Jenkins CI

敏捷開發團隊之 Jenkins CI實踐 實戰手冊

docker

鋼鐵人創造了 Jarvis 為他打理大小事，讓他的主人可以專心在創意的發想以及對付強大的敵人；在軟體開發同樣的我們也可以透過機器人管家 Jenkins CI，來為我們打理開發流程中的大小事，讓我們專注在最有價值的工作上。

本講義將從豐田式生產（Lean Production）切入，引導學習持續整合的目的，並介紹從各個階段如何搭配 Jenkins CI 持續整合。另外也將一步步帶領使用 Jenkins CI 進行自動化建置、測試與部署的實作，透過此課程將領引大家打造一個專案的管家，關注開發流程的每個一環節，一步一步帶著大家完成透過 Jenkins 完成 CI / CD 流程。

透過 Jenkins CI 的操作，傳授在敏捷開發流程中如何無痛進行 CI 的導入，並且搭配 Jenkins CI 實務上如何在專案的每一環節協助團隊運作。

創科資訊團隊 編著

Table of Contents

1. Introduction
2. 上課須知
    i. 相關服務事前註冊
    ii. 事前安裝：自備筆電
    iii. 事前安裝：電腦教室
3. TDD(測試驅動開發方式)
    i. Java
        i. JUnit 基礎
        ii. JUnit 與 Maven
        iii. Assertions
        iv. Test Fixtures
        v. 略過 test
        vi. Junit 測試 API
        vii. JUnit Test Suite
4. Jenkins
    i. Introduction
    ii. Quick Start
        i. Lab 101
        ii. Lab 102
        iii. Lab 103
        iv. Lab 104
    iii. 基本概念介紹
        i. 豐田式生產
        ii. 敏捷式開發
        iii. 持續整合
        iv. 安裝 Jenkins
    iv. 設定
        i. 環境變數
        ii. ssh
        iii. security
        iv. master/slave
    v. Jenkins CI 常用功能
        i. build archive
        ii. 執行 Shell Script

iii. Jenkins API 使用簡介
iv. Plugin 安裝
v. JUnit 報表
vi. Script Console
vi. 常用 plugin 介紹
i. Config File Provider
ii. Cobertura
iii. 使用指令安裝常用套件
vii. CI flow 簡介
viii. Task 實作以 Java + git 為例
i. build
ii. test
iii. preview
iv. release
ix. Task 實作進階
i. cron jobs test
ii. if test ok then preview
x. 搭配 Jenkins 2.0 之 Pipeline 進行建置
i. 基礎練習
ii. 使用 Pipeline 進行建置
iii. 使用 Multibranch Pipeline 進行建置
xi. 自動化檢查與報表設置
i. java doc 使用與報表產出設置
ii. Checkstyle 使用與報表產出設置
iii. PMD(source code analyzer) 使用與報表產出設置
iv. CPD(Copy/Paste Detector) 使用與報表產出設置
v. Simian 使用與報表產出設置
vi. FindBugs 使用與報表產出設置
xii. 建置工具
i. maven
i. Multi Module Project
xiii. Q & A
i. 一般問題
ii. JAVA
5. Appendix
i. vt-x amd-v 異常造成 VM 無法開啟
ii. Ubuntu install docker
iii. Mac OS X install docker

iv. docker 參考資料
v. git 參考資料
vi. docker 相關練習解答

AgileWorks

本教材由創科資訊研發團隊謝宗穎、林彥宏等共同編撰。

敏捷軟體開發培訓課程，預約請洽創科資訊。

2016 年企業培訓課程包含：

- Docker 從基礎到進階實戰班
- Jenkins 從基礎到進階實戰班
- 持續整合與自動化測試實作（含 Jenkins 與 Docker）

持續整合與自動化測試實作班

適合對象

企業內訓 / 從基礎到進階

學習目的

透過 Learning by Doing 從做中學的方式，瞭解持續整合與自動化測試的方法論與專案實作方式，包括 Jenkins 與 Docker 的工具實務應用，幫助學員能在日常工作中實際應用。

- 理解敏捷開發中持續整合的價值與目的
- 學習如何搭配 Jenkins 完成專案自動化建置，包含單元測試及整合測試
- 學習如何透過 Docker 配置開發與部署的軟體環境，並應用在持續整合的流程中
- 以常用的開發工具（Java / Node.js 等）提供自動化專案建置與測試的範例

課程大綱

面對快速變動的系統開發需求，想要維持軟體的高品質要求，持續整合是重要的關鍵。Jenkins 是實現 CI 的工具首選，而 Docker 則是目前最受歡迎的輕量級虛擬化技術，透過這兩種工具與現有各種原始碼版本管理方案的整合，讓企業更容易實現持續整合流程中，關於自動化軟體建置及測試，以及快速驗證軟體版本演進的需求。

鋼鐵人創造了 Jarvis 為他打理大小事，讓他的主人可以專心在創意的發想以及對付強大的

敵人。在軟體開發同樣的我們也可以透過機器人管家 Jenkins，來為我們打理開發流程中的大小事，讓我們專注在最有價值的工作上。本課程將從豐田式生產（Lean Production）切入，引導學習持續整合的目的，並介紹軟體開發流程的各個階段如何搭配 CI 運作。另外也將一步步帶領學員，使用 Jenkins 進行自動化建置、測試與部署的實作，以及搭配 Docker 容器進行測試與部署，透過此課程將領引大家打造一個軟體專案的管家，關注開發流程的每個一環節。

課程特色

採用 Workshop 形式授課，每個單元提供實作練習。 強調每位學員皆能動手完成實作練習，搭配 AgileWorks 教學平台提供線上教材與 VM Lab 操作環境。 基礎課程包含 TDD 觀念、專案自動化建置實務、Jenkins 基本功能講解及常用 Jenkins Plugins 功能說明等。 進階課程包含 Jenkins Cluster 建置、專案自動化測試實務及搭配 Docker 容器的測試部署整合等。 可依照開發團隊需求調整搭配的軟體工具（例如 Git、SVN；Java、PHP、Node.js 等）

備註

為確保課程品質及學員滿意度，Workshop 總人數建議以 30 人為限。

時數

建議安排 12 小時（分成 2 次上課，每次 6 小時，例如：上午 2 小時、下午 4 小時）

課程實績

iThome、資策會、關貿網路、台灣大哥大、中華電信學院、…等。

授課講師

謝宗穎 / 創科資訊共同創辦人

> 實務上的 Web Full Stack 開發經驗，熟悉 Java 以及 JavaScript 開發技術，對開發流程優化有高度興趣，提倡 TDD 開發方式，以及透過 CI Jenkins 進行自動化，利用 Docker 輔助開發或測試環境建置，技術團隊管理者。
>
> 培訓經驗：曾於 JSDC 2015；JCConf Taiwan 2015；Container Summit 2014 / 2015 等技術大會擔任講者，並在 QNAP、關貿網路、資策會與 iThome 等機構擔

> 任技術相關演講及工作坊講師。專題演講經驗有台灣大學、台中科大、正修科大與義守大學等校。

林彥宏 / 創科資訊共同創辦人

> 參與軟體開發專案經驗逾 10 年，經歷 PHP、.NET、Java 與 JavaScript / Node.js 等多種開發平台及框架，對 Windows Server及 Linux 伺服器與 AWS 雲端部署皆有多年的經驗，亦熟悉 SQL Server、MySQL 等資料庫軟體。
>
> 培訓經驗：協同教學業師：修平科大；靜宜大學；嘉南藥理科大；成功大學；高雄海洋科大；長榮大學；和春技術學院；正修科大；建國科大；聯合大學；南台科大；嶺東科大。曾擔任 Java Developer Day 2014；JCConf Taiwan 2014 / 2015 等技術大會演講者。

持續整合與自動化測試：相關服務事前註冊

註冊 GitHub 帳號

本課程會帶你實機操作，課程中將會需要下載範例專案，以及進行 GitHub 與 Docker Hub 串接運用。

如果你還沒有 GitHub 帳號，請在 GitHub 官方網站進行註冊。

註冊 Docker Hub 帳號

如果你還沒有 Docker Hub 帳號，請到 Docker Hub 網站 進行註冊。

持續整合與自動化測試：自備筆電課前準備

本課程以實機操作實作練習為主，為維持課程品質及確保學習進度，我們假設您已經熟悉：

1. Linux 基本操作指令
2. Git 版本控制工具基本操作指令

請先完成以下軟體安裝：

1. Google Chrome 瀏覽器
2. VirtualBox 5.x 以上版本虛擬機器軟體及 Extension Pack

並確保電腦規格符合以下建議條件：

1. 建議使用 64 位元作業系統
2. 作業系統 Windows、Mac 或 Linux 皆可
3. 建議 8GB 以上實體記憶體（最低需求為 4GB）
4. 至少 40GB 磁碟可用空間

我們在課程中會提供實作練習專用的 VM Image，以方便搭配教材內容。

安裝 VirtualBox

依照您的作業系統類型，安裝 5.x 以上最新版本的 VirtualBox 軟體，建議同時安裝「VirtualBox Extension Pack」以支援虛擬機器的進階功能。

VirtualBox 下載頁面 https://www.virtualbox.org/wiki/Downloads

安裝 VirtualBox Extension Pack 之步驟可以參考這份文件。

常見的 VirtualBox 安裝問題：

- vt-x amd-v 異常造成 VM 無法開啟

註冊 GitHub 帳號

本課程會帶你實機操作，課程中將會需要下載範例專案，以及進行 GitHub 與 Docker Hub 串接運用。

如果你還沒有 GitHub 帳號，請在 GitHub 官方網站進行註冊。

註冊 Docker Hub 帳號

如果你還沒有 Docker Hub 帳號，請到 Docker Hub 網站 進行註冊。

持續整合與自動化測試：電腦教室課前準備

請先完成以下軟體安裝：

1. Google Chrome 瀏覽器
2. VirtualBox 5.x 以上版本虛擬機器軟體及 Extension Pack

並確保電腦規格符合以下建議條件：

1. 建議使用 64 位元作業系統
2. 作業系統 Windows、Mac 或 Linux 皆可
3. 建議 8GB 以上實體記憶體（最低需求為 4GB）
4. 至少 40GB 磁碟可用空間

我們在課程中會提供實作練習專用的 VM Image，以方便搭配教材內容。

安裝 VirtualBox

依照您的作業系統類型，安裝 5.x 以上最新版本的 VirtualBox 軟體，建議同時安裝「VirtualBox Extension Pack」以支援虛擬機器的進階功能。

VirtualBox 下載頁面 https://www.virtualbox.org/wiki/Downloads

安裝 VirtualBox Extension Pack 之步驟可以參考這份文件。

常見的 VirtualBox 安裝問題：

- vt-x amd-v 異常造成 VM 無法開啟

JUnit 4 基礎

JUnit 是元老級的 Java 單元測試框架，大多數的 Java IDE 與建置工具（Build tools）都支援 JUnit，例如下列的軟體對 JUnit 都有原生的支援：

Java IDE

- IntelliJ IDEA
- Eclipse

Build tools

- Maven
- Gradle

DevOps

- Jenkins CI

Java Class

File Name: `Calculator.java`

```
public class Calculator {
  public int evaluate(String expression) {
    //TODO
  }
}
```

Test Case

File Name: `CalculatorTest.java`

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
```

```
public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
}
```

Compile and Run Test

```
# Compile
javac Calculator.java
javac -cp .:junit-4.XX.jar CalculatorTest.java

# Run Test
java -cp .:junit-4.XX.jar:hamcrest-core-1.3.jar org.junit.runner.JUnitCore Calculator
```

Final Java Class

因為有 Test Case，因此在 `evaluate` 方法的撰寫就有更清楚的目標：完成第一版可以通過測試的程式。

File Name: `Calculator.java`

```
public class Calculator {
    public int evaluate(String expression) {
        int sum = 0;

        for (String summand: expression.split("\\+")) {
            sum += Integer.valueOf(summand);
        }

        return sum;
    }
}
```

重新跑一次測試。

參考資料

- JUnit 4 Wiki: Getting Started
- JUnit 4 Wiki 中文翻譯

JUnit 4 與 Maven

若專案是用 Maven 作為建置工具，我們可以對 `pom.xml` 設置 `<dependency>` 來加入 `junit` 的 dependency

設定方式如下。

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

因為 maven 約定大於配置的特性，執行下列指令：

```
mvn test
```

預設將會執行位於 `src/test/java` 的所有 *.java 的 Test Case

參考 Maven 官方 Standard Directory Layout 文件，有其他不同 path 的使用時機。

若要指定特定得 Class 進行測試，我們可以使用

`-Dtest=TestSamle`

來進行，假設我們只要執行單一 Class 完整指令如下

`mvn -Dtest=TestSamle test`

Maven 會去找對應得 Class Name 來進行測試，若有同名的情況下就需要再加上 Package

一旦測試完成，我們可以在 `target/surefire-reports` 找到測試 report 可以使用 Jenkins 進行報表輸出。

Assertion

JUnit 對於所有的 primitive type、object 跟（primitive type 或 object 的）array 提供 overload 的 assertion method。 參數的順序是期望值後頭接著實際值。 第一個參數可以是一個在失敗時輸出的字串。 有一個些微不同的 assertion 是 asserThat，他的參數有一個選擇性的失敗訊息、實際值、以及 Matcher object。 要注意的是，期望值與實際值的順序跟其他 assert method 是顛倒過來的。

下列是每一個 assert method 的典型寫法：

```
package sample;


import static org.hamcrest.CoreMatchers.*;
import static org.junit.Assert.*;

import java.util.Arrays;
import org.junit.Test;

public class TestAssertions {
    @Test
    public void testAssertArrayEquals() {
        byte[] expected = "trial".getBytes();
        byte[] actual = "trial".getBytes();
        assertArrayEquals("failure - byte arrays not same", expected, actual);
    }

    @Test
    public void testAssertEquals() {
        assertEquals("failure - strings are not equal", "text", "text");
    }

    @Test
    public void testAssertFalse() {
        assertFalse("failure - should be false", false);
    }

    @Test
    public void testAssertNotNull() {
        assertNotNull("should not be null", new Object());
    }

    @Test
    public void testAssertNotSame() {
        assertNotSame("should not be same Object", new Object(), new Object());
    }
```

```
    @Test
    public void testAssertNull() {
        assertNull("should be null", null);
    }

    @Test
    public void testAssertSame() {
        Integer aNumber = Integer.valueOf(768);
        assertSame("should be same", aNumber, aNumber);
    }

    // JUnit Matchers assertThat
    @Test
    public void testAssertThatBothContainsString() {
        assertThat("albumen", both(containsString("a")).and(containsString("b")));
    }

    @Test
    public void testAssertThathasItemsContainsString() {
        assertThat(Arrays.asList("one", "two", "three"), hasItems("one", "three"));
    }

    @Test
    public void testAssertThatEveryItemContainsString() {
        assertThat(
                Arrays.asList(new String[]{"fun", "ban", "net"}),
                everyItem(containsString("n"))
        );
    }

    // Core Hamcrest Matchers with assertThat
    @Test
    public void testAssertThatHamcrestCoreMatchers() {
        assertThat("good", allOf(equalTo("good"), startsWith("good")));

        assertThat("good", not(allOf(equalTo("bad"), equalTo("good"))));

        assertThat("good", anyOf(equalTo("bad"), equalTo("good")));

        assertThat(7, not(either(equalTo(3)).or(equalTo(4))));

        assertThat(new Object(), not(sameInstance(new Object())));
    }

    @Test
    public void testAssertTrue() {
        assertTrue("failure - should be true", true);
    }
}
```

其中關於 assert* 的函式第一個參數都可以指定 reason 讓每個判斷是都可以有個註解

使用

```
import static org.hamcrest.CoreMatchers.*;
import static org.junit.Assert.*;
```

讓 test case Assertion 更簡潔一點

範例專案

https://github.com/agileworks-tw/spring-boot-sample/blob/master/src/test/java/sample/TestAssertions.java

執行 `mvn -Dtest=TestAssertions test`

test fixture

test fixture 用來作為執行測試前的準備工作。目的是確保測試的環境是可重覆驗證，這樣測試結果才會是可重複的。可能使用情境如下：

- 準備輸入資料、建立或設定 mock object。
- 載入一組指定、已知的 database 資料
- 複製一些指定的檔案

JUnit 提供 annotation 給 test class 指定在每個測試前後的 fixture、 或是在執行所有 test method 前先執行一次性的 fixture。

總共有四個 fixture annotation，分為兩個 level：

class level

`@BeforeClass` 跟 `@AfterClass`

在 class level 可以進行資料庫連線相關，或是 fake / mock object。

method level

`@Before` 跟 `@After`

在 method level 只要有定義上述 annotation 每次進行 test method 時必會進行，可用於準備測試資料，並在測試完成後進行銷毀。

使用範例

```
package test;

import java.io.Closeable;
import java.io.IOException;

import org.junit.After;
import org.junit.AfterClass;
```

```java
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

public class TestFixtures {

    //  進行函式的 Mock
    static class ExpensiveManagedResource implements Closeable {
        @Override
        public void close() throws IOException {}
    }

    static class ManagedResource implements Closeable {
        @Override
        public void close() throws IOException {}
    }

    private ManagedResource myManagedResource;
    private static ExpensiveManagedResource myExpensiveManagedResource;


    @BeforeClass
    public static void setUpClass() {
        System.out.println("@BeforeClass setUpClass");
        myExpensiveManagedResource = new ExpensiveManagedResource();
    }

    @Before
    public void setUp() {
        System.out.println("@Before setUp");
        this.myManagedResource = new ManagedResource();
    }


    @Test
    public void test1() {
        System.out.println("@Test test1()");
    }

    @Test
    public void test2() {
        System.out.println("@Test test2()");
    }

    @After
    public void tearDown() throws IOException {
        System.out.println("@After tearDown");

        this.myManagedResource.close();
        this.myManagedResource = null;
    }

    @AfterClass
    public static void tearDownClass() throws IOException {
        System.out.println("@AfterClass tearDownClass");
        myExpensiveManagedResource.close();
```

```
            myExpensiveManagedResource = null;
        }

    }
```

範例專案

https://github.com/agileworks-tw/spring-boot-sample/blob/master/src/test/java/sample/TestFixturesExample.java

為了讓學員容易理解，輸出結果就如上面程式碼所排列的順序一樣

執行 `mvn -Dtest=TestFixturesExample test`

將輸出下列訊息

```
@BeforeClass setUpClass
@Before setUp
@Test test1()
@After tearDown
@Before setUp
@Test test2()
@After tearDown
@AfterClass tearDownClass
```

搭配 JUnit 進行 Rest API 測試

在進行 API 測試時，會有兩個部分，一個是專案本身為 API server，我們希望可以對 API 的 request 與 response 進行驗證。

一則是需要驗證第三方 API，比如 Facebook 或是 Google 所提供的權限驗證的 API 則是另外一個情境，此章節將針對兩個情形進行說明與演練。

server api 測試，以 Spring-boot 為例

若是 Server 內部 API 測試，會根據使用的 Framework 有所不同，此範例使用 Spring-boot。

參考範例

https://github.com/agileworks-tw/spring-boot-sample/blob/master/src/test/java/sample/data/rest/SampleDataRestApplicationTests.java

測試案例

透過 `this.mvc = MockMvcBuilders.webAppContextSetup(this.context).build();` 啟動 server
透過 `this.mvc.perform` 來對 API 進行存取

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(SampleDataRestApplication.class)
@WebAppConfiguration
@ActiveProfiles("scratch") //此行將載入 src/test/resources/application-scratch.propert
// Separate profile for web tests to avoid clashing databases
public class SampleDataRestApplicationTests {

    @Autowired
    private WebApplicationContext context;

    private MockMvc mvc;

    @Before
    public void setUp() {
        //mock api server
        this.mvc = MockMvcBuilders.webAppContextSetup(this.context).build();
```

```
    }

    @Test
    public void testHome() throws Exception {

        this.mvc.perform(get("/api")).andExpect(status().isOk())
                .andExpect(content().string(containsString("hotels")));
    }

    @Test
    public void findByNameAndCountry() throws Exception {

        this.mvc.perform(
                get("/api/cities/search/findByNameAndCountryAllIgnoringCase?name=Melb
                .andExpect(status().isOk())
                .andExpect(jsonPath("state", equalTo("Victoria")))
                .andExpect(jsonPath("name", equalTo("Melbourne")));
    }

    @Test
    public void findByContaining() throws Exception {

        this.mvc.perform(
                get("/api/cities/search/findByNameContainingAndCountryContainingAllIg
                .andExpect(status().isOk())
                .andExpect(jsonPath("_embedded.cities", hasSize(3)));
    }
}
```

測試第三方 API

參考範例

https://github.com/agileworks-tw/spring-boot-sample/blob/master/src/test/java/sample/data/rest/SampleDataRestApplicationTests.java

測試案例

以原 Sample 為例，測試步驟

1. 透過 `spring-boot:run` 來啟動 server
2. 執行 `mvn -Dtest=ThirdPartyApiTest test` 進行測試

這樣的測試方式通常會用於 TDD 剛開始導入時，先針對 API 進行驗證，也比較容易進行，缺點是因為是外部執行測試，所以自動化測試上步驟較繁瑣。

另外一種情形為 API 不屬於我們的維護範圍，為了確保第三方 API 運作正常，也就不得不使用次方式進行測試

```
package sample.data.rest;
import org.json.JSONObject;
import org.junit.Test;
import org.springframework.web.client.RestTemplate;
import static org.assertj.core.api.Assertions.assertThat;


public class ThirdPartyApiTest {

    final RestTemplate template = new RestTemplate();

    @Test
    public void testCitiesSearch() throws Exception {
        String api = "http://localhost:8000/api/cities/search/findByNameAndCountryAl
        String params = "?name=Melbourne&country=Australia";
        String url = api + params;

        String message = template.getForObject(url, String.class);
        JSONObject jsonObj = new JSONObject(message);
        assertThat(jsonObj.has("country")).isTrue();
    }
}
```

Junit Suite

當 JUnit 有多個或所有的 Test Case 要執行，此時就需要 Test Suite 來管理眾多的Test Case，Suite 可以用來管理 Test 的執行順序，也可以在所有 Test Case 開始或結束之前，統一進行設置。

Sample 專案

https://github.com/agileworks-tw/spring-boot-sample

新增 SuiteTest Class 定義 Test 執行順序

```
@RunWith(Suite.class)
@Suite.SuiteClasses({
        BTest.class,
        ATest.class
})
public class SuiteTest {
}
```

其中在

```
@Suite.SuiteClasses({
        BTest.class,
        ATest.class
})
```

所設定的 Test Class 的順序，將會依序進行，在沒有使用 SuiteTest 之前 Junit 將會根據 Class 字母依序執行，其執行順序為：

1. ATest.class
2. BTest.class

使用了上述程式碼的定義，執行順序將會調整為

1. BTest.class
2. ATest.class

如此一來，就可以根據實際狀況調整 test 執行的順序。

進行測試前後設置

在 SuiteTest Class 中可以在所有測試開始之前，以及所有測試結束之後進行統一的設置

經由使用 `@BeforeClass` 及 `@AfterClass` 來完成相關作業，程式碼如下：

```
@RunWith(Suite.class)
@Suite.SuiteClasses({
  BTest.class,
  ATest.class
})
public class SuiteTest {
    @BeforeClass
    public static void setUpClass() {
        System.out.println("=== Master setup ===");
    }

    @AfterClass
    public static void tearDownClass() {
        System.out.println("=== Master tearDown ===");
    }
}
```

