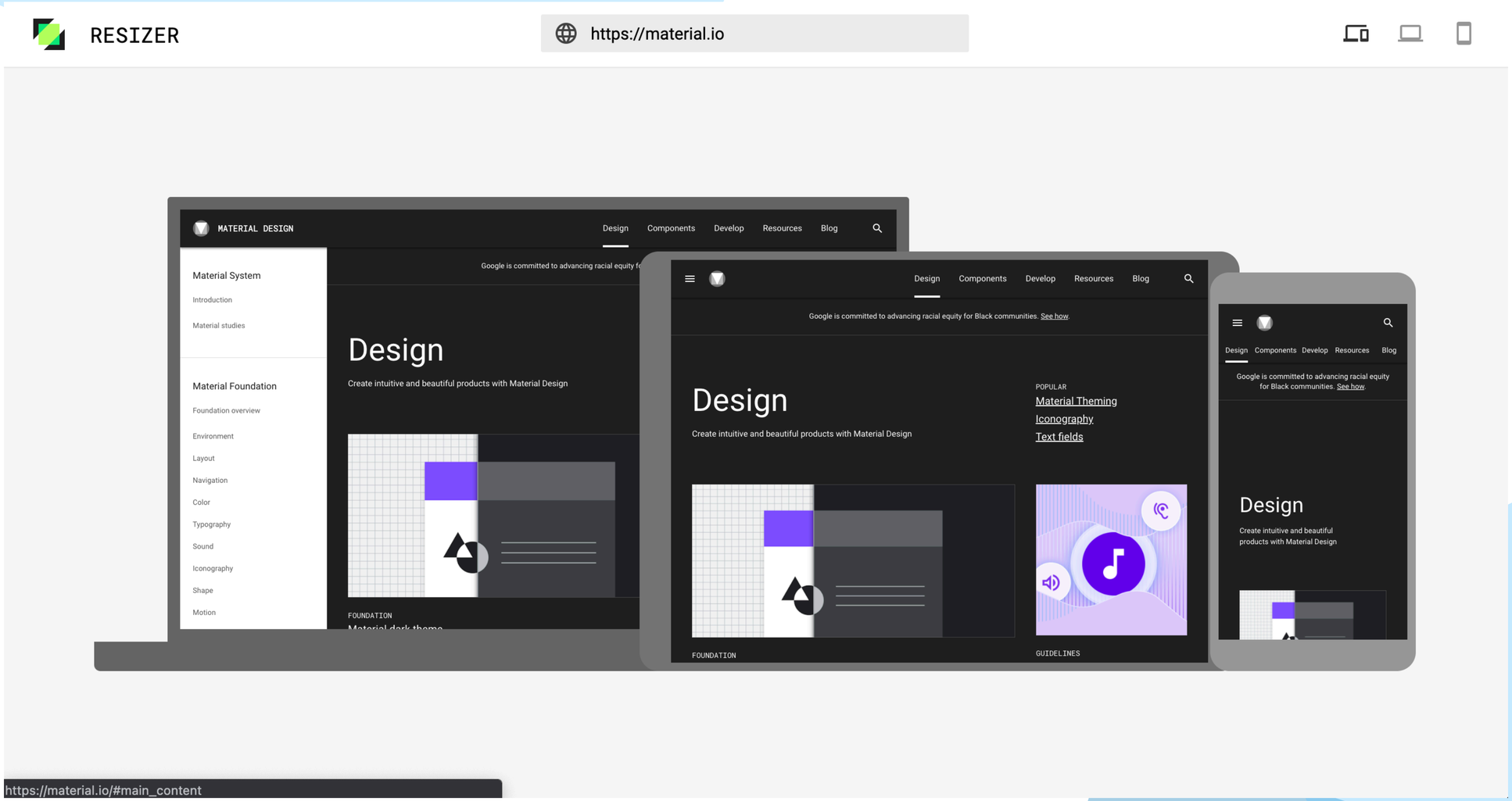
slack

Grafana Labs

最後分享個好用的前端測試工具

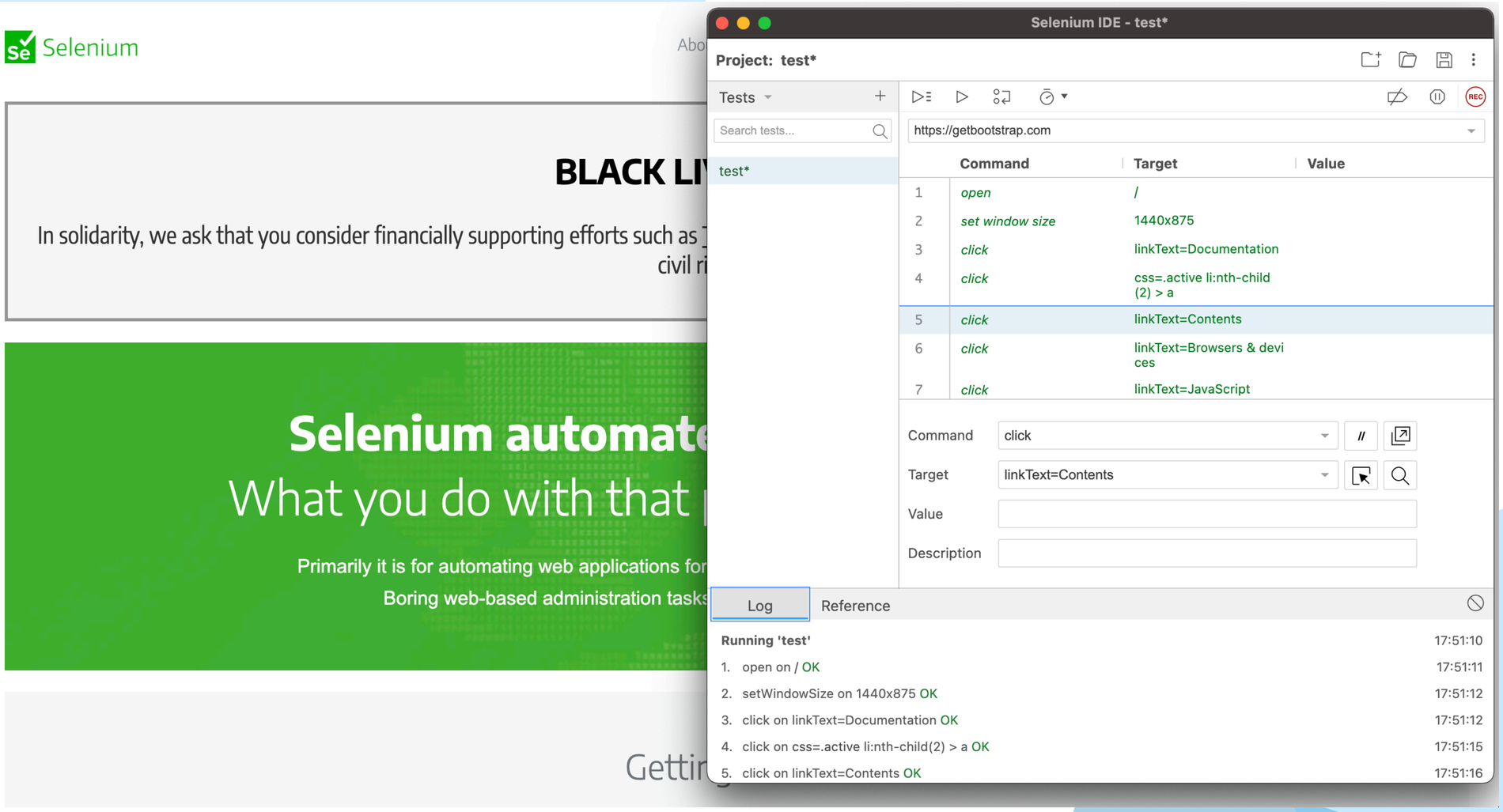
Material Design Resizer

https://material.io/resources/resizer/