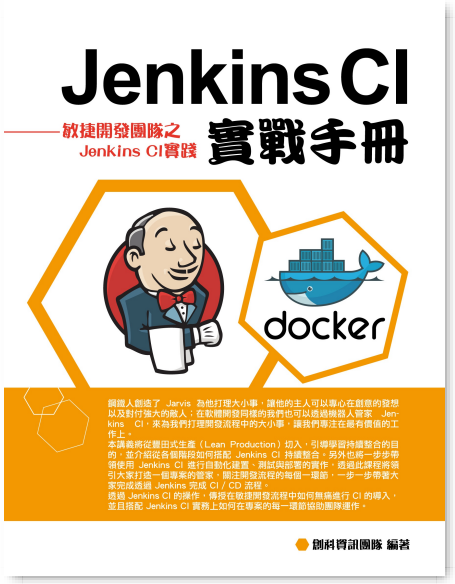
運行 Test 的情況會為：

1. SuiteTest.setUpClass
2. BTest.setUpClass
3. BTest 執行測試
4. BTest.tearDownClass
5. ATest.setUpClass
6. ATest 執行測試
7. ATest.tearDownClass
8. SuiteTest.tearDownClass

如此一來，我們就可以在所有 Test 開始前，進行如測試資料的準備，或是預先連接相關測試服務，方便進行測試。

Jenkins CI 實戰手冊

鋼鐵人創造了 Jarvis 為他打理大小事，讓他的主人可以專心在創意的發想以及對付強大的敵人；在軟體開發同樣的我們也可以透過機器人管家 Jenkins CI，來為我們打理開發流程中的大小事，讓我們專注在最有價值的工作上。



本講義將從豐田式生產（又稱為精實生產；Lean production ）切入，引導學習持續整合的目的，並介紹從各個階段如何搭配 Jenkins CI。另外也將一步步帶領使用 Jenkins CI 進行自動化建置、測試與部署的實作，透過此課程將領引大家打造一個專案的管家，關注開發流程的每個一環節，一步一步帶著大家完成透過 Jenkins 完成 Continuous Integration（持續整合）及 Continuous Delivery（持續交付；CD）流程。

透過 Jenkins CI 的操作，傳授在敏捷開發流程中如何無痛進行 CI 的導入，並且搭配 Jenkins CI 實務上如何在專案的每一環節協助團隊運作

閱讀須知

本講義之範例實作皆搭配 Ubuntu Linux 作業系統，請於課前先向活動主辦單位取得 VirtualBox 虛擬機器映像檔。

適合對象

適合對開發流程自動化感興趣的 IT 從業人員，或是導入敏捷開發卻無法敏捷的團隊學習，適合不排斥在 Linux 命令列輸入指令的人，因為大部份自動化程序都需要透過命令列輸入指令執行。

本講義將針對 Web 相關應用如 Web Application、API Server 相關開發應用進行演示（因 App 開發流程與 Web 有所差異，特別提醒 App 開發者留意）講師本身熟悉 Java 以及 JavaScript 開發，因此課程將以 Node.js 範例專案進行演練，但實際 CI 的導入應用觀念

與流程，可適用於大多數程式語言及開發工具。

講座與課程

本講義提供給參加 Jenkins CI 課程活動之學員。

- 資策會 舉辦之 DevOps：Jenkins + Docker 持續整合實務班
- iThome 舉辦之 Jenkins 持續整合實戰講堂
- 企業或學校辦理相關軟體開發培訓課程請聯繫 workshop@trunksys.com

講義作者

本講義由創科資訊研發團隊的 Spooky 及 Kyle 編著，如需要分享請先獲得授權許可。

Jenkins Quick Start

安裝提示

1. 安裝 Ubuntu Linux 16.04 LTS
2. 安裝 Java

```
sudo apt-get install openjdk-8-jdk
```

3. 安裝 Jenkins

```
wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add
sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian-stable binary/ > /etc/apt/
sudo apt-get update
sudo apt-get install jenkins
```

4. 設定 Jenkins

```
# /etc/default/jenkins
HTTP_PORT=8080
JAVA_ARGS="-Djava.awt.headless=true -Xmx1024m"
```

5. 設定 sudo 權限與免密碼

```
# /etc/passwd
jenkins:x:111:117:Jenkins,,,:/var/lib/jenkins:/bin/bash

# /etc/sudoers
%sudo   ALL=(ALL:ALL) NOPASSWD: ALL

# /etc/group
sudo:x:27:user,jenkins
```

6. 管理 Jenkins

```
# Upgrade
```

```
sudo apt-get update && sudo apt-get upgrade jenkins

# Start, Restart, Stop
sudo service jenkins start
sudo service jenkins restart
sudo service jenkins stop
```

Hello! Jenkins!

難度： 簡易

目標： 認識 Jenkins Job 基礎操作。

專案： https://github.com/agileworks-tw/java-hello-world

練習：

1. 新增作業（Jenkins Job）名稱： `HelloWorld`
2. 使用 Shell 指令檢查 JDK 版本

   ```
   java -version
   javac -version
   ```

3. 使用 Git SCM，設定 Git Repository URL

   ```
   https://github.com/agileworks-tw/java-hello-world
   ```

4. 使用 `javac` 指令進行編譯、 `java` 指令執行
5. 使用 curl 指令操作 Remote API： `curl http://localhost:8080/job/HelloWorld/build`
6. 自訂字串參數： `WHO` ，並帶入 Shell 指令
7. 設定 Git 分支（branch）

   ```
   with-name-argument
   ```

8. 使用 curl 指令操作參數化建置 Remote API： `http://localhost:8080/job/HelloWorld/buildWithParameters?WHO=Mary`
9. 使用 Git Parameter 參數化建置切換分支： `BRANCH_NAME` ，並帶入 Git SCM 設定
10. 利用 Remote API 回傳 XML 或 JSON 格式的 Jenkins Job 定義資料
11. 使用 Jenkins CLI 刪除 Job

參考資料

Remote API 範例

```
curl http://localhost:8080/job/HelloWorld/api/json | python -m json.tool
```

```
curl http://localhost:8080/job/HelloWorld/api/json | xmllint --format -
```

Jenkins CLI 範例

```
java -jar jenkins-cli.jar -s http://localhost:8080/ delete-job HelloWorld
```

Test Report: JUnit v.s. TAP

難度： 簡易

目標： 瞭解 Jenkins 的測試報表（Reporting）功能。

使用 YSlow for PhantomJS。

PhantomJS is a headless WebKit with JavaScript API. YSlow for PhantomJS is a command line script that allows page performance analysis from live URLs, unlike YSlow for Command Line (HAR) where a pre-generated HAR file is needed in order to analyze page performance.

http://yslow.org/phantomjs/

練習：

1. 新建 Free-Style Jenkins Job： `YSlow`
2. 執行 Shell 指令檢查 `phantomjs` 版本
3. 下載並解壓縮： `http://yslow.org/yslow-phantomjs-3.1.8.zip`
4. 執行 `phantomjs` 產生 JUnit 格式的測試報告 yslow.xml
5. 使用 Post-build actions 發佈 JUnit 測試結果報告
6. 改用 TAP Test Results 輸出測試結果

PhantomJS + YSlow 指令範例：

```
# junit
phantomjs yslow.js -i grade -threshold "B" -f junit http://localhost:8080/ > yslow.xm

# tap
phantomjs yslow.js -i grade -threshold "B" -f tap http://localhost:8080/ > yslow.tap
```

Markdown Converter

難度： 簡易

目標： 使用 Jenkins 處理轉檔任務並保存結果。

專案： https://github.com/agileworks-tw/markdown-example

練習：

1. 建立 Free-Style Jenkins Job： `Markdown`
2. 執行 Shell 指令檢查 `pandoc` 版本
3. 設定 Git Repository URL
4. 將 `README.md` 轉成 html 文件格式
5. 保存 `README.html` 文件

```
pandoc -s README.md -o README.html
```

Using Gradle

難度： 普通

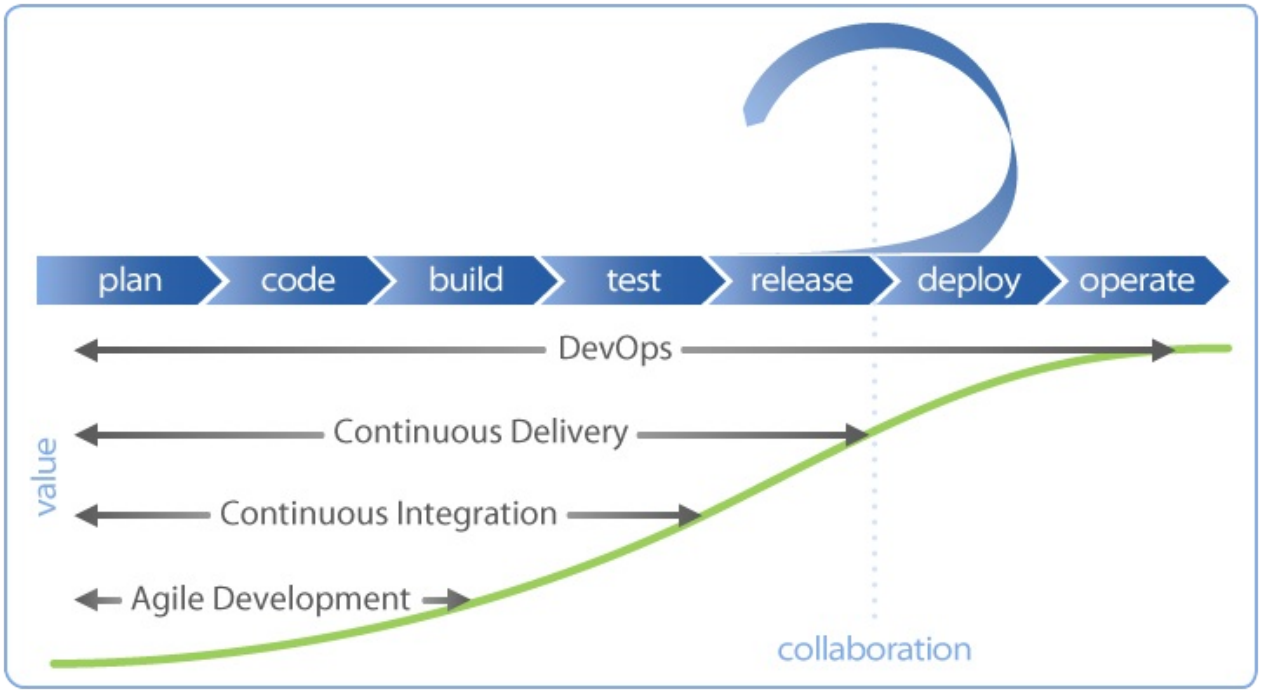
目標： 使用 Gradle 自動化建置與測試 Java 專案。

專案： https://github.com/agileworks-tw/gradle-example

練習：

- 建立 Free-Style Jenkins Job： `gradle-example`
- 執行 Shell 指令檢查 `gradle` 版本
- 設定 Git Repository URL
- 執行 `gradle build` 指令
- 封存產生的 .jar 檔案
- 建立並封存 JUnit Report
- 建立並封存 JavaDoc Report

基本概念



上圖中可以看到每個階段都有其對應要處理的事項，而關於這些不同階段的事項所要解決的事情如下：

1. Agile Development：有效率的溝通，做最有價值的事。
2. Continuous Integration：自動化測試，自動化建置，讓重覆的事情可以自動執行。
3. Continuous Delivery：讓專案可以自動發佈，隨時有穩定版本可以運作，加速 test 驗證。
4. DevOps：確保運行環境正常，負責將上述各個機制建置及維護，還有一旦 production deployment 之後的維運。

當然，實際狀況絕對不是 DevOps 這個職位的人單一可以完成上述所有的整合，必須要專案開發人員與及規劃時皆已考慮才有可能順暢的進行。

上述每一個環節都是要避免所有不必要的浪費

1. Agile Development：避免用昂貴的人力（你）無效的溝通。
2. Continuous Integration：避免用昂貴的人力（你）進行重覆測試，避免用昂貴的人力（你）進行重覆專案的建置及測試。
3. Continuous Delivery：避免用昂貴的人力（你）處理 produciton deployment 的瑣碎細節。

除了解決不必要的浪費，還有避免人為錯誤，一旦所有程序都自動化後，錯誤將容易被排除驗證。

就像豐田式生產（TPS, Lean Production）所提倡的理念：杜絕浪費；每一次的浪費一旦被避免，也就表示有更多時間資源可以進行有價值的事！

上述內容節錄自：企業端如何推動持續整合開發流程

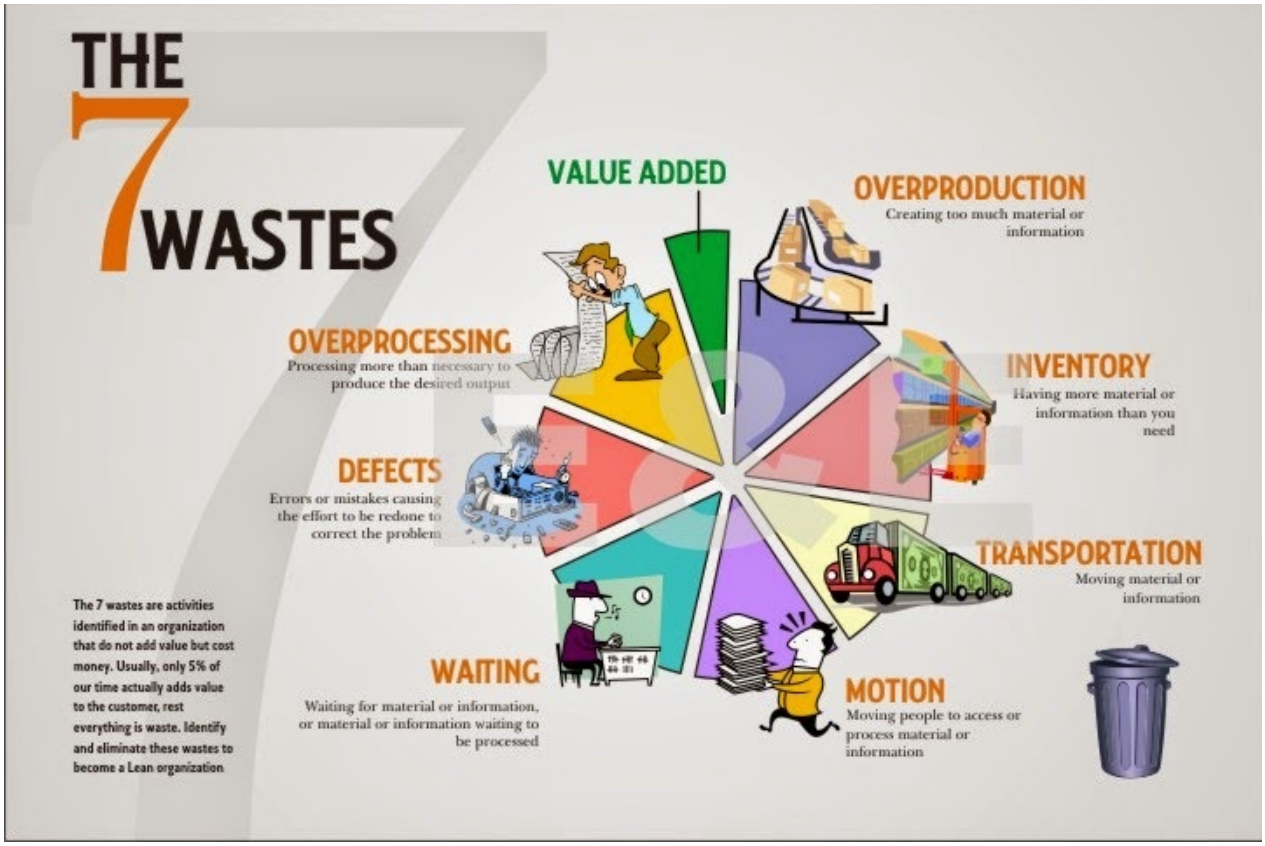
刊登於 ithome：【JSDC客座文章】企業端如何推動持續整合開發流程

豐田式生產

如果想瞭解持續整合的目的，就需要先知道豐田式生產管理（Toyota Management）的概念，豐田式生產又稱為精實生產（Lean Producton）。

根據 wiki：豐田生產方式

> 豐田式生產主要目標，是杜絕負荷過重（muri）、不平準（mura）和一切的浪費（muda）。TPS 把浪費歸納成七種：
>
> 1. 等待的浪費；
> 2. 搬運的浪費；
> 3. 不良品的浪費；
> 4. 動作的浪費；
> 5. 加工的浪費；
> 6. 庫存的浪費；
> 7. 製造過多（早）的浪費。



TPS 背後的理念是杜絕浪費，因而不再需要庫存。基本原則是「豐田模式」（The

Toyota Way），概述如下：

1. 正確的流程方能產生正確結果
2. 發展員工與事業夥伴，以為組織創造價值
3. 持續解決根本問題是組織型學習的驅動力

從製造業的角度來看，能夠避免不必要的浪費跟重工，相對的效率就會變快，我們可以把 TPS 歸納的七種浪費就軟體開發的角度來看：

1. 等待的浪費，搬運的浪費：程式開發週期除了寫程式之外 build, test, deploy 都需要處理，但可以自動化
2. 不良品的浪費：程式錯誤一再發生，沒辦法杜絕，若能在每個環節測試確認，則可以確定每個動作沒有問題
3. 動作的浪費，加工的浪費：可以自動化，就不需要人的接入
4. 庫存的浪費：軟體沒有庫存壓力
5. 製造過多（早）的浪費：重覆的動作，就是一種浪費

所有在製造業所會發生的問題，大部份軟體開發也會有同樣的問題，我們可以借鏡其精神，與豐田式生產相互輝映，發展出敏捷式開發（agile），下一章節將近行介紹。

敏捷式開發

說明持續整合，就不得不說明敏捷軟體開發（Agile），跟豐田式生產一樣，敏捷式開發也致力於杜絕所有可能的浪費。

參考 Wikipedia 中的說明：敏捷軟體開發

> 敏捷軟體開發（英語：Agile software development），又稱敏捷開發，是一種從1990 年代開始逐漸引起廣泛關注的一些新型軟體開發方法，是一種應對快速變化的需求的一種軟體開發能力。它們的具體名稱、理念、過程、術語都不盡相同，相對於「非敏捷」，更強調程式設計師團隊與業務專家之間的緊密協作、面對面的溝通（認為比書面的文檔更有效）、頻繁交付新的軟體版本、緊湊而自我組織型的團隊、能夠很好地適應需求變化的代碼編寫和團隊組織方法，也更注重軟體開發中人的作用。

敏捷開發宣言

- 個人與互動 **重於** 流程與工具：避免資訊的錯誤，浪費溝通的時間。
- 可用的軟體 **重於** 詳盡的文件：若詳盡的文件就是可用的軟體，可避免維護兩個地方，且文件就是運作結果本身。
- 與客戶合作 **重於** 合約協商：客戶才是需求的本身，與客戶合作將更接近目標，不會花錯力氣。
- 回應變化 **重於** 遵循計劃：唯一不變的就是變，計畫趕不上變化，如果只是不知變通的依計劃進行，往往知道方向錯了已來不及。

持續整合改善

有了可運行的規格以及每次異常產生的規格，接著就要讓日常累積的開發能量，能每日運行規格成為品質的後盾。

搭配 CI 定時產出報表，隨時注意專案的健康度，即早發現問題即早治療，在這要說明如何搭配 CI（Continous Integration），讓這一連串的活動能夠持續運行。

Daliy Build（or Nightly Build）

1. 每天早上以及中午進行一次建置
2. 建置前需要通過所有的規格運行
3. 一旦發生錯誤，一律需要當天確認，並且當天修正完畢
4. 一旦建置完成 deploy 到 develop 機器進行測試

Weekly Build

1. 每個週末選定一天進行建置
2. 建置前需要通過所有的規格運行
3. week build 可選擇是否 deploy 到 production 機器

每天每週的建置，正如豐田式生產一樣，發生問題即早面對並解決，避免產出不良品，修正完成再繼續往前，將異常或是浪費降到最低。

結論

程式開發是一連串的協同合作，人為疏失是沒有辦法根絕的，但卻可以有機制讓問題及早發現，TDD 正是一個良好的開發流程，讓開發過程包含了驗證機制，確保開發能量能夠被累積並且重覆使用，隨著專案的成長，偵錯的能樣也會同步成長。

日常有 TDD 的習慣，導入 CI 水到渠成，利用 CI 讓專案定時被健康檢查，專案上線或是更新不再提心吊膽，在令人安心的環境下開發，更能專注於系統功能的完善，我想有遵循 TDD 開發流程的團隊是幸福的，就像有爸媽呵護一樣不用擔心受怕 :)

持續整合流程從豐田式生產而來，其所提倡的核心目標是剛剛好，不浪費，以及透明，並且透過持續整合 Jenkins 的幫忙，讓專案有“心跳”，若沒有正常跳動盡早急救！

Jenkins 可以在 Windows、Linux 或 Mac OS X 環境執行，一般來說我們會盡可能選擇跟 Production / Testing 伺服器接近的環境來運行。

本書範例皆以 Ubuntu Linux 演示

因為 Ubuntu Linux 相容多數 Open Source 軟體，也被用於許多 Production Server 環境，因此使用 Ubuntu Linux 作為 Jenkins 入門的系統，可以避免很多干擾學習的問題發生。

建議搭配 VirtualBox 虛擬機器使用 Ubuntu Linux + Jenkins，我們也為學習者提供已安裝設定完成的 VirtualBox Image，請向課程主辦單位取得檔案下載位址。

安裝 Java 軟體

建議使用 JDK 1.8 以上版本。

安裝 java 8 & maven

```
sudo add-apt-repository ppa:openjdk-r/ppa
sudo apt-get update
sudo apt-get install openjdk-8-jdk
```

選擇 java 8 作為預設 java version

```
sudo update-alternatives --config java
sudo update-alternatives --config javac
```

安裝 Jenkins 軟體

參考 Jenkins 官方的 Ubuntu Linux 安裝說明

```
wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ > /etc/apt/sources.list
sudo apt-get update
sudo apt-get install jenkins
```

管理 Jenkins 伺服器

重新啟動 Jenkins 伺服器。

```
sudo service jenkins restart
```

設置建置環境

Jenkins 有許多社群貢獻的第三方 Plugins：

- NodeJS Plugin
- Rvm

但是以筆者過去經驗來說，就遇過安裝 Node.js Plugin 就會影響 RVM Plugin 的運作，安裝過多的 Plugins 可能導致許多問題。

除非您已經瞭解這些 Plugins 的特性及可能帶來的副作用，否則並不建議一開始就安裝太多 Plugins。

事實上絕大多數的建置流程，都可以利用 Jenkins 內建的任務定義方式來完成；如果搭配 Docker 來用更可以大幅化簡開發環境的配置工作。

一般來說，我們建議直接使用執行 Jenkins 程式的使用者帳號，把相關所需的環境設置完成。

配置 Jenkins 使用者的環境

在 Ubuntu Linux 利用 `apt-get` 完成 Jenkins 安裝後，會自動產生名為 `jenkins` 的系統使用者帳號（user）及群組（group）。

`/etc/passwd`

```
jenkins:x:117:125:Jenkins,,,:/var/lib/jenkins:/bin/bash
```

`/etc/group`

```
jenkins:x:125:
```

在 Jenkins 定義的任務，預設會使用 `jenkins` 使用者的權限進行操作，因此並非所有的檔案皆可任意讀寫或執行。

檔案權限的觀念在 Linux 系統中很重要，你在自己或其他帳號的家目錄中安裝的程式或放置的檔案，並不一定能夠被 Jenkins 的任務使用。

如果你希望幫 `jenkins` 帳號增加一些專屬的配置，需要在終端機中使用 `sudo su jenkins` 可以切換成 `jenkins` 使用者的身份進行各項操作，該 user 的家目錄（Home Directory）位置是 `/var/lib/jenkins`。

不使用 plugin 來安裝相關 library 的好處

除了 task 的執行之外，另外一個原因是，在除錯的過程中讓 jenkins CI 跟 jenkins user 同樣環境設置方式將會有利於加速確認環境是否正確或是有異常時可以直接用 jenkins user 進行 debug

這件事非常重要，因為若都只能用 task 確認，每次確認結果將會比較慢，執行上會比較有挫折感。

相同的，若我們需要新增新的處理，也可以進行直接用 jenkins user 進行，將會更直接的把自動化需求完成，一旦確認沒有問題 理論上再把相關指令讓 jenkins CI 執行就會正確！

TDD 的精神無誤。

SSH

為 Jenkins User 建立 SSH Keys

SSH 在自動化的過程扮演很重要的角色，因此我們需要先將 jenkins 的 ssh key 產生出來

透過 jenkins user 建立 ssh key

`ssh-keygen -t rsa`

default 會產生在 `~/.ssh` 將會有 `id_rsa`, `id_rsa.pub` 這兩個檔案

將建立好的 public key 加入到要傳輸的目標機器

`cat ~/.ssh/id_rsa.pub >>~/.ssh/authorized_keys`

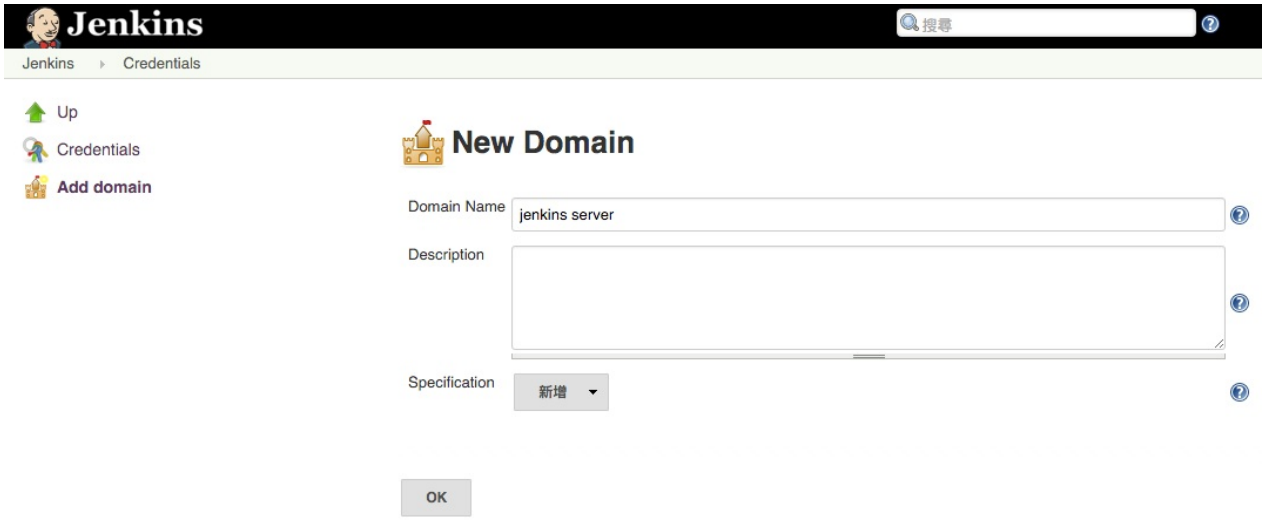
有了 ssh key 以便後續自動化進行處理，在 jenkins user 底下執行上述指令，也就表示：

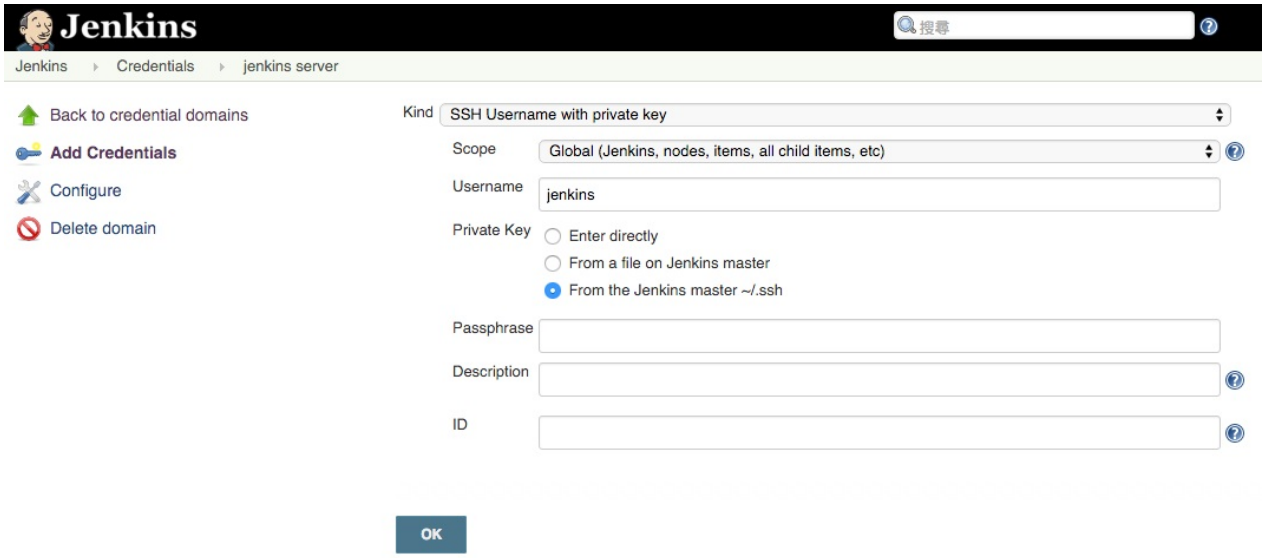
> 把本機當作 stage machine

這樣做的好處是，一旦 stage machine 在其他位置

只要改變 ssh server ip 並且再把 id_rsa.pub 加入對象機器即可，其他程序皆相通

設定 credentials





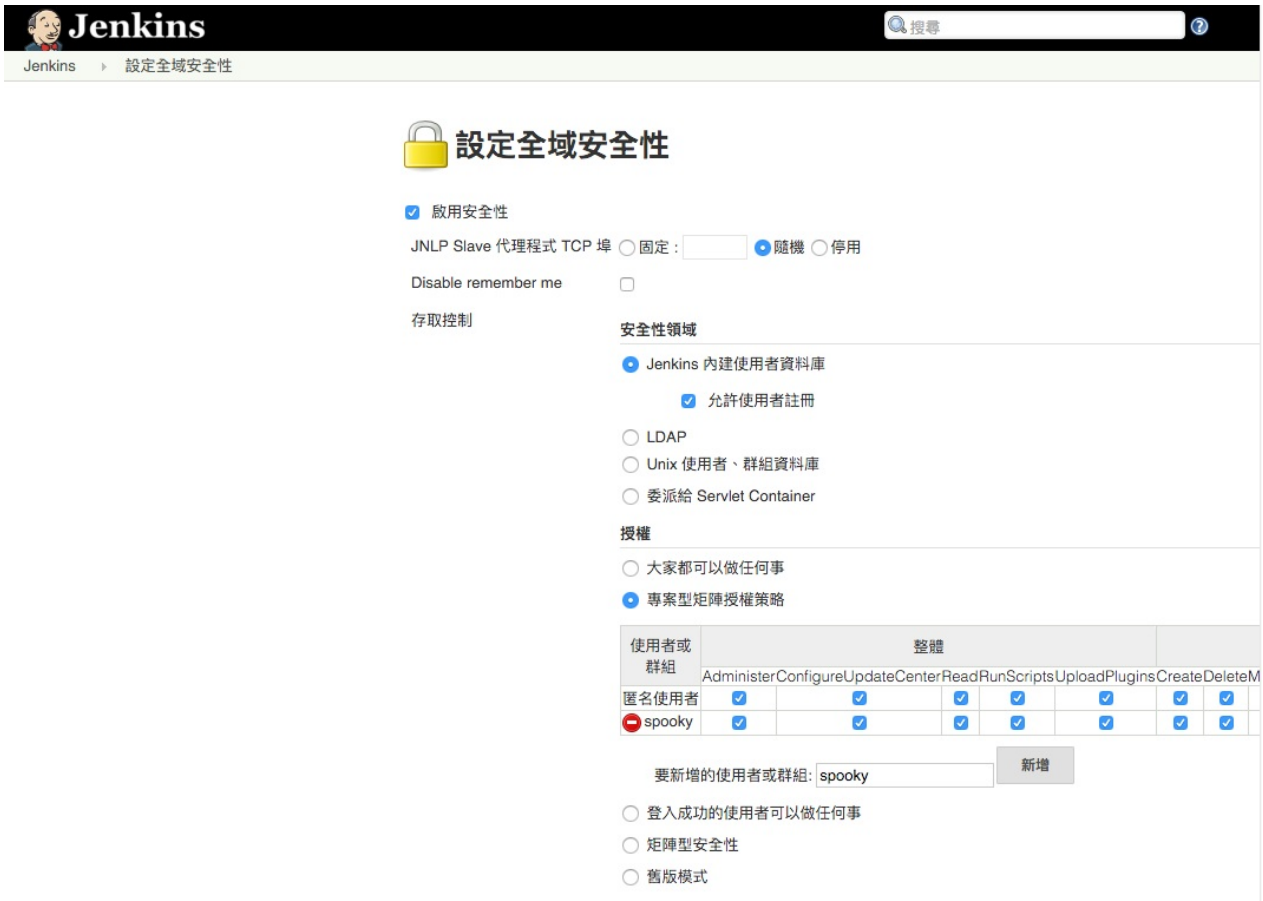
新增 ssh server

我們可以先把 jenkins CI server 視為 development 測試環境，因此我們需要進行 ssh server 相關設置

相關設置參考：plugin/publish over ssh 透過該章節設置，我們可以確保 ssh 連結正常

設置安全性

設置畫面



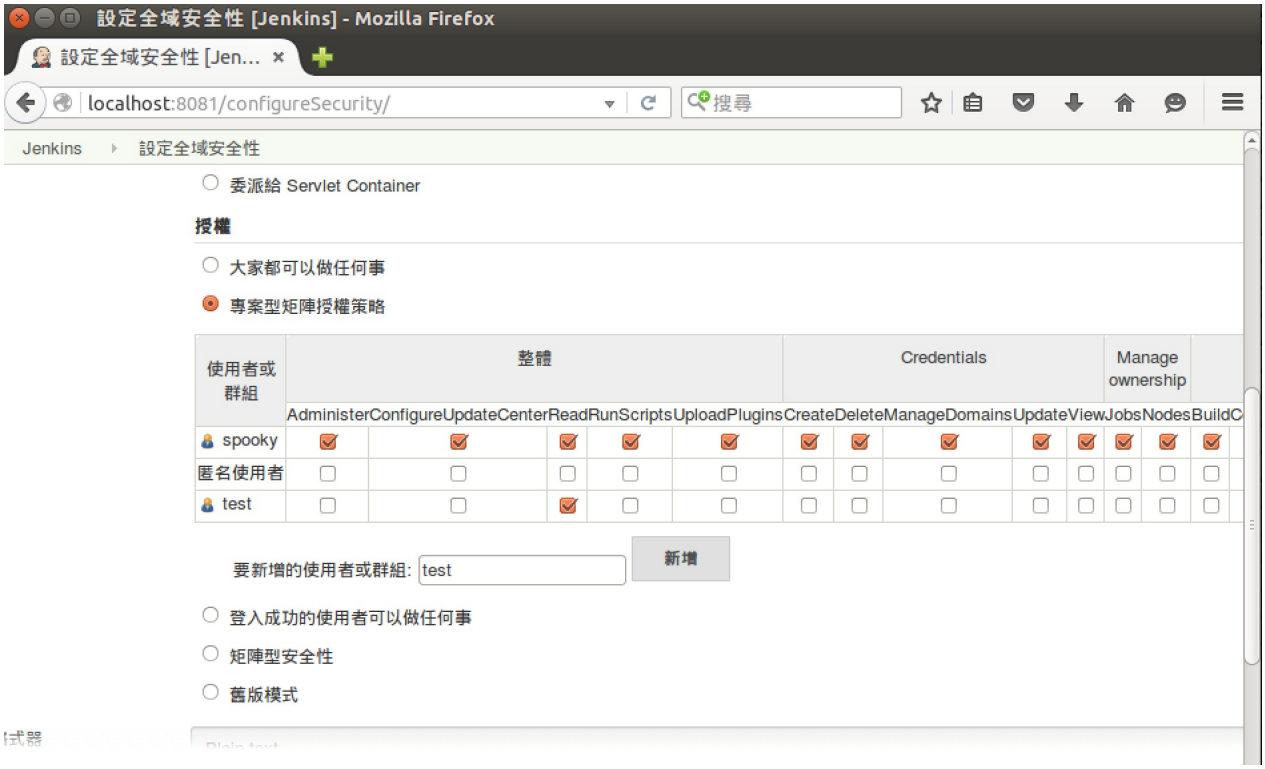
專案型矩陣授權策略

使用時機

希望可以控管特定 task 可被特定人員存取，例如：production deploy 時，只有 DevOps 人員可以進行 release task 的執行

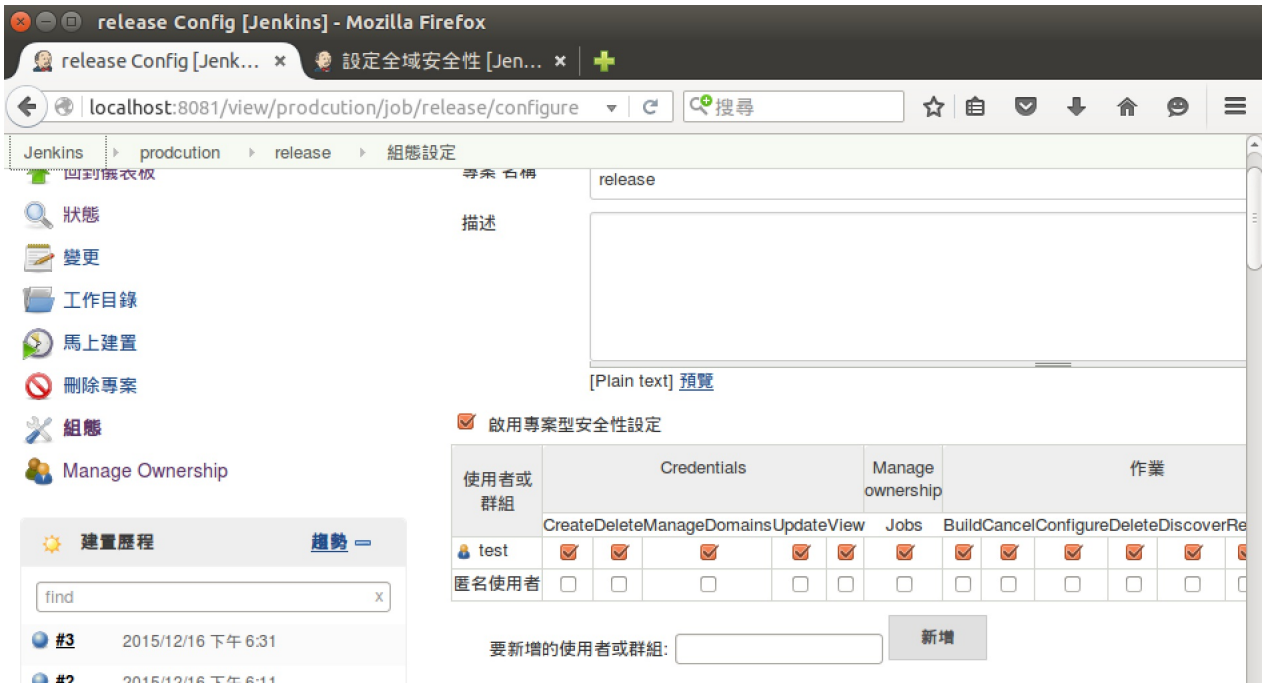
設置方式

在 `Jenkins > 設定全域安全性` 使用 `專案型矩陣授權策略` 並且在新增人員至少要有整體 `read` 權限，如下圖：



需要注意的是，若使用者有設置 `整體 Administer` 權限，預設將可以看到所有的 task。

接著，我們可以進入特定 task 進行權限設置，如下圖：



最後再使用 test 這個 user 進行登入後，將只能看到有設置可以存取的 task 被顯示出來

儀表板 [Jenkins] - Mozilla Firefox (隱私瀏覽)
儀表板 [Jenkins]
localhost:8081
搜尋
Jenkins
搜尋
test
登出
Jenkins
開啟自動更新頁面
使用者
建置歷程
我的視景
建置佇列
佇列中沒有建置作業。
demo
prodcution
全部
S
W
名稱 ↓
上次成功時間
上次失敗時間
上次建置花費時間
release
10 天 - #3
無
1 分 41 秒
圖示: S M L
圖例
RSS 摘要: 全部
RSS 摘要: 失敗
RSS 摘要: 最近幾次建置

Master/slave

節點套件安裝

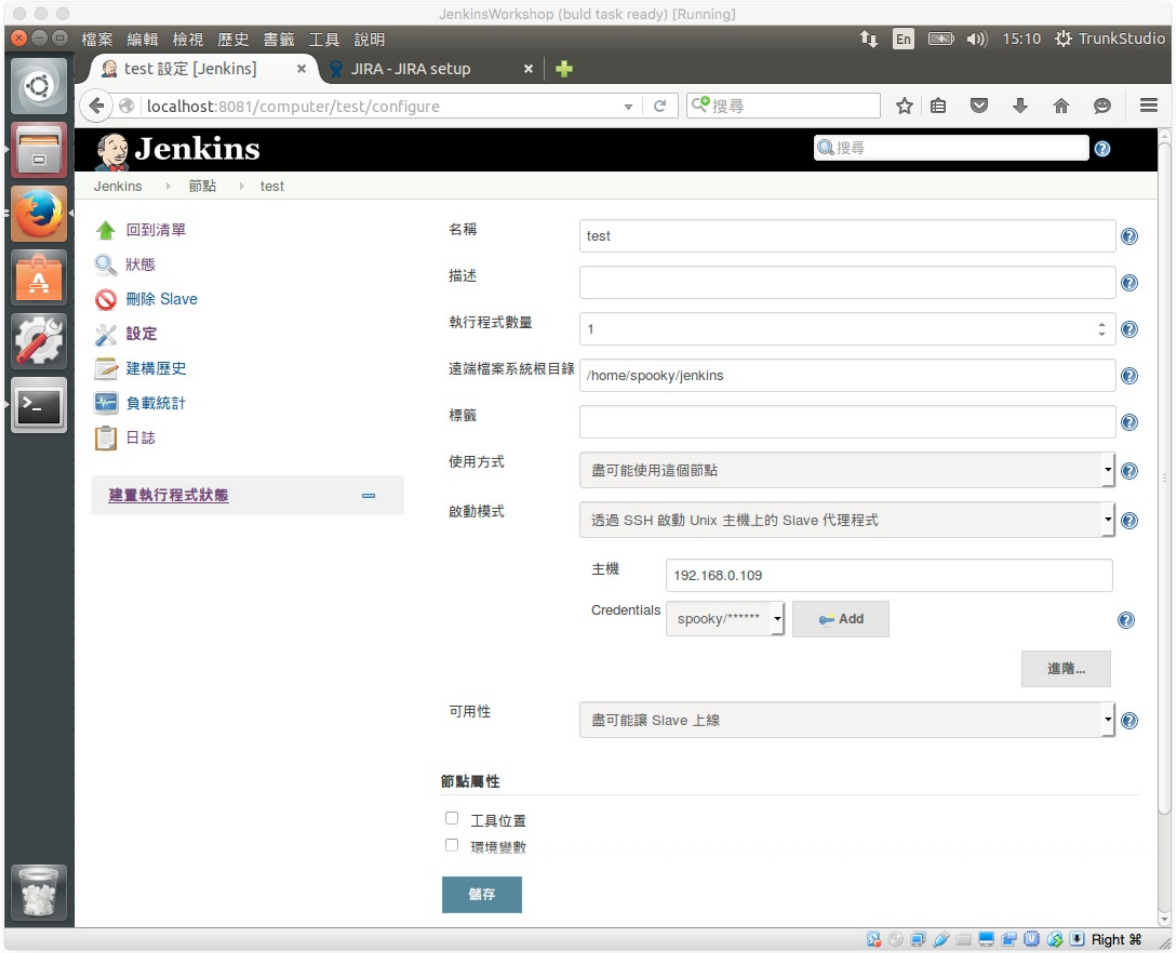
```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install default-jre
```

節點使用者設定 ssh

新增你要用來執行 slave 的 user，e.g. slave

參考 ssh 進行設定

jenkins slave 設置



slave 無網路情形

可以使用 `在 Master 上執行指令啟動 Slave`

參考 jenkins 內的說明

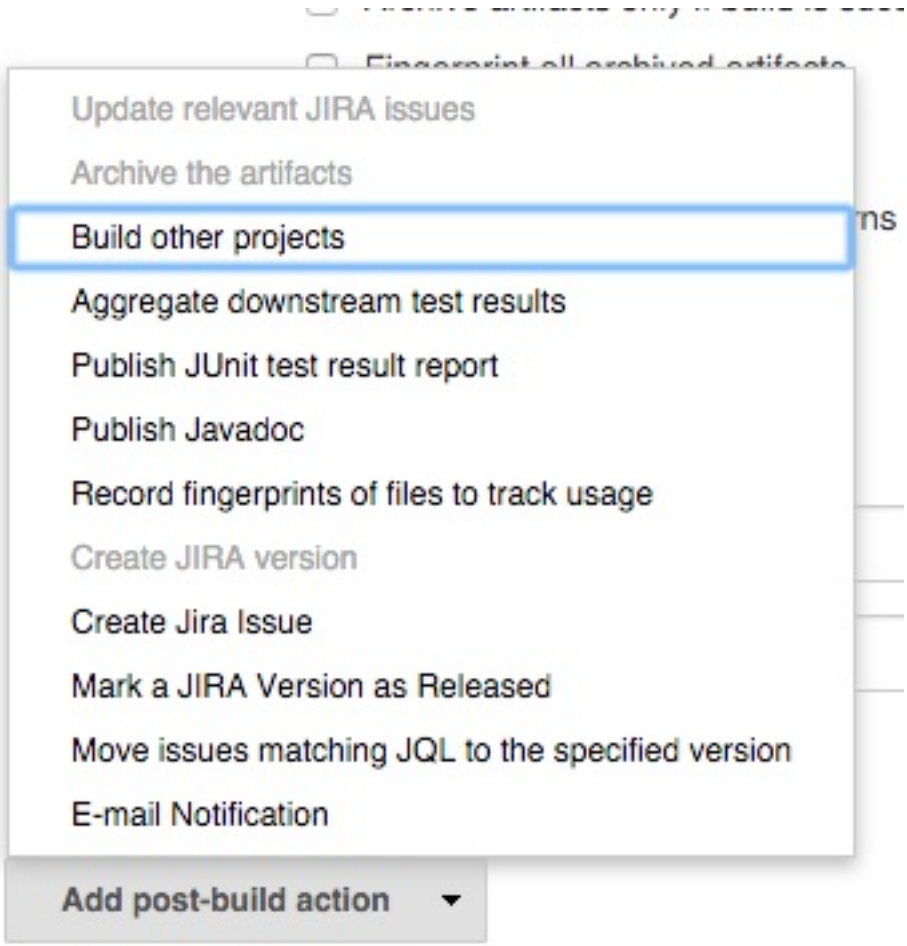
> 啟動 Slave 代理程式的指令，可以控制 Slave 電腦並與 Master 溝通。 Jenkins 假設執行的程式會在正式的 Slave 機器上啟動 slave.jar。
>
> 可以由 http://${jenkinsServerHost}/jnlpJars/slave.jar 下載 slave.jar。
>
> 簡單一點就像 "ssh 主機名稱 java -jar ~/bin/slave.jar"。 但是，一般會建議您在 Slave 上面寫一個小 Shell Script，控制 Java 及 slave.jar 的位置， 也能設定 PATH 這類節點間不盡相同的環境變數。就像:
>
> # !/bin/sh exec java -jar ~/bin/slave.jar