Selenium IDE

https://www.selenium.dev/



Google LightHourse

https://developers.google.com/web/tools/lighthouse

Progressive Web App

Performance

Accessibility

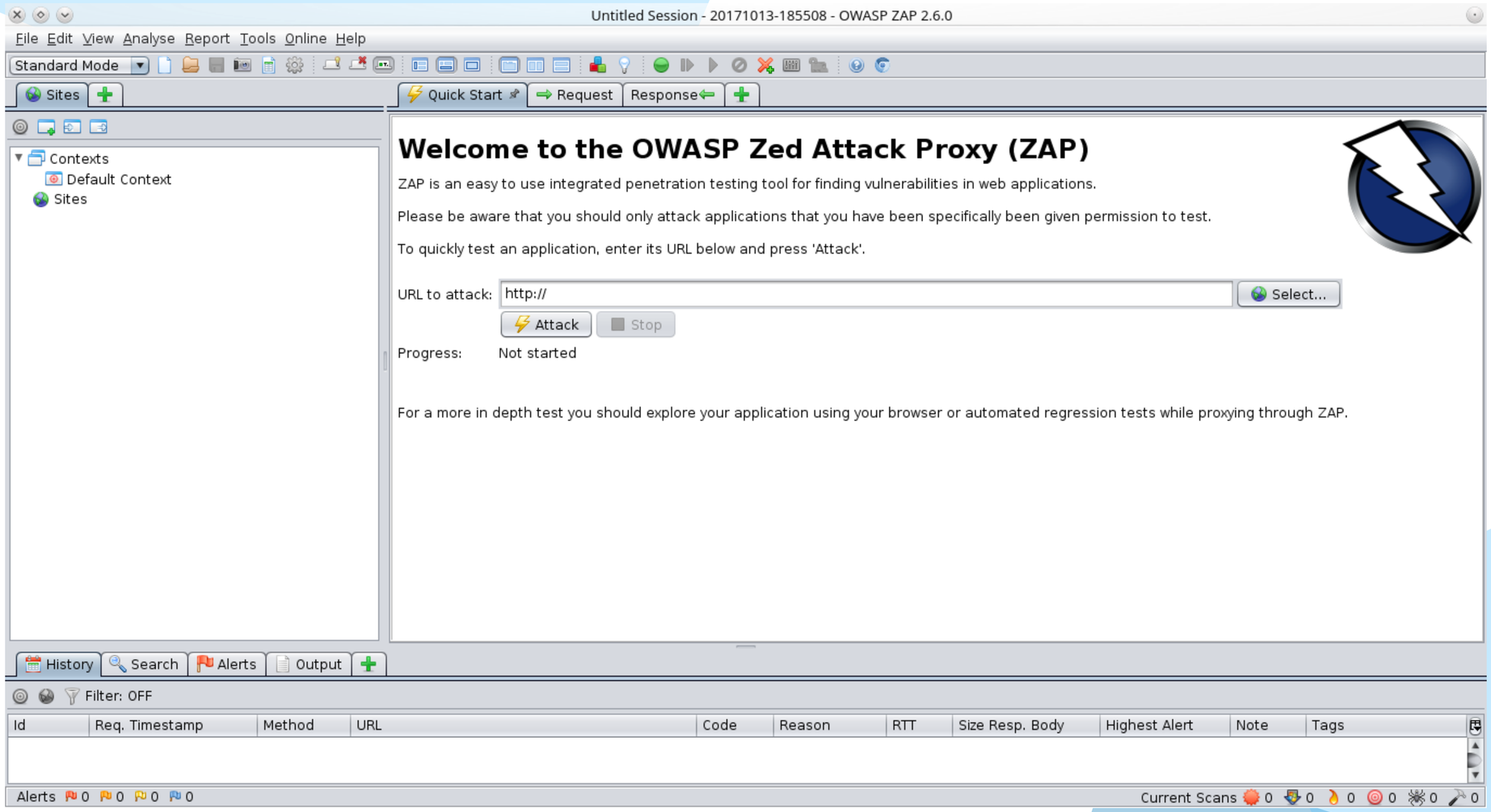
Best Practices

Progressive Web App

These audits validate the aspects of a Progressive Web App, as specified by the baseline PWA Checklist.

OWASP ZAP

https://www.zaproxy.org/



謝謝大家