> 您可以使用任何指令執行 Slave 機器上的程式，例如 RSH。 只要最後程式的 stdin 及 stdout 被連到 "java -jar ~/bin/slave.jar" 就好。
>
> 大型部署環境下，可以考慮從掛載 NFS 的共通位置中載入 slave.jar， 就不用每次升級 Jenkins 時還要同步更新每部機器上的這個檔案。
>
> 設定成 "ssh -v 主機名稱" 可以幫助您處理連線問題。

build archive

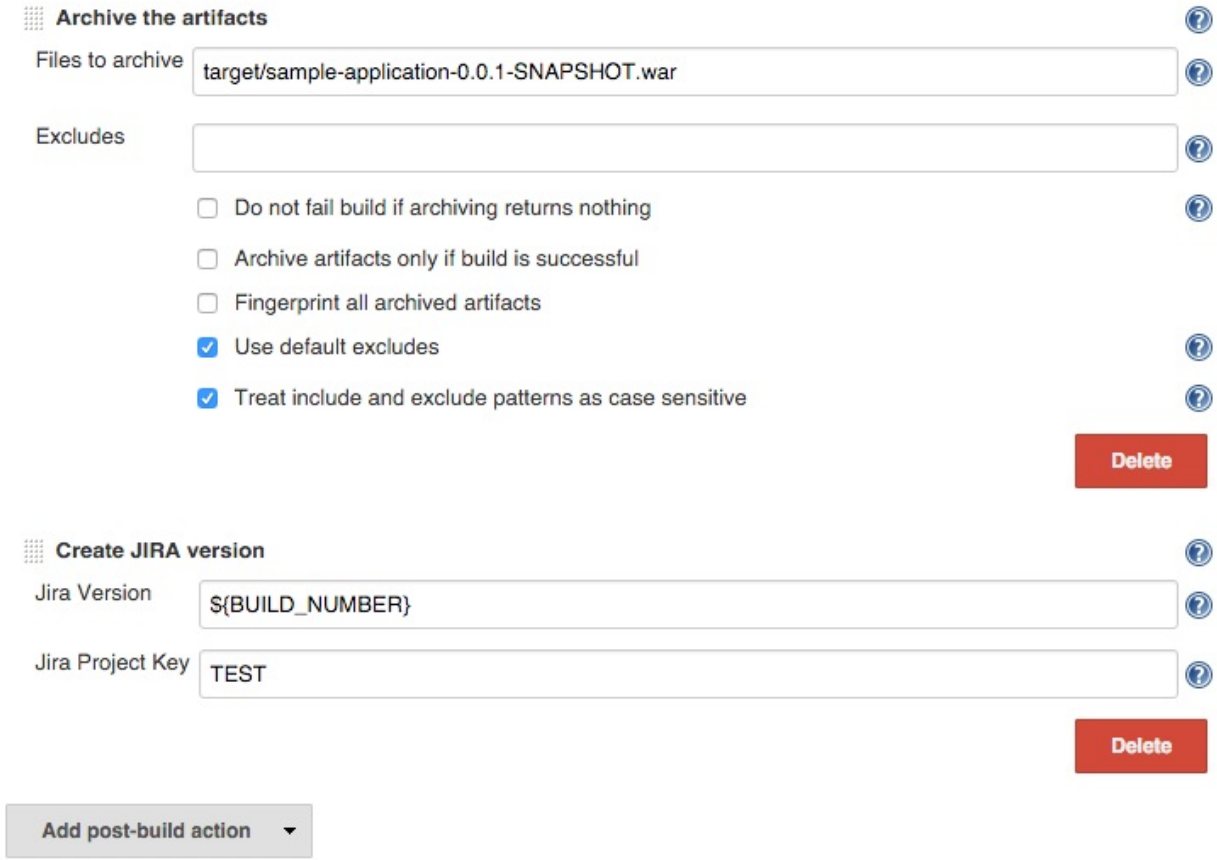
Archive the artifacts

參考下圖進行選曲



選擇 `Archive the artifacts` 此選項

設置要封存的檔案

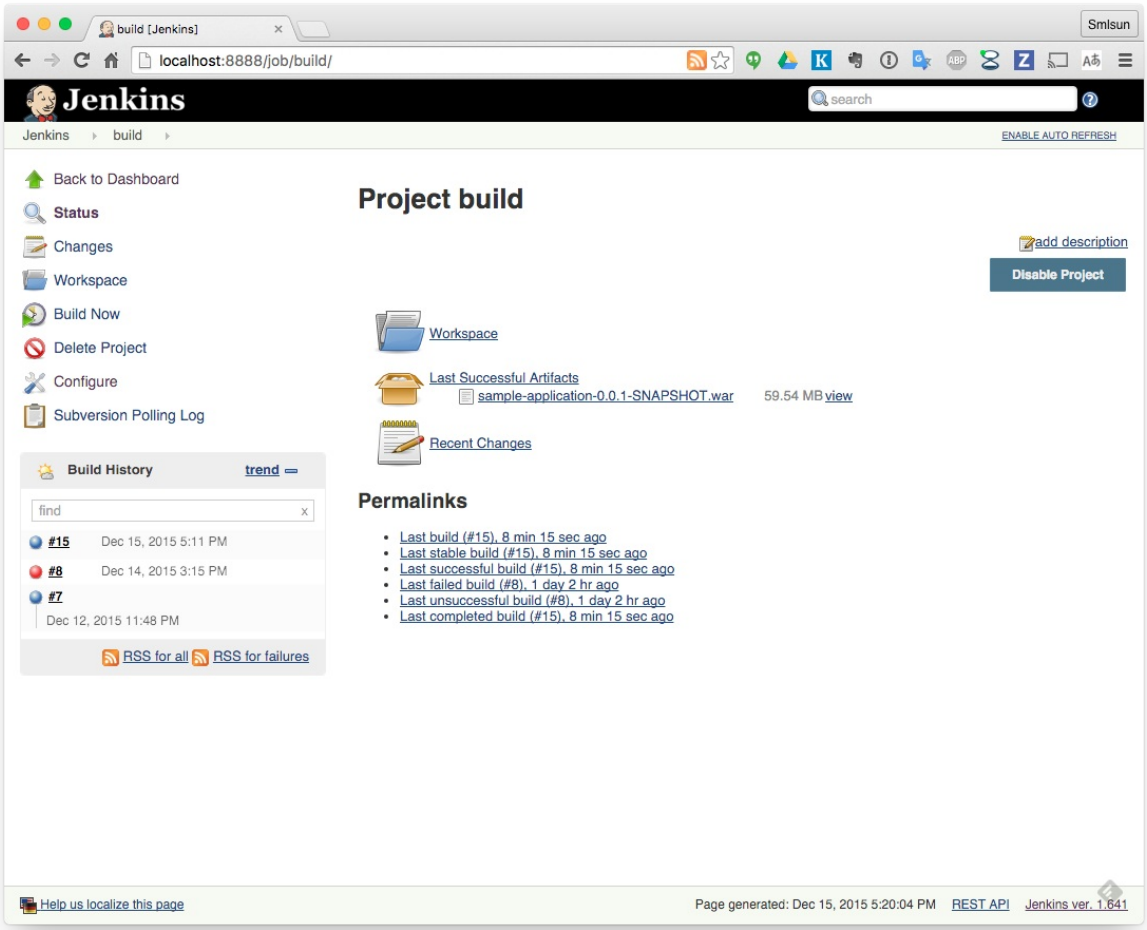


此例而言，我們可以填入 `target/sample-application-0.0.1-SNAPSHOT.war`

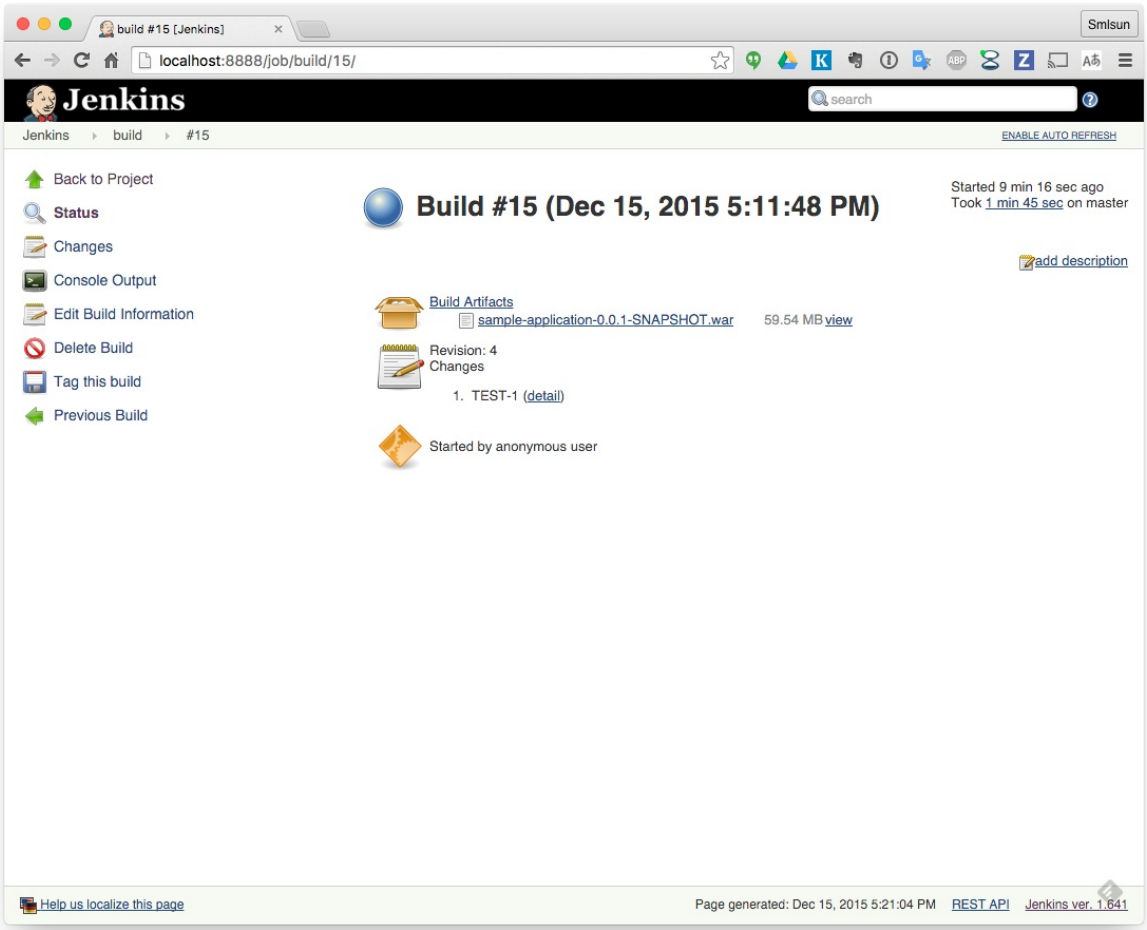
透過 `mvn -Pprod package` 此指令將會產生上述檔案

檢視封存結果

overview


build [Jenkins]
localhost:8888/job/build/
Jenkins
search
Jenkins
build
ENABLE AUTO REFRESH
Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Project
Configure
Subversion Polling Log
Build History
trend
find
#15 Dec 15, 2015 5:11 PM
#8 Dec 14, 2015 3:15 PM
#7 Dec 12, 2015 11:48 PM
RSS for all
RSS for failures
Project build
add description
Disable Project
Workspace
Last Successful Artifacts
sample-application-0.0.1-SNAPSHOT.war 59.54 MB view
Recent Changes
Permalinks
Last build (#15), 8 min 15 sec ago
Last stable build (#15), 8 min 15 sec ago
Last successful build (#15), 8 min 15 sec ago
Last failed build (#8), 1 day 2 hr ago
Last unsuccessful build (#8), 1 day 2 hr ago
Last completed build (#15), 8 min 15 sec ago
Help us localize this page
Page generated: Dec 15, 2015 5:20:04 PM
REST API
Jenkins ver. 1.641

task



可以看到 jenkins 將會提供連結讓使用者可以進行下載

執行 Shell Script

在專案建置過程中，經常需要執行 Shell Script 來完成某些任務，例如：

```
make clean
eb deploy
```

在平時開發階段所使用的各種專案建置指令，只要能用 Shell Script 完成，就可以定義在 Jenkins CI 的任務中。

Jenkins CI 在執行自訂的 Shell Script 時，會帶入以下專案建置所需的各項環境變數。

- BUILD_NUMBER
- BUILD_ID
- BUILD_DISPLAY_NAME
- JOB_NAME
- BUILD_TAG
- EXECUTOR_NUMBER
- NODE_NAME
- NODE_LABELS
- WORKSPACE
- JENKINS_HOME
- JENKINS_URL

- BUILD_URL
- JOB_URL
- SVN_REVISION
- SVN_URL

我們可以利用這些參數，例如產生帶有建置流水號的打包檔。

顯示所有環境變數

```
env
```

參數化建置

以 JDBC 連線字串為例：

```
JDBC_CONNECTION_STRING
```

Shell Script 常見指令

在 Jenkins CI 定義 Shell Script，必須「避免太過複雜」。如果真的需要複雜的 Shell 指令，應該先寫成一個 Script 檔案，然後直接執行它。

jenkins api 使用

可以參考

https://github.com/jansepar/node-jenkins-api

api 有：

- lastBuild
- lastStableBuild
- lastSuccessfulBuild
- lastFailedBuild
- lastUnstableBuild
- lastUnsuccessfulBuild
- lastCompletedBuild

常用 API

last success: 取得最後建置版號

http://localhost:port/job/build/lastSuccessfulBuild/api/json

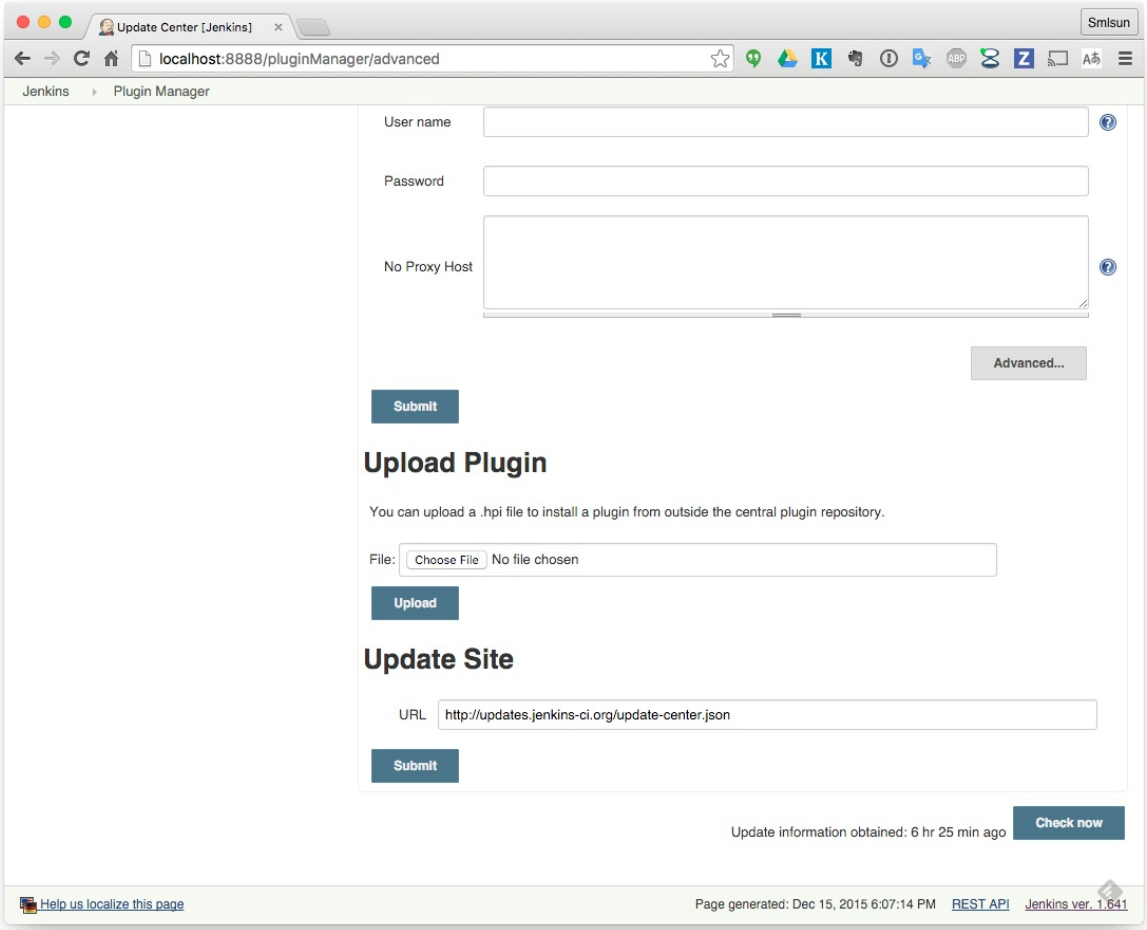
使用於取得版號資訊，以及 changelog

Plugin 安裝

離線安裝

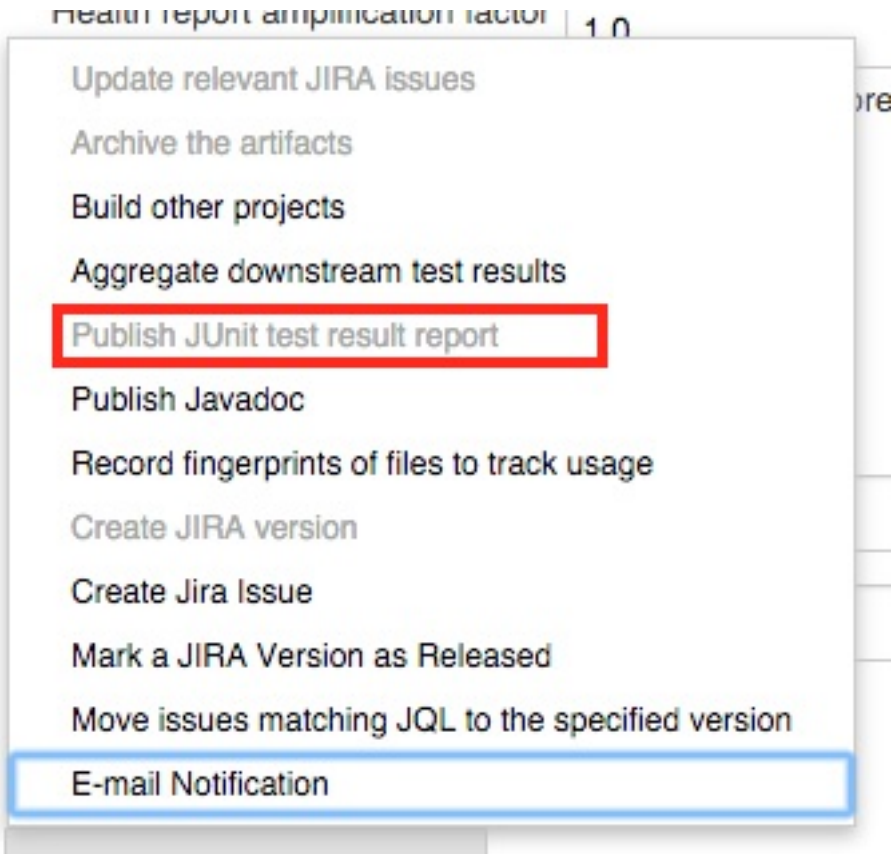
Download the Plugin (*.hpi File) and put it in the following directory:
%JENKINS_HOME%/plugins/

透過檔案上傳安裝

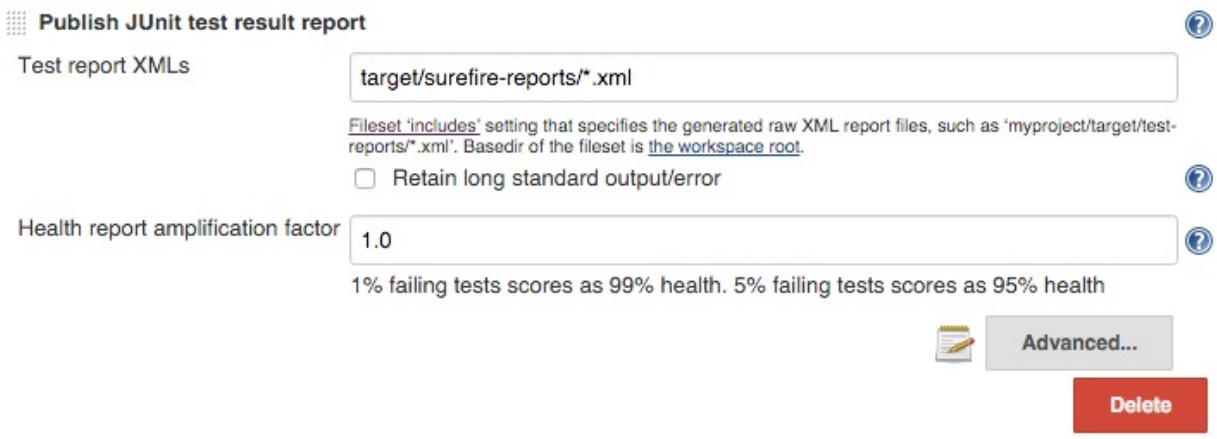


JUnit report

設置 task output report

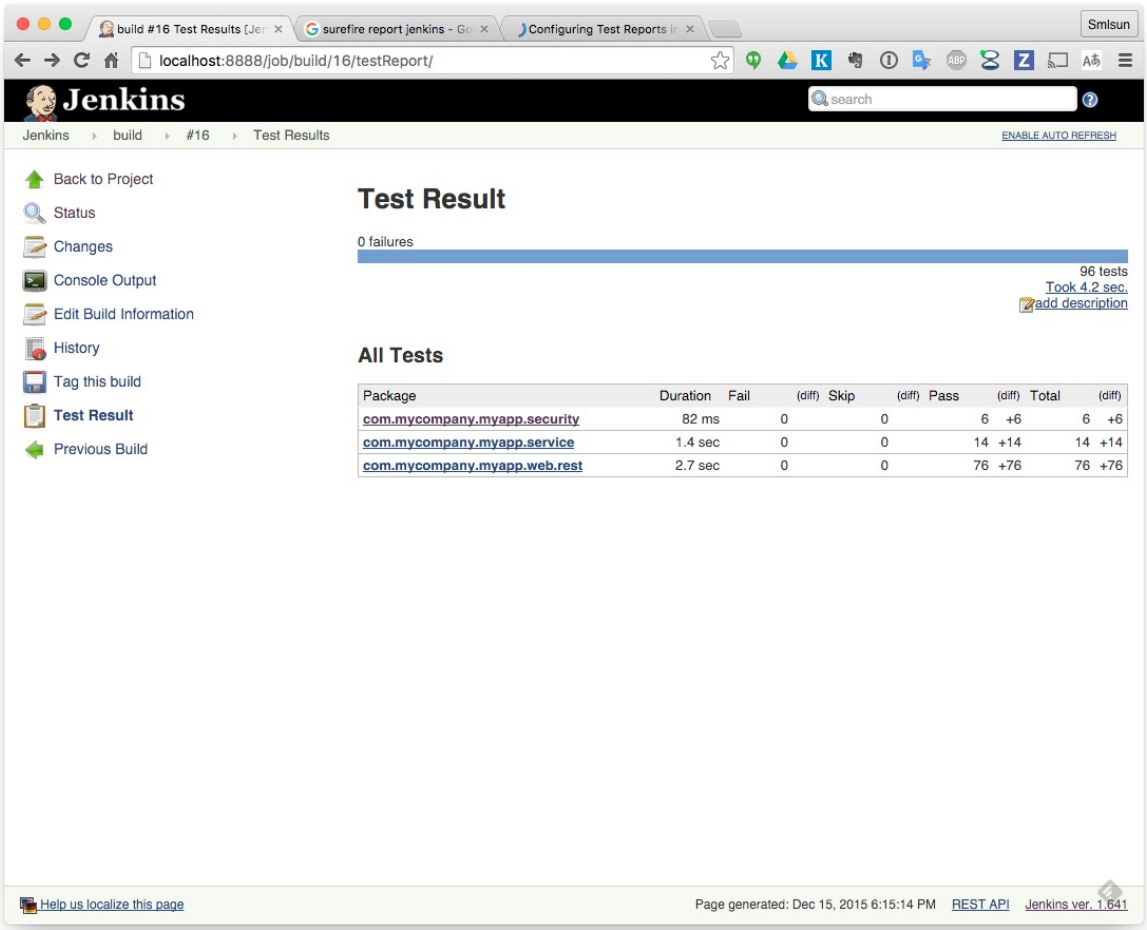


設定讀取 report 位置



如上圖所示，將 Test report XMLs 設置為 `target/surefire-reports/*.xml`

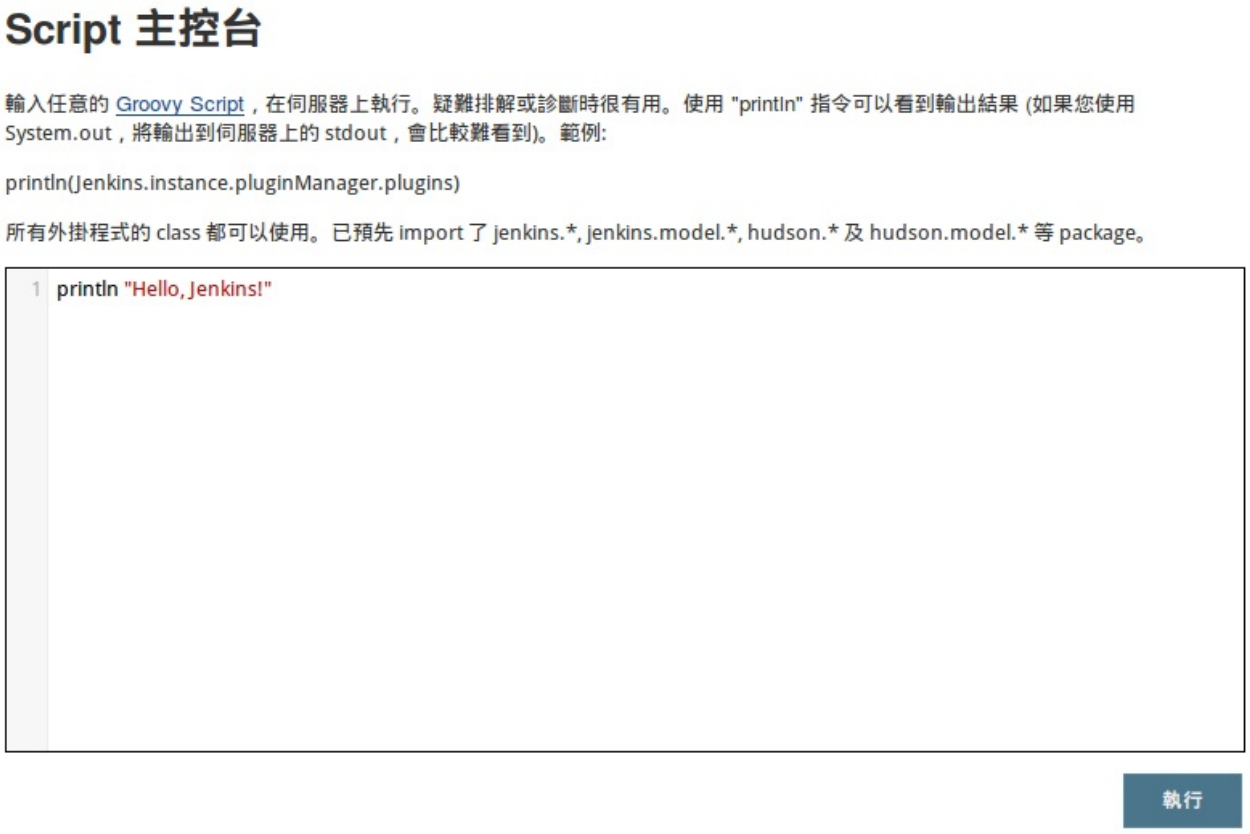
檢視 test report



Task 實作以 Java 為例

Jenkins 提供「Script Console」讓開發者容易撰寫例行性維護的小程式。

開啟方式：管理 Jenkins > Script 主控台



Script Console 支援 Groovy 程式語言，一個最簡單的 Hello World 範例：

```
println "Hello, Jenkins!"
```

按下「執行」就可以看到結果。

範例：利用 Script 檢視（維護）所有 Job 的參數。

```
Hudson.instance.items.each {
  item ->

  prop = item.getProperty(ParametersDefinitionProperty.class)
```

```
    if (prop != null) {
      println "--- ${item.name} ---"

      prop.parameterDefinitions.each {
        println "${it.name} ${it.defaultValue} \"${it.description}\""
      }
    }
  }
```

Script 可以配合 crontab 等機制定期執行，例如每日（daily）或每小時（hourly），利用 command-line 來執行 Script 的方法如下：

```
curl --data-urlencode "script=$(<script1.groovy)" http://jenkins.local/scriptText
```

備註：必須先將 Script 程式碼儲存為 `script1.groovy`

Script 主要用於 Job / Task 的維護，例如我們想要一次更新數十甚至數百的 Job 的特定參數內容，就可以撰寫 Script 來取代瑣碎的人工作業。

範例：利用 Script 建立新的作業（Job）。

```
Hudson.instance.createProject(FreeStyleProject.class, "free-style-project")
```

Jenkins Script 更多功能可參考 API 文件。

- http://javadoc.jenkins-ci.org/hudson/model/Hudson.html

plugin

plugin 是在使用 jenkins 時很重要的一環，透過 plugin 可以簡化一些設置跟操作，本章節將介紹一些好用的 plugin

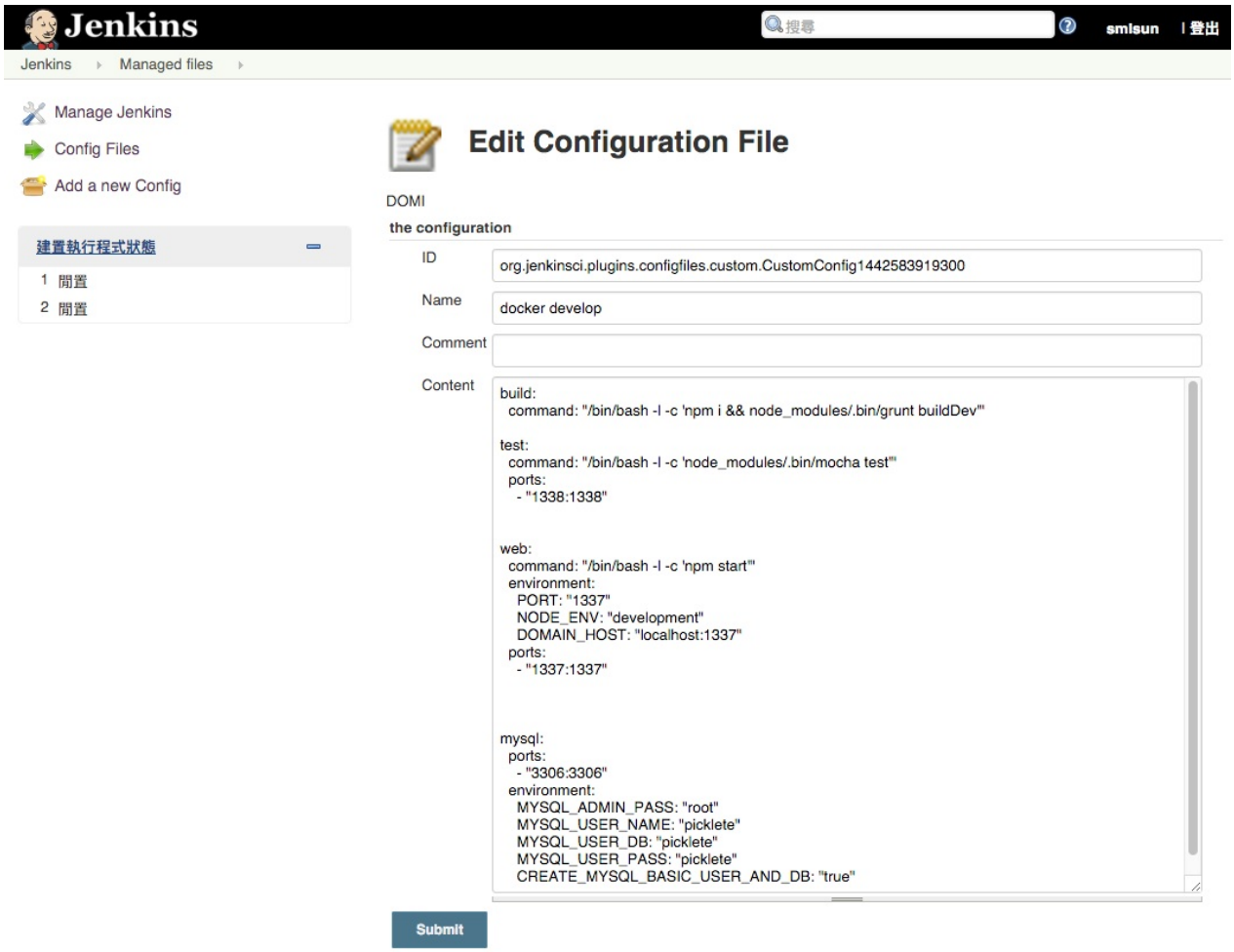
Config File Provider

使用時機

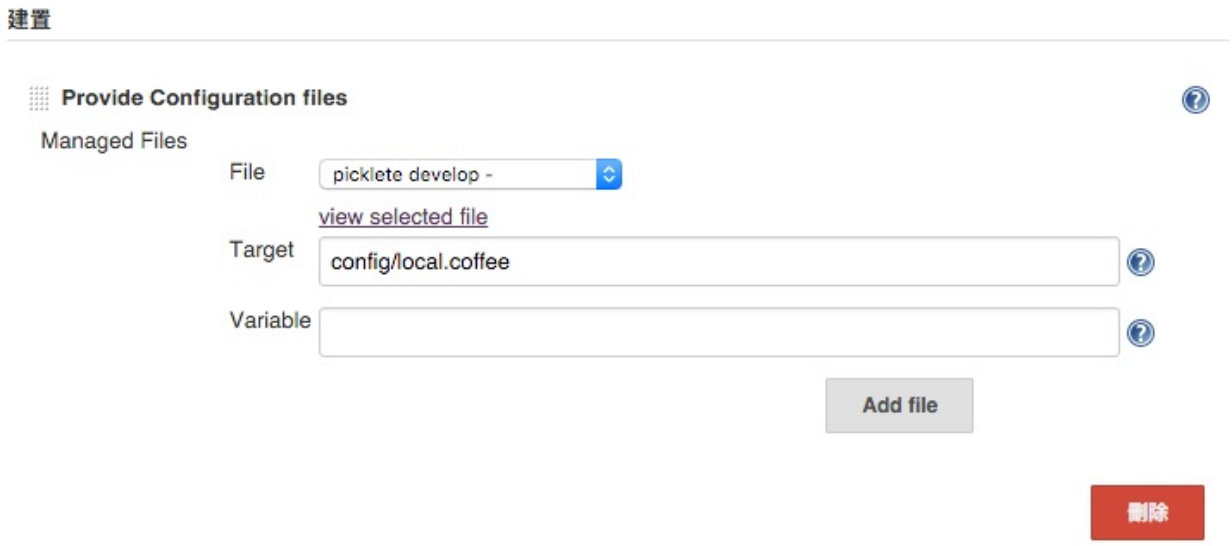
1. 有一些服務的 token 不想要被控管
2. 因應不同測試條件進行設定檔定義
3. 切換不同運行環境

使用方式

建立 config file

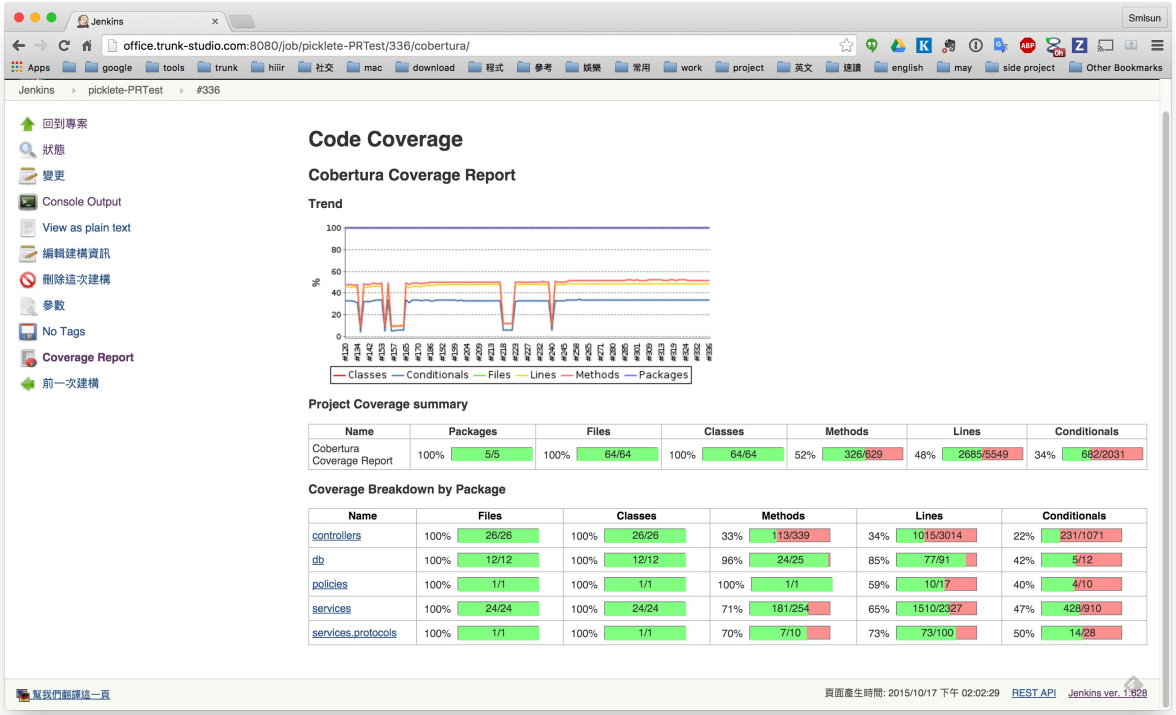


在 task 使用 config file


建置
Provide Configuration files
Managed Files
File
picklete develop -
view selected file
Target
config/local.coffee
Variable
Add file
刪除

Cobertura

程式覆蓋率



task 設置畫面

picklete-PRTest Config [Je

office.trunk-studio.com:8080/job/picklete-PRTest/configure

Jenkins › picklete-PRTest › 組態設定

建置後動作

Publish Cobertura Coverage Report

Cobertura xml report pattern: **/coverage/cobertura-coverage.xml

This is a file name pattern that can be used to locate the cobertura xml report files (for example with Maven2 use **/target/site/cobertura/coverage.xml). The path is relative to the module root unless you have configured your SCM with multiple modules, in which case it is relative to the workspace root. Note that the module root is SCM-specific, and may not be the same as the workspace root.
Cobertura must be configured to generate XML reports for this plugin to function.

Consider only stable builds

Include only stable builds, i.e. exclude unstable and failed ones.

Fail builds if no reports

fail builds if No coverage reports are found.

Fail unhealthy builds

Unhealthy projects will be failed.

Fail unstable builds

Unstable projects will be failed.

Health auto update

Auto update threshold for health on successful build.

Stability auto update

Auto update threshold for stability on successful build.

Zoom coverage chart

Zoom the coverage chart and crop area below the minimum and above the maximum coverage of the past reports.

Maximum number of builds: 0

Only graph the most recent N builds in the coverage chart, 0 disables the limit.

Source Encoding: ASCII

Encoding when showing files.

Coverage Metric Targets

Methods	刪除	40.0	20.0	30.0
Lines	刪除	40.0	20.0	30.0
Conditionals	刪除	40.0	20.0	30.0

新增

Configure health reporting thresholds.
For the row, leave blank to use the default value (i.e. 80).
For the and rows, leave blank to use the default values (i.e. 0).

刪除

新增建置後動作

儲存 套用

使用指令進行 plugin 安裝

下面連結可以看到目前官方所有的 plugins

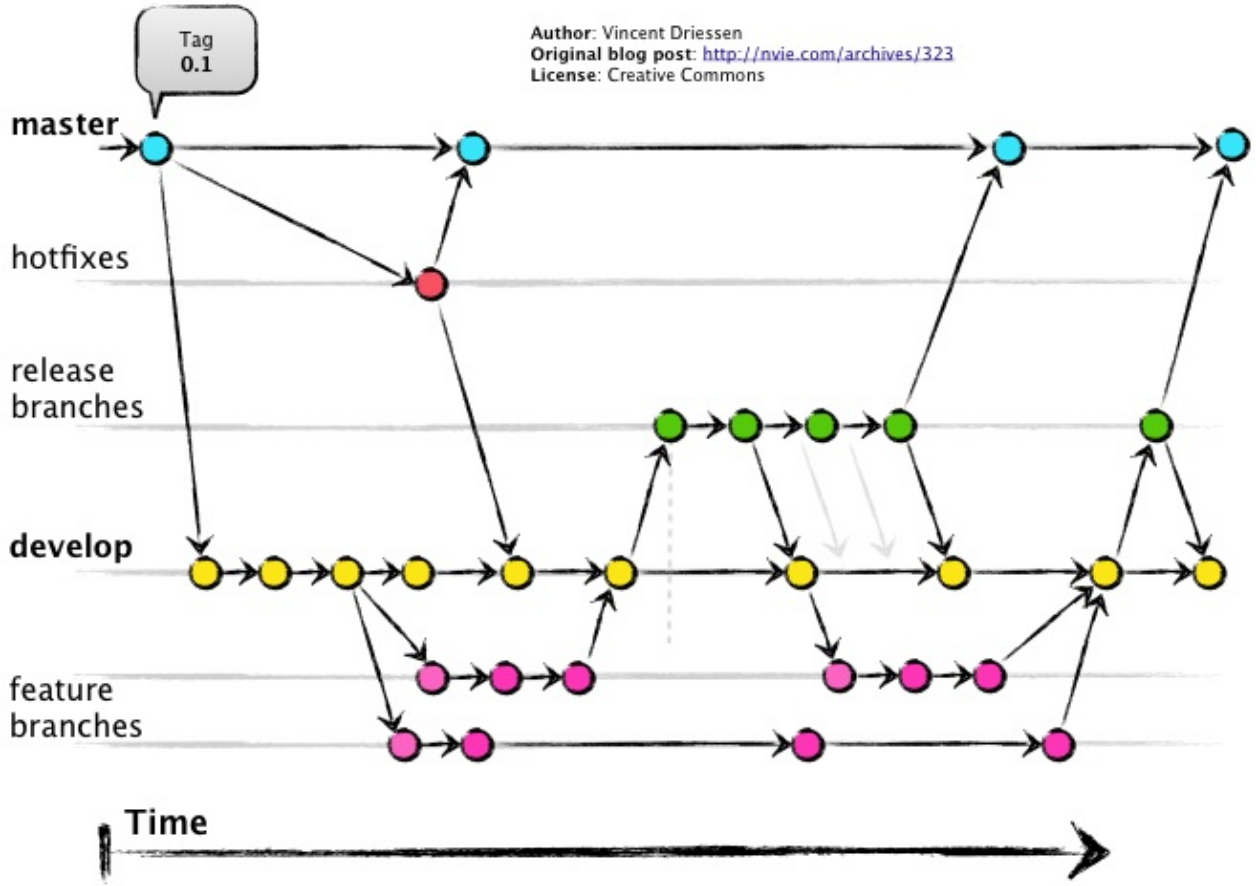
http://updates.jenkins-ci.org/download/plugins/

我們可以透過下列指令進行套件安裝

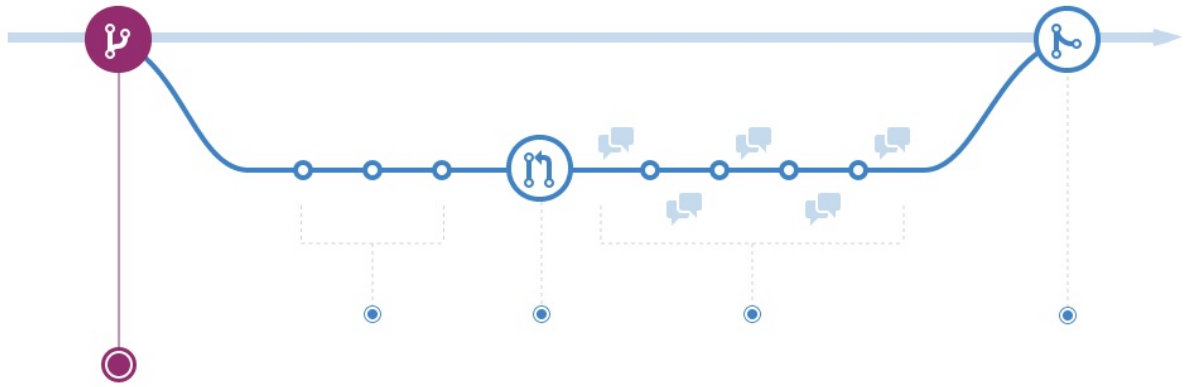
```
cd $JENKINS_HOME/plugins
curl -O http://updates.jenkins-ci.org/latest/plugins/cobertura.hpi
curl -O http://updates.jenkins-ci.org/latest/plugins/publish-over-ssh.hpi
curl -O http://updates.jenkins-ci.org/latest/plugins/git.hpi
curl -O http://updates.jenkins-ci.org/latest/plugins/config-file-provider.hpi
```

專案開發各個階段如何搭配 CI

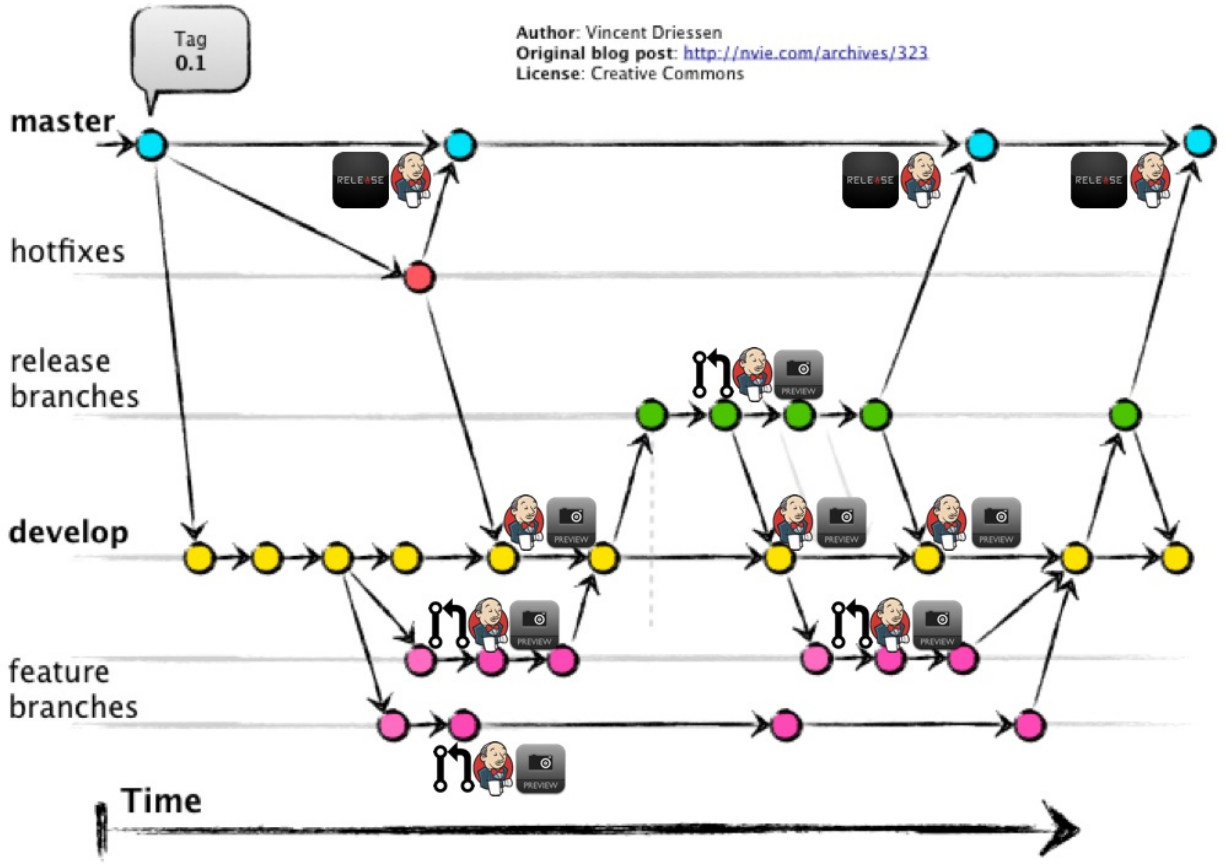
Git-Flow



GitHub Flow



Git-Flow with GitHub Flow and Jenkins



Build 自動建置

使用時機

1. Daily Build
2. 週期性建置
3. 進行正式機部署時事先建置

Task 實作

先設置 git Repository 使用：

```
https://github.com/agileworks-tw/spring-boot-sample
```

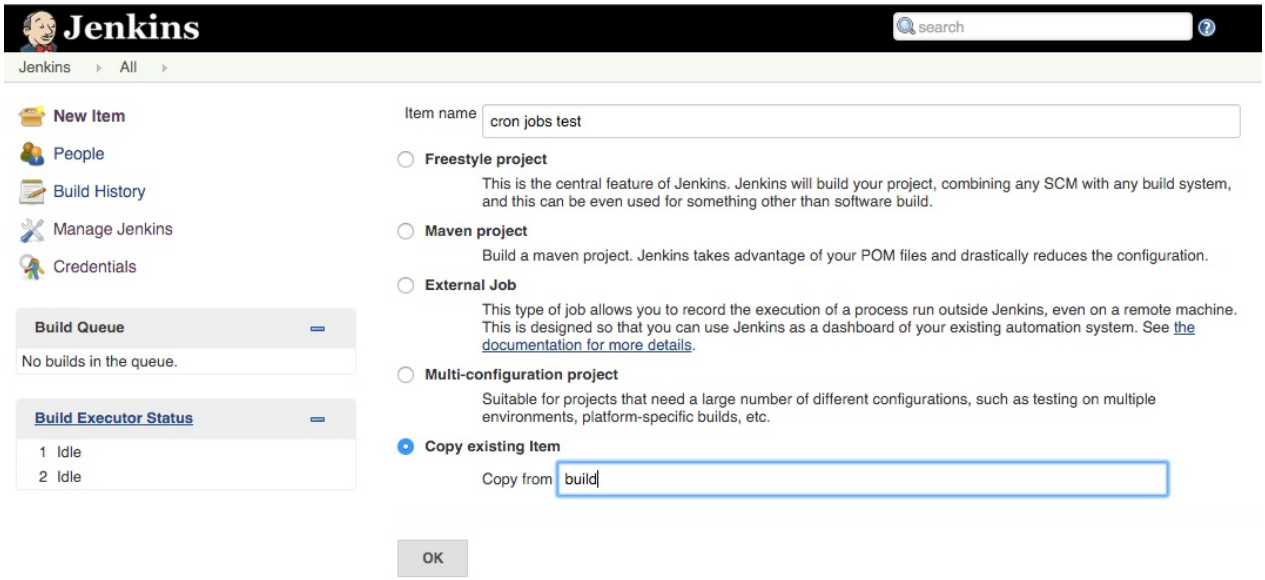
Shell 指令

```
#!/bin/bash
mvn package
```

test

延續 task/build

我們可以建置透過 build task 直接複製如下圖：



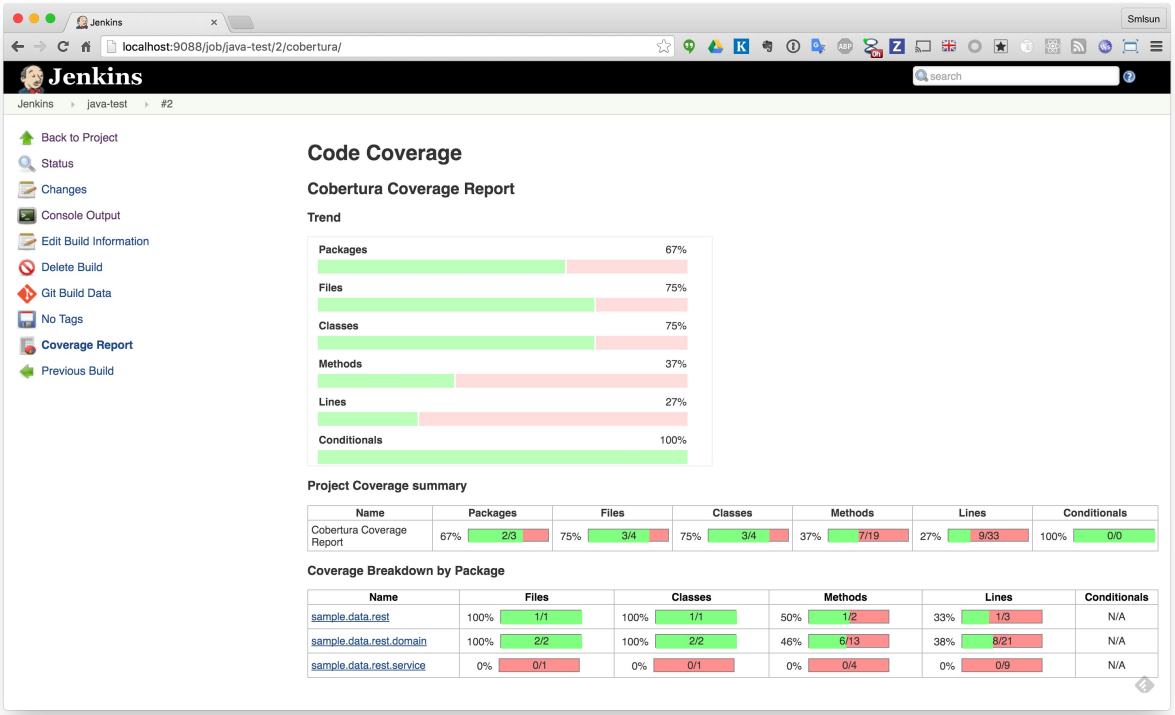
建置指令

```
#!/bin/bash
mvn clean cobertura:cobertura test
```

設置 Cobertura 呈現測試覆蓋率

report 位置：`target/site/cobertura/coverage.xml`

執行結果


Jenkins
localhost:9088/job/java-test/2/cobertura/
Jenkins
Jenkins › java-test › #2
Back to Project
Status
Changes
Console Output
Edit Build Information
Delete Build
Git Build Data
No Tags
Coverage Report
Previous Build
Code Coverage
Cobertura Coverage Report
Trend
Packages 67%
Files 75%
Classes 75%
Methods 37%
Lines 27%
Conditionals 100%
Project Coverage summary
Name | Packages | Files | Classes | Methods | Lines | Conditionals
Cobertura Coverage Report | 67% 2/3 | 75% 3/4 | 75% 3/4 | 37% 7/19 | 27% 9/33 | 100% 0/0
Coverage Breakdown by Package
Name | Files | Classes | Methods | Lines | Conditionals
sample.data.rest | 100% 1/1 | 100% 1/1 | 50% 1/2 | 33% 1/3 | N/A
sample.data.rest.domain | 100% 2/2 | 100% 2/2 | 46% 6/13 | 38% 8/21 | N/A
sample.data.rest.service | 0% 0/1 | 0% 0/1 | 0% 0/4 | 0% 0/9 | N/A

preview

建置指令

參考 task/build

透過 ssh 進行 preview

ssh publish setup

- Name: 選擇在 publish-over-ssh 建置的 ssh server
- Source files: target/spring-boot-sample-data-rest-0.1.0.jar
- Remote directory: deploy/preview

preview 執行指令

```
cd deploy/preview

kill `cat run.pid` || true
kill `cat ../release/run.pid` || true

java -jar target/spring-boot-sample-data-rest-0.1.0.jar > /dev/null 2>&1 & echo $! >
```

如此就可以很快速的將最新的程式碼啟動進行功能驗證

> 之所以這樣處理，因為 preview 跟 production release 非常像，下一個章節將可以直接複製修正

正式機部署

task 設置

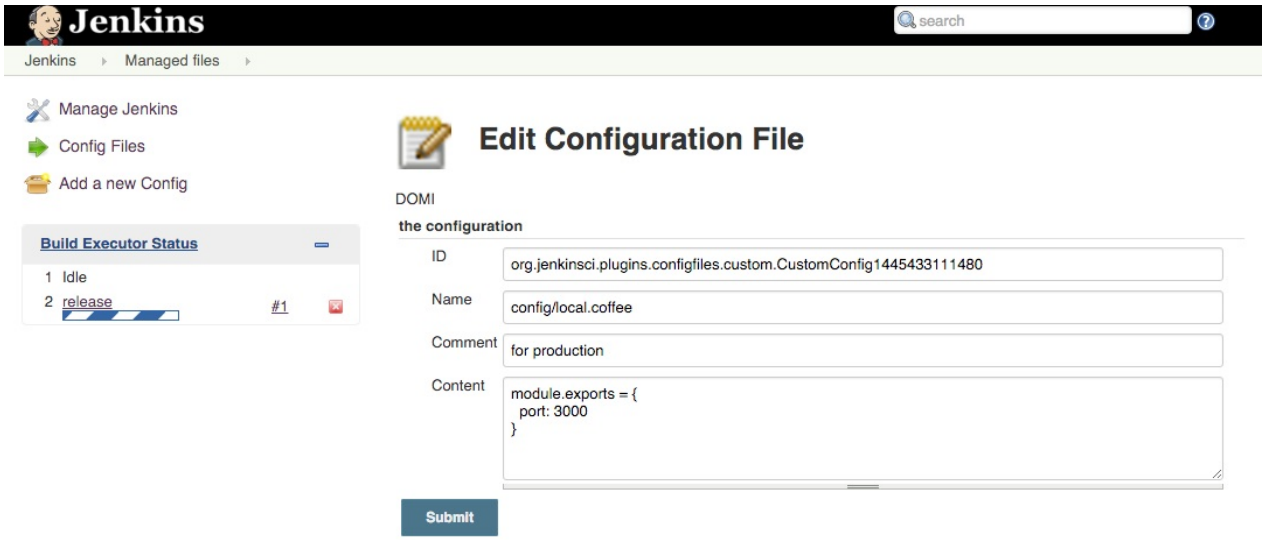
基本設置與 task/preview 基本上是一樣的

因為是 production 環境，所以我們需要將設定改為 production 機器專用

替換 Config

使用 config-file-provider

設置畫面如下：



假設 production 機器啟動後 port 為 8000

我們需要新增設定為 `application.properties`

並且在建置時替換建置專案底下之 `src/main/resources/application.properties`

令 production 相關設置生效。

ssh publish setup

假設 preview 機器就是 production 機器

- Name: 選擇在 publish-over-ssh 建置的 ssh server
- Source files: target/spring-boot-sample-data-rest-0.1.0.jar
- Remote directory: deploy/release

production release 執行指令

基本指令跟 preview 很像，資料夾是不一樣的

```
cd deploy/release

kill `cat run.pid` || true
kill `cat ../preview/run.pid` || true

java -jar target/spring-boot-sample-data-rest-0.1.0.jar > /dev/null 2>&1 & echo $! >
```

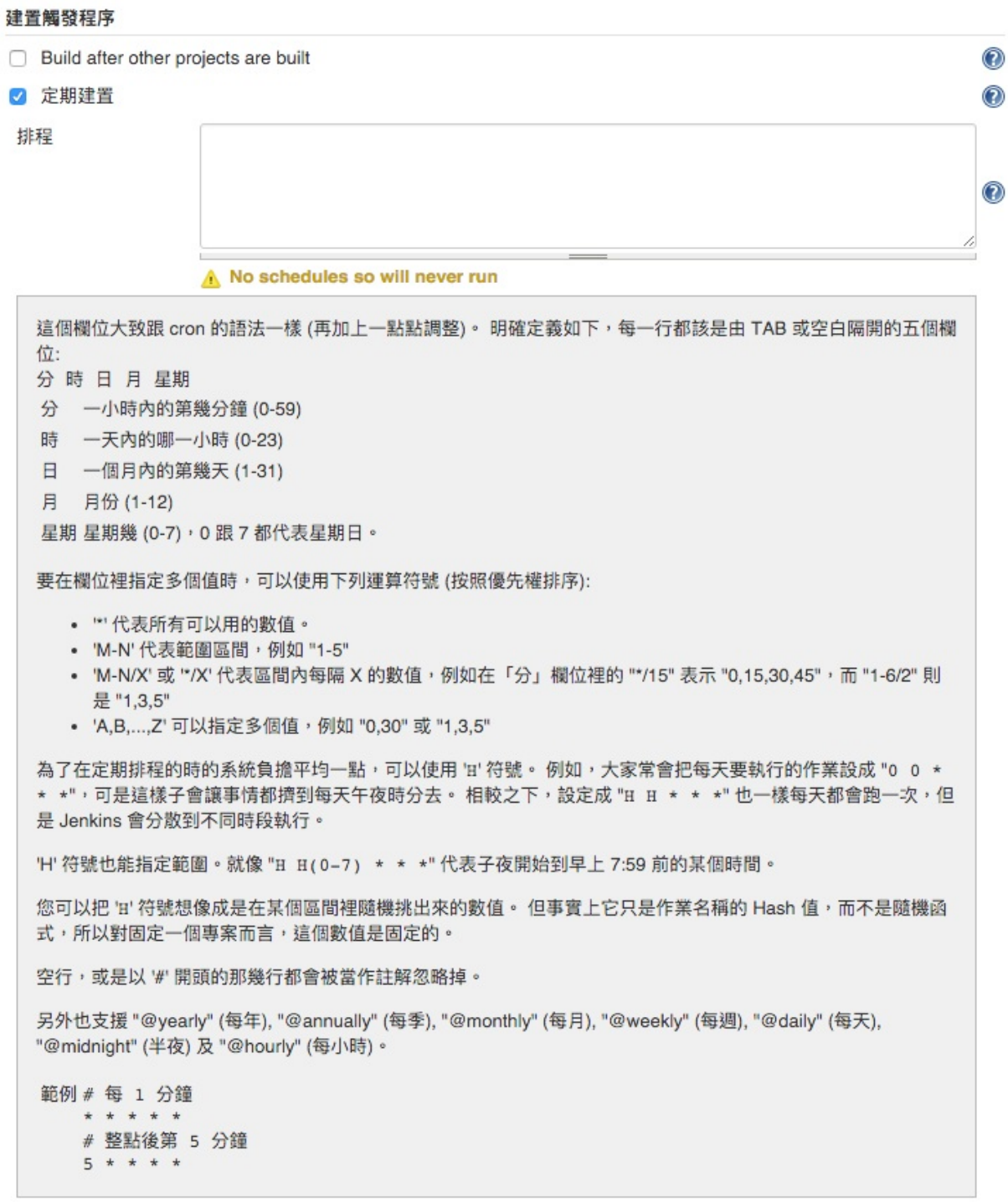
透過這些 task 的設置就把整個開發流程自動化完成。

定時執行測試

可以複製 task/test 來進行

我們可以設置每 5 分鐘執行一次，指令如下：

```
H/5 * * * *
```



test 成功自動進行 preview

test 成功自動進行 release 也是非常簡單，請自由發揮，不過一般來說 production release 會是比較嚴謹控管，建議由人員控制時機點

jenkins 2.0

課程難度：進階

Jenkins 2.0 版現在已經發佈。該版本增加了一種新的定義構建流程的方式，改善了新用戶的體驗。

Jenkins workflow plugin 並不是什麼新東西，但是2.0版本將提供一種新的核心特性（名字就叫 Pipeline），該特性允許用戶在 DSL（Domain Specific Language）的幫助下定義他們的構建流程。DSL 是 Groovy 其中之一的特性，允許把「建構設定(build definitioin)」當作一般程式碼撰寫。因此，建構設定就不只是定義在 Jenkins 的 task，而是可以被版本控制。

其實在此之前，有一些其他 Plugin 用來解決同樣的問題，但是 Jenkins 的創建者 Kohsuke Kawaguchi 對 InfoQ 說 Pipeline 不只是一款 Plugin。

實際上 Pipeline 是由一系列 Plugin 組成的一種意義重大的子系統。關於 2.0 版，我們其中一部分想法是想讓用戶拋棄「從內核開始然後安裝插件」的心態。取而代之的是，當你拿到 Jenkins 2.0 版的時候，你就會得到我們認為能覆蓋 80% 使用場景的功能，其中有些來自內核，有些來自 Plugin。 把管道功能作為核心特性也是業界中其他同行的做法。

紐約市的一名軟體工程師 Jacques Chester 認為，在關注 Jenkins Pipeline 功能的開發者也應該瞭解一下 Concourse。在一次 InfoQ 的採訪中，Chester 說Jenkins 的 Pipeline Plugin 系統其實是為其固有缺陷上的 Patch。

該系統將「通過 Plugin 而實現 Concourse 從設計之初就具有的部分功能從而重塑 Jenkins。Concourse 的基因是可以對整合流程進行版本控制，用完即可丟棄的構建環境以及可以將任務計劃委派給工具而不是手動組裝」。

另外，2.0 版的「Getting Started」有了一種新的體驗，在安裝插件方面為用戶提供一些建議。

課程內容

此章節將介紹 Jenkins 2.0 的新特性，以及在前面章節關於 Docker 的應用進行整合，透過 Pipeline 進行 CI 建置。

資料來源

http://www.infoq.com/cn/news/2016/04/jenkins-2-beta-pipeline-system

Pipeline 使用

Pipeline 是以 Groovy Script 作為撰寫，為 Java 的動態語言。

第一個 Pipeline: Hello World

```
echo 'hello from Pipeline'
```

執行結果

```
Started by user anonymous
[Pipeline] echo
hello from Pipeline
[Pipeline] End of Pipeline
Finished: SUCCESS
```

以上面的例子來說，完整的語法為：

```
echo("hello from Pipeline");
```

我們可以省略 `;`, `()` 並且 `''`, `""` 皆可以使用。

建立簡單的 Pipeline

```
node {
  git url: 'https://github.com/agileworks-tw/spring-boot-sample.git'
  def mvnHome = tool 'M3'
  sh "${mvnHome}/bin/mvn test"
}
```

其中

```
git url: 'https://github.com/agileworks-tw/spring-boot-sample.git'
```

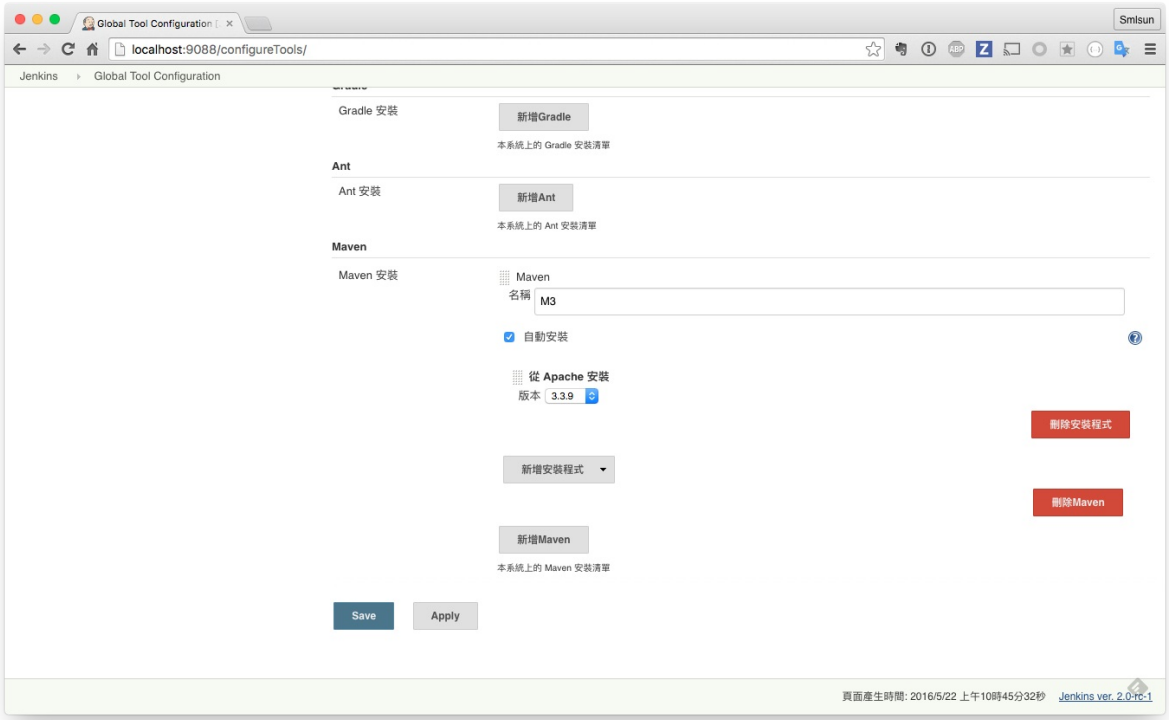
也可以指定 branch，指令如下

```
git url: 'https://github.com/agileworks-tw/spring-boot-sample.git', branch: 'master'
```

若沒有指定的話，預設為 Master，完整程式碼如下

```
git([url: 'https://github.com/agileworks-tw/spring-boot-sample.git', branch: 'master
```

tool 實際上是讀取 jenkins 中的 `Global Tool Configuration` 所設定的建置工具，如下圖：




```
def mvnHome = tool 'M3'
```

就是取得該 tool 定義的結果來進行專案建置。

而下面指令

```
sh "${mvnHome}/bin/mvn test"
```

則為使用該 tool 利用 mvn 進行專案 test，而實際指令為

```
sh([script: mvnHome + '/bin/mvn test'])
```

可以看到簡化不少

for windows 可以改為

```
bat "${mvnHome}\\bin\\mvn test"
```

管理環境變數

在 Pipeline 我們可以使用 env 來存取 jenkins 所設置的環境變數，我們也可以使用 env 來設置新的環境變數。

延續上一個範例：

```
node {
  git url: 'https://github.com/agileworks-tw/spring-boot-sample.git'
  def mvnHome = tool 'M3'
  env.PATH = "${mvnHome}/bin:${env.PATH}"
  sh 'mvn test'
}
```

透過 env 的使用，我們就不需要使用 `${mvnHome}/bin/mvn` 這麼長的程式路徑。

上述的處理方式，只有當你使用但一 slave 時比較沒有問題，有時候我們需要指定某項建置需要特定的環境變數時，我們可以改為

```
node {
  git url: 'https://github.com/agileworks-tw/spring-boot-sample.git'
  withEnv(["PATH+MAVEN=${tool 'M3'}/bin"]) {
    sh 'mvn test'
  }
```

```
  }
```

`withEnv(["PATH+MAVEN=${tool 'M3'}/bin"]) {` 得意思就是原有的 `PATH` 變數，加上新的 `Maven` 環境變數

如此就可以限制該環境變數限制在特定的 scope 一但執行完成，環境變數就會還原成原本的內容。

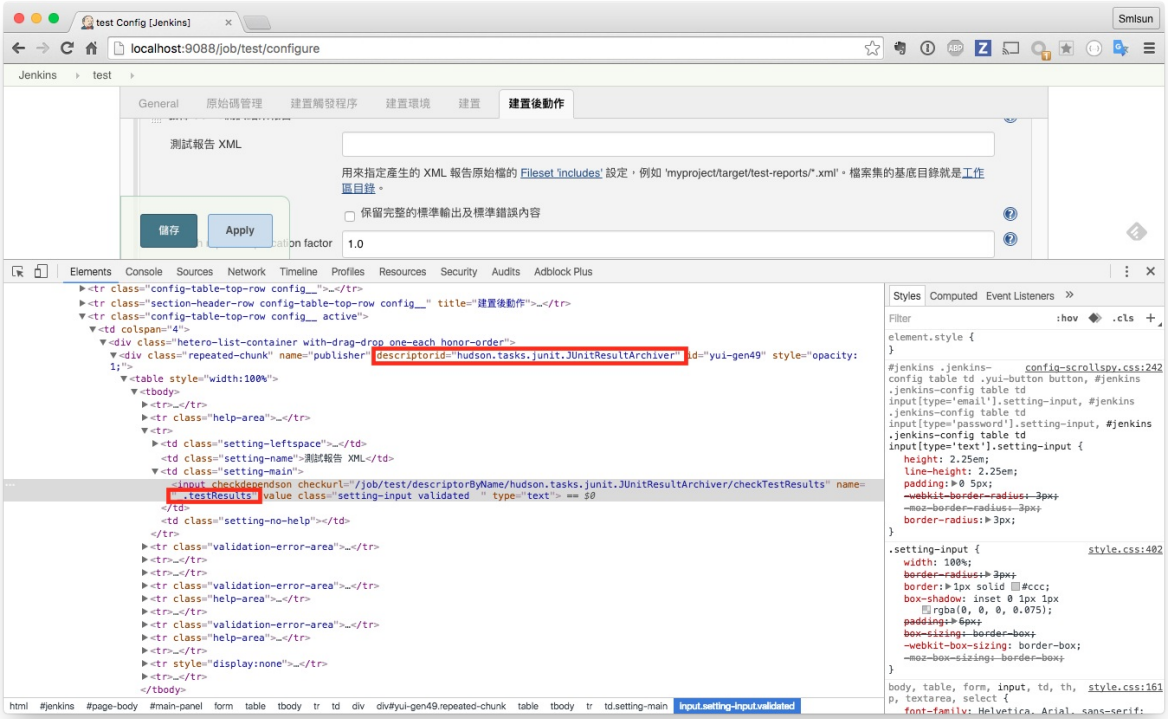
除此之外，一些 Jnekins 本來在建置過程中就會可用的變數也可以透過 env 直接存取，比如 `env.BUILD_TAG` 可在 groovy script 中使用，或是在 `sh` 時使用 `$BUILD_TAG` 進行該變數存取。

紀錄測試結果，以及建置封存

```
node {
  git url: 'https://github.com/agileworks-tw/spring-boot-sample.git'
  def mvnHome = tool 'M3'
  sh "${mvnHome}/bin/mvn test package"
  step([$class: 'ArtifactArchiver', artifacts: '**/target/*.jar', fingerprint: true])
  step([$class: 'JUnitResultArchiver', testResults: '**/target/surefire-reports/TEST-
}
```

其中 `$class` 為縮寫，以 JUnitResultArchiver 為例，完整路徑為 `hudson.tasks.junit.JUnitResultArchiver`

可以從 jenkins UI 找到一些線索，如下圖：



指定特定 Slave 進行建置

假設已經有設置 jenkins slave 節點

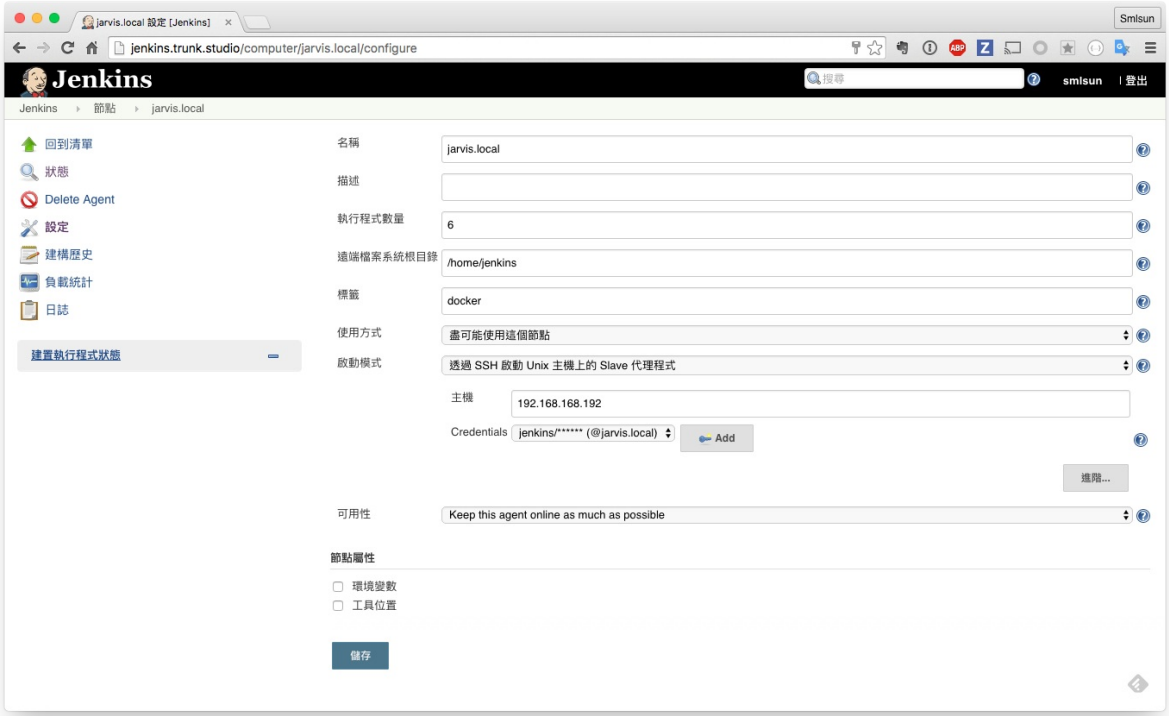
```
node('master') {
    // as before
}
```

我們可以透過上述方式指定特定的 node 進行建置，若沒有指定將由 jenkins 自行判斷

除了節點的 name，也可以指定特定的 label，如

```
node('unix && 64bit') {
    // as before
}
```

定義方式可參考下圖



若點選建置，你將可以看到下列訊息輸出

```
Running on <yourslavename> in /<slaveroot>/workspace/<jobname>
```

中途暫停

```
node('remote') {
  input 'Ready to go?'
  // rest as before
}
```

使用情境如：用於確認 QA 是否正確，若確認完成進行 production deploy，又或者輸入特定參數。

使用 try... catch... final

我們在進行建置時，就像在寫程式一樣，一定會有另外產生，同樣我們可以透過 `try... catch... final` 來定義，如：

```
try{
  stage 'Approve, go production'
  def url = 'http://localhost:8000/'
  input message: "Does staging at $url look good? ", ok: "Deploy to production"
}finally{
  sh "docker-compose stop server"
}
```

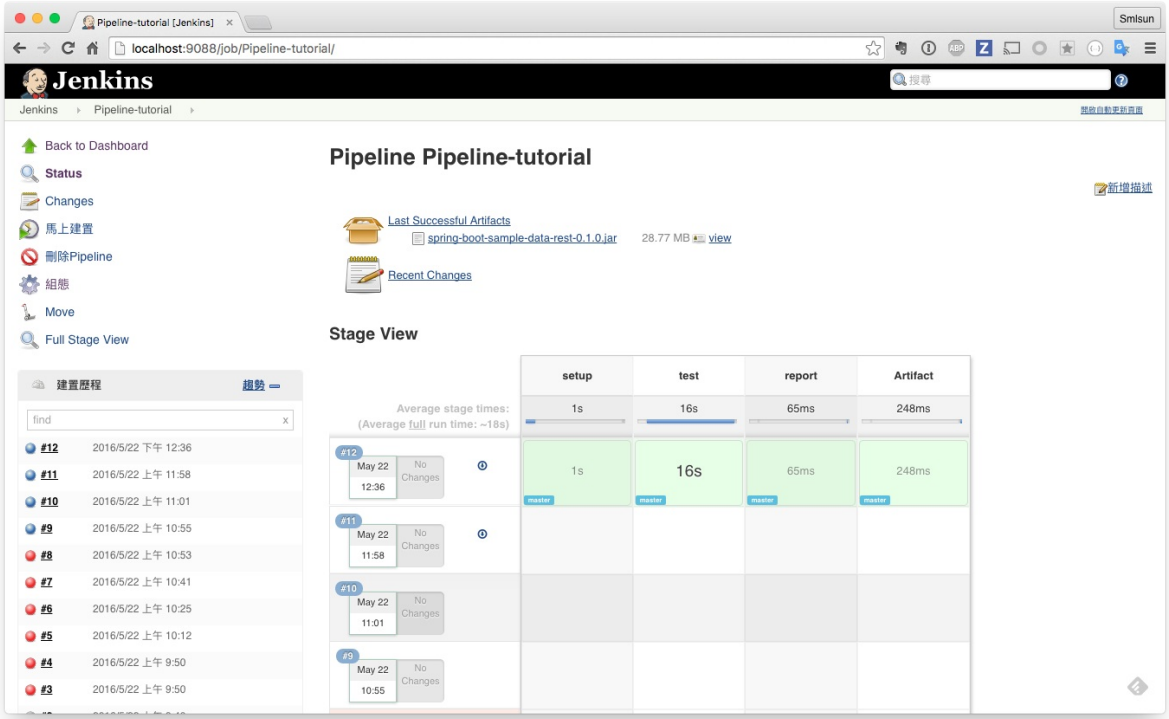
使用 docker 進行 QA 驗證，無論最後是否要要進行 production deploy 皆需要將 docker 停止。

stage 定義

stage 的定義，可以讓你的建置步驟更清楚，可讀性更好

```
node {
  stage 'setup'
  git url: 'https://github.com/agileworks-tw/spring-boot-sample.git'
  def mvnHome = tool 'M3'

  stage 'test'
  sh "${mvnHome}/bin/mvn test package"


  stage 'report'
  step([$class: 'JUnitResultArchiver', testResults: '**/target/surefire-reports/TEST-

  stage 'Artifact'
  step([$class: 'ArtifactArchiver', artifacts: '**/target/*.jar', fingerprint: true])

}
```

且 jenkins 在 task 中 stage view 進度上可以看到每個 stage 的執行過程與結果，如下圖：

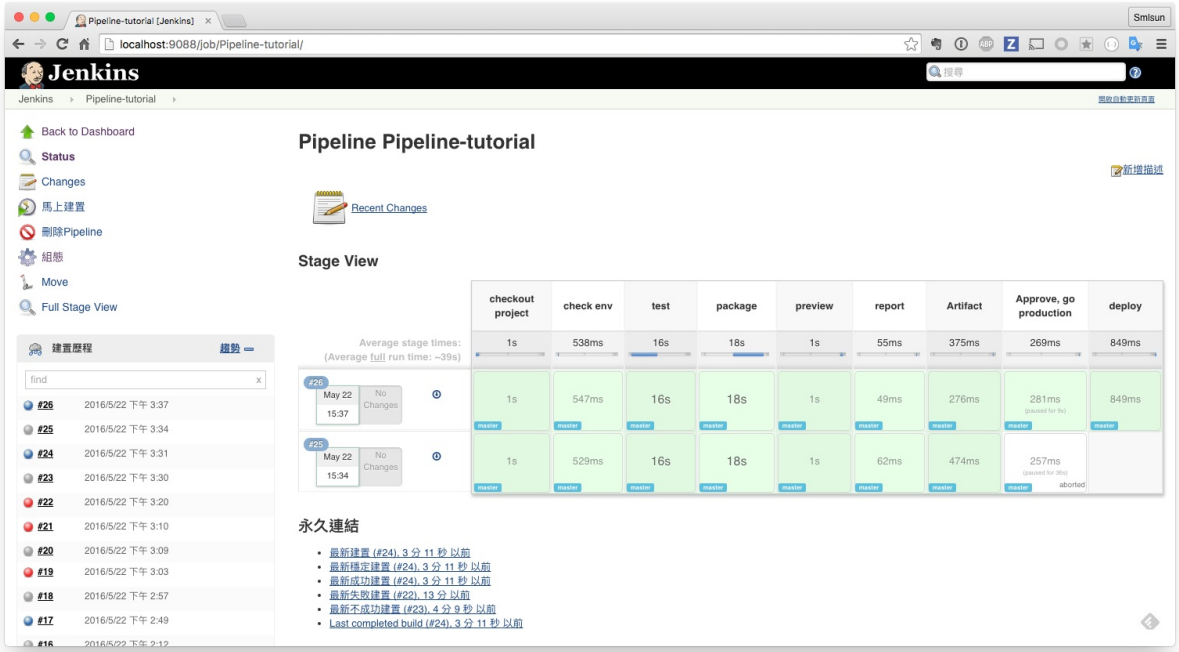

Pipeline-tutorial [Jenkins]
localhost:9088/job/Pipeline-tutorial/
Jenkins
Jenkins › Pipeline-tutorial
Back to Dashboard
Status
Changes
馬上建置
刪除Pipeline
組態
Move
Full Stage View
建置歷程
趨勢
#12 2016/5/22 下午 12:36
#11 2016/5/22 上午 11:58
#10 2016/5/22 上午 11:01
#9 2016/5/22 上午 10:55
#8 2016/5/22 上午 10:53
#7 2016/5/22 上午 10:41
#6 2016/5/22 上午 10:25
#5 2016/5/22 上午 10:12
#4 2016/5/22 上午 9:50
#3 2016/5/22 上午 9:50
Pipeline Pipeline-tutorial
新增描述
Last Successful Artifacts
spring-boot-sample-data-rest-0.1.0.jar 28.77 MB view
Recent Changes
Stage View
setup
test
report
Artifact
Average stage times:
(Average full run time: ~18s)
1s
16s
65ms
248ms
#12 May 22 12:36 No Changes
#11 May 22 11:58 No Changes
#10 May 22 11:01 No Changes
#9 May 22 10:55 No Changes

使用 Pipeline 進行建置

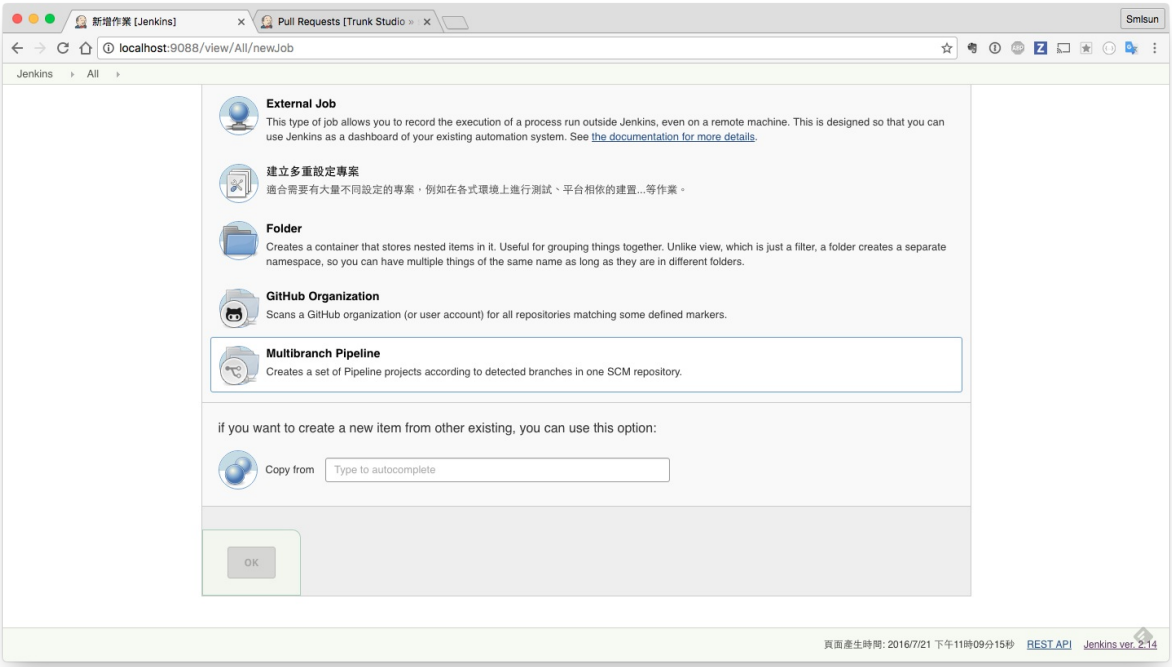
我們可以把前面章節所完成的相關 task 濃縮為單一個 Jenkinsfile

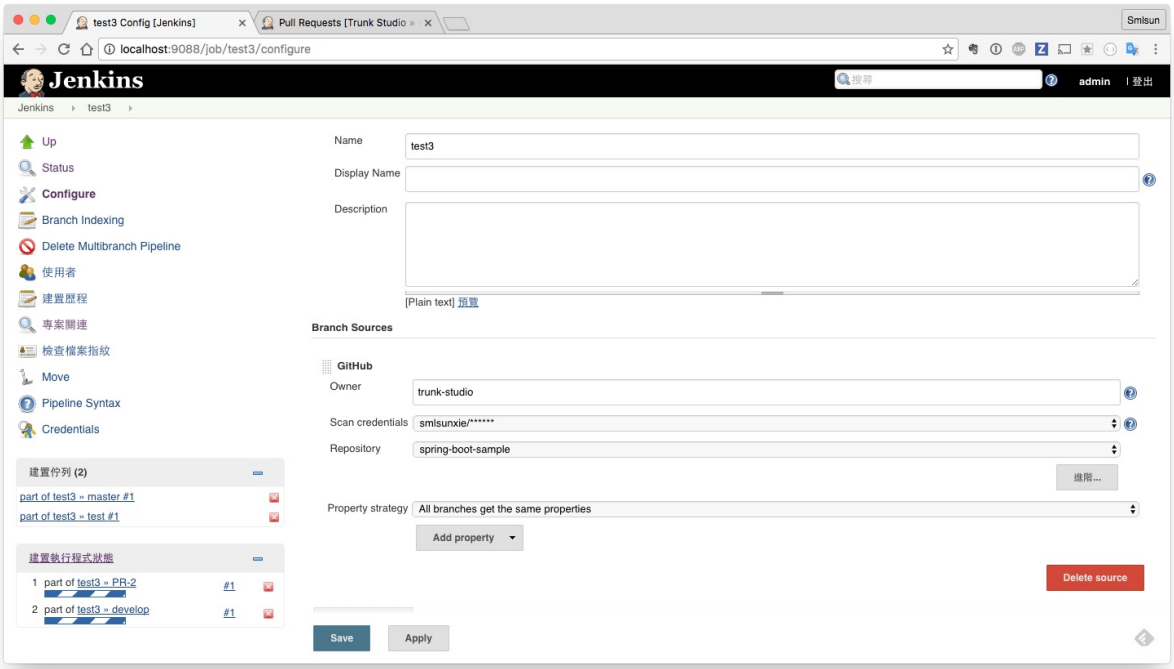
```
node {

  stage 'checkout project'
  git url: 'https://github.com/agileworks-tw/spring-boot-sample.git'

  stage 'check env'

  sh "mvn -v"
  sh "java -version"

  stage 'test'
  sh "mvn test"

  stage 'package'
  sh "mvn package"

  stage 'preview'
  sh 'make deploy-default'

  stage 'report'
  step([$class: 'JUnitResultArchiver', testResults: '**/target/surefire-reports/TEST-

  stage 'Artifact'
  step([$class: 'ArtifactArchiver', artifacts: '**/target/*.jar', fingerprint: true])

  try{
    stage 'Approve, go production'
    def url = 'http://localhost:8000/'
    input message: "Does staging at $url look good? ", ok: "Deploy to production"
  }finally{
    sh "ssh jenkins@localhost 'kill `cat deploy/release/run.pid`'"
  }

  stage 'deploy'
  sh 'make deploy-default'
}
```

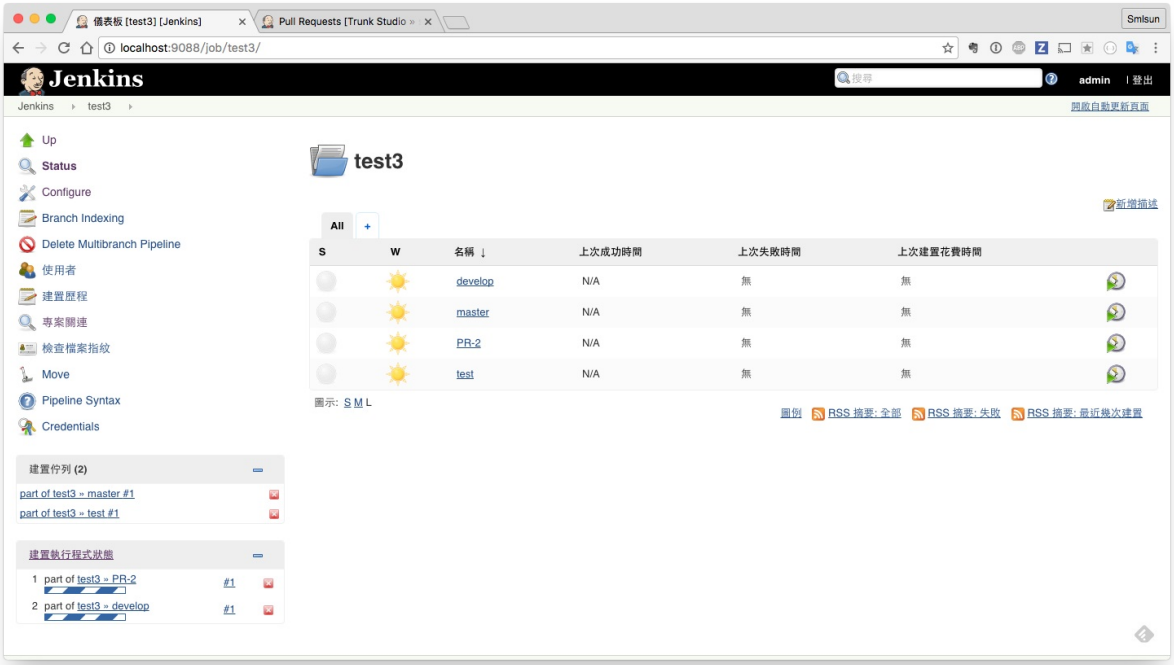
執行結果如下

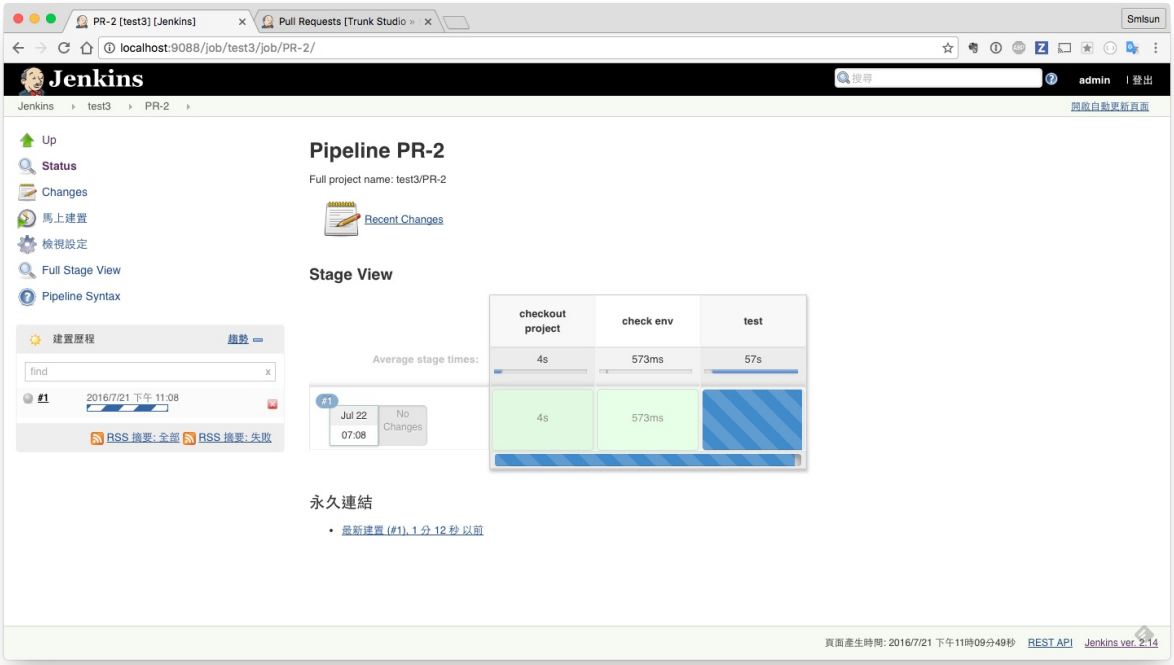

Pipeline-tutorial [Jenkins]
localhost:9088/job/Pipeline-tutorial/
Jenkins
Jenkins › Pipeline-tutorial
Back to Dashboard
Status
Changes
馬上建置
刪除Pipeline
組態
Move
Full Stage View
建置歷程
趨勢
#26 2016/5/22 下午 3:37
#25 2016/5/22 下午 3:34
#24 2016/5/22 下午 3:31
#23 2016/5/22 下午 3:30
#22 2016/5/22 下午 3:20
#21 2016/5/22 下午 3:10
#20 2016/5/22 下午 3:09
#19 2016/5/22 下午 3:03
#18 2016/5/22 下午 2:57
#17 2016/5/22 下午 2:49
Pipeline Pipeline-tutorial
新增描述
Recent Changes
Stage View
checkout project
check env
test
package
preview
report
Artifact
Approve, go production
deploy
Average stage times:
(Average full run time: ~39s)
1s
538ms
16s
18s
1s
55ms
375ms
269ms
849ms
#26 May 22 15:37 No Changes
1s
547ms
16s
18s
1s
49ms
276ms
281ms
849ms
#25 May 22 15:34 No Changes
1s
529ms
16s
18s
1s
62ms
474ms
257ms
aborted
永久連結
最新建置 (#24), 3 分 11 秒 以前
最新穩定建置 (#24), 3 分 11 秒 以前
最新成功建置 (#24), 3 分 11 秒 以前
最新失敗建置 (#22), 13 分 以前
最新不成功建置 (#23), 4 分 9 秒 以前
Last completed build (#24), 3 分 11 秒 以前

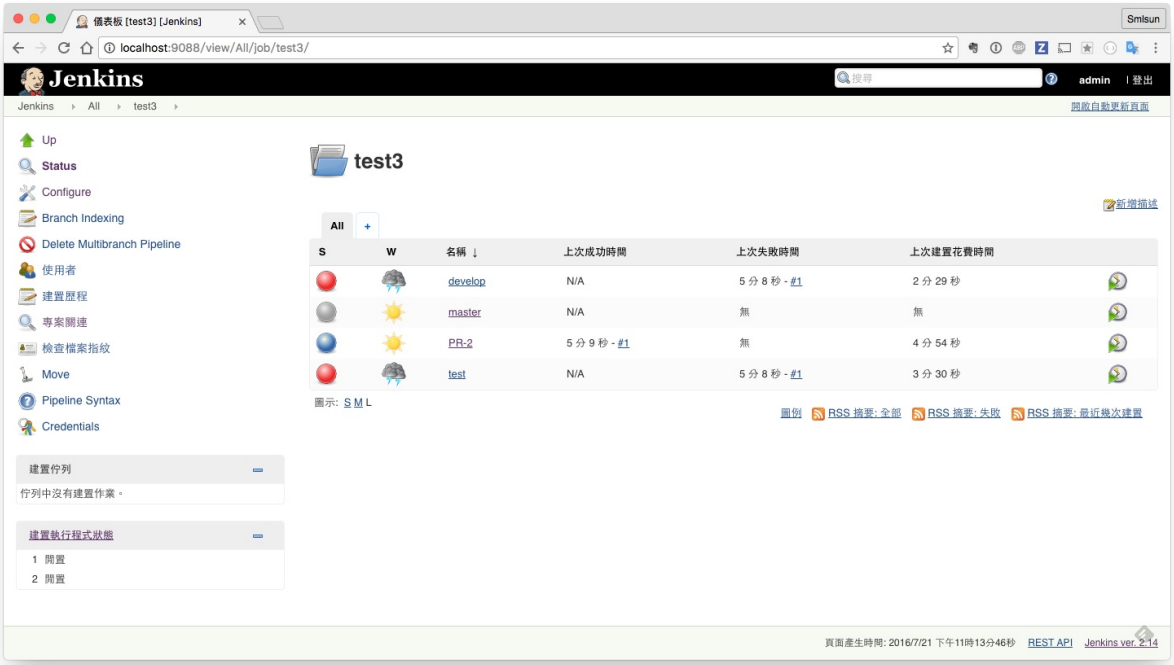
Multibranch Pipeline






儀表板 [test3] [Jenkins]
Pull Requests [Trunk Studio
localhost:9088/job/test3/
Jenkins
admin
登出
Jenkins
test3
開啟自動更新頁面
Up
Status
Configure
Branch Indexing
Delete Multibranch Pipeline
使用者
建置歷程
專案關連
檢查檔案指紋
Move
Pipeline Syntax
Credentials
test3
新增描述
All
S
W
名稱 ↓
上次成功時間
上次失敗時間
上次建置花費時間
develop
master
PR-2
test
N/A
無
圖示: S M L
圖例
RSS 摘要: 全部
RSS 摘要: 失敗
RSS 摘要: 最近幾次建置
建置佇列 (2)
part of test3 » master #1
part of test3 » test #1
建置執行程式狀態
1 part of test3 » PR-2 #1
2 part of test3 » develop #1


PR-2 [test3] [Jenkins]
Pull Requests [Trunk Studio
localhost:9088/job/test3/job/PR-2/
Jenkins
admin
登出
Jenkins
test3
PR-2
開啟自動更新頁面
Up
Status
Changes
馬上建置
檢視設定
Full Stage View
Pipeline Syntax
建置歷程
趨勢
find
#1
2016/7/21 下午 11:08
RSS 摘要: 全部
RSS 摘要: 失敗
Pipeline PR-2
Full project name: test3/PR-2
Recent Changes
Stage View
checkout project
check env
test
Average stage times:
4s
573ms
57s
#1
Jul 22
07:08
No Changes
4s
573ms
永久連結
最新建置 (#1), 1 分 12 秒 以前
REST API
Jenkins ver. 2.14


儀表板 [test3] [Jenkins]
localhost:9088/view/All/job/test3/
Jenkins
admin
登出
Jenkins › All › test3 ›
開啟自動更新頁面
Up
Status
Configure
Branch Indexing
Delete Multibranch Pipeline
使用者
建置歷程
專案關連
檢查檔案指紋
Move
Pipeline Syntax
Credentials
建置佇列
佇列中沒有建置作業。
建置執行程式狀態
1 閒置
2 閒置
test3
新增描述
All
S
W
名稱 ↓
上次成功時間
上次失敗時間
上次建置花費時間
develop
N/A
5 分 8 秒 - #1
2 分 29 秒
master
N/A
無
無
PR-2
5 分 9 秒 - #1
無
4 分 54 秒
test
N/A
5 分 8 秒 - #1
3 分 30 秒
圖示: S M L
圖例
RSS 摘要: 全部
RSS 摘要: 失敗
RSS 摘要: 最近幾次建置
REST API
Jenkins ver. 2.14

java doc

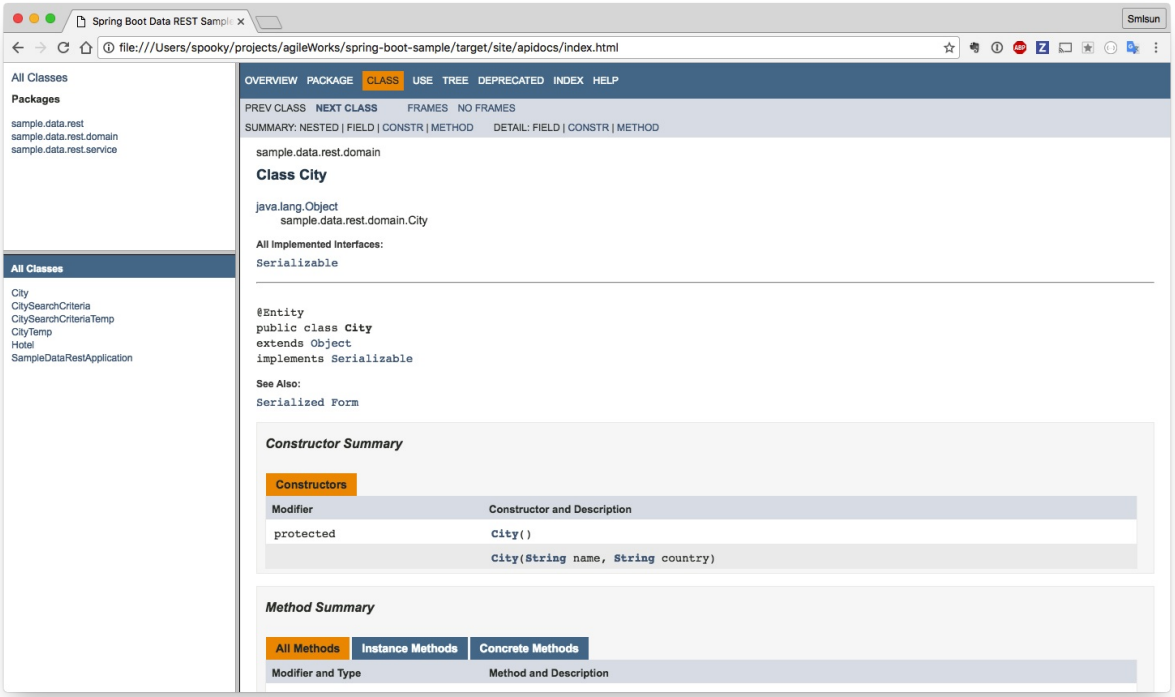
maven 產生 java doc 指令

`mvn javadoc:javadoc`

上述指令執行完成後，我們可以在下面路徑找到 html report

`${project_home}/target/site/apidocs/index.html`

相關呈現如下：



使用 jenkins 產出 javadoc

建置

執行 Shell

指令 mvn javadoc:javadoc

可以參考可用環境變數清單

新增建置步驟

建置後動作

發佈 Javadoc

Javadoc 目錄 target/site/apidocs/

相對於工作區根目錄的目錄，例如 'myproject/build/javadoc'

保留每次成功建置的 Javadoc

新增建置後動作

Jenkins

Jenkins ▸ java-sample-github ▸

回到儀表板
狀態
變更
工作目錄
馬上建置
刪除專案
組態
Move
Javadoc

專案 java-sample-github

Javadoc
工作區
最近變更

建置歷程 趨勢

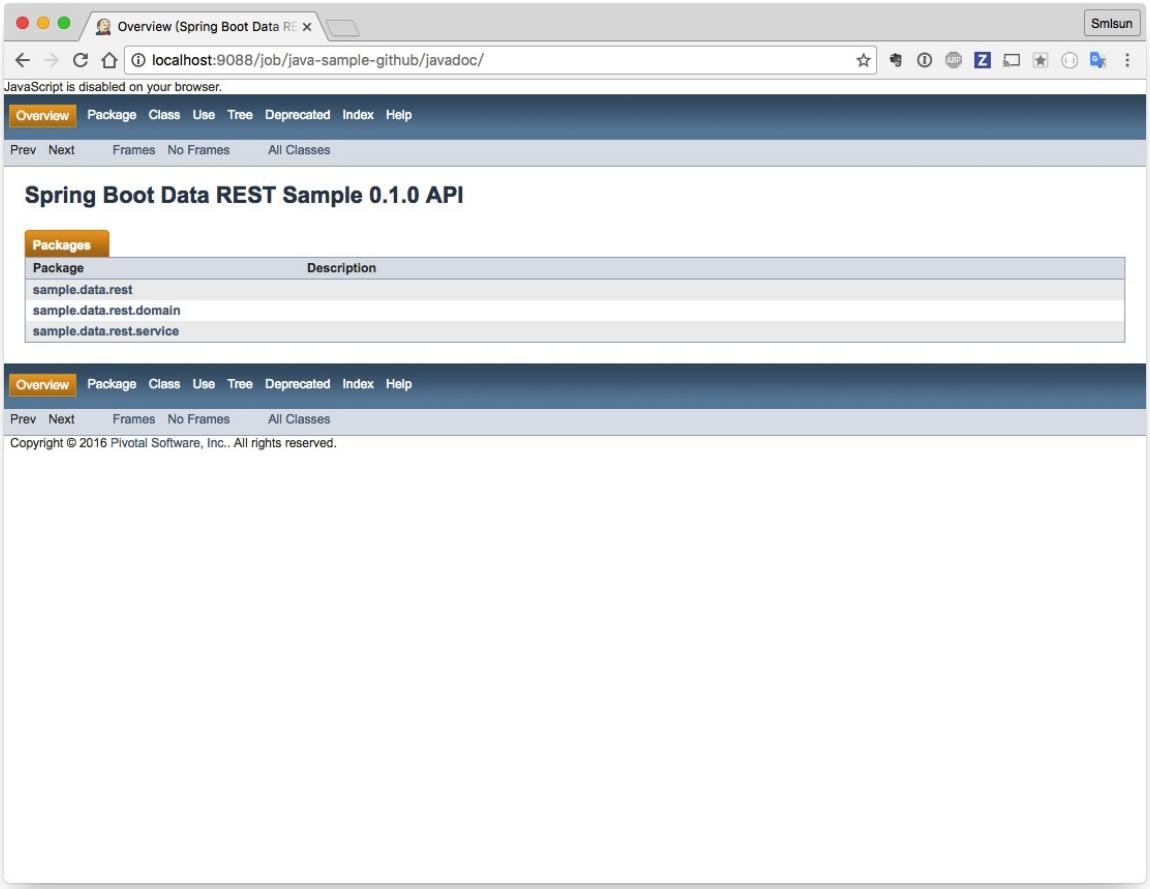
find

#1 2016/7/31 上午 6:11

RSS 摘要: 全部 RSS 摘要: 失敗

永久連結

- 最新建置 (#1), 3 分 26 秒 以前
- 最新穩定建置 (#1), 3 分 26 秒 以前
- 最新成功建置 (#1), 3 分 26 秒 以前
- Last completed build (#1), 3 分 26 秒 以前


Overview (Spring Boot Data RE
Smlsun
localhost:9088/job/java-sample-github/javadoc/
JavaScript is disabled on your browser.
Overview Package Class Use Tree Deprecated Index Help
Prev Next Frames No Frames All Classes
Spring Boot Data REST Sample 0.1.0 API
Packages
Package Description
sample.data.rest
sample.data.rest.domain
sample.data.rest.service
Overview Package Class Use Tree Deprecated Index Help
Prev Next Frames No Frames All Classes
Copyright © 2016 Pivotal Software, Inc.. All rights reserved.

checkStyle

checkStyle 主要用於檢查 coding style 是否正確，透過 checkStyle 可以讓程式為我們的專案進行 Style 的把關

maven POM.xml 設定

透過 maven POM.xml 定義，可以讓我們使用 mvn 指令來進行 checkStyle

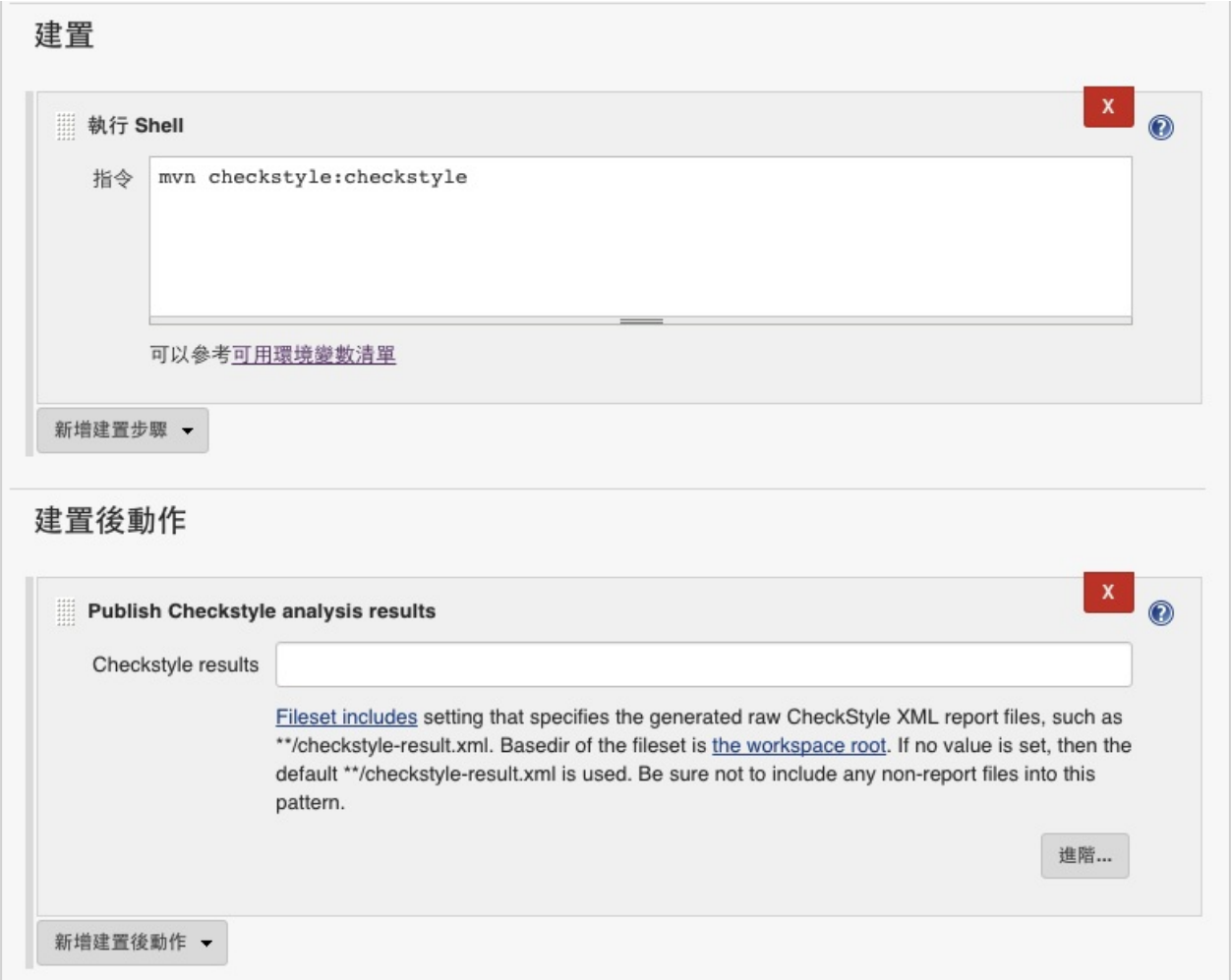
```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-checkstyle-plugin</artifactId>
  <version>2.9.1</version>
  <executions>
      <execution>
      <id>checkstyle</id>
      <phase>validate</phase>
      <goals>
          <goal>check</goal>
      </goals>
      <configuration>
          <failOnViolation>false</failOnViolation>
      </configuration>
      </execution>
  </executions>
</plugin>
```

執行 `mvn checkstyle:checkstyle` 來進行 checkstyle 的執行

可以在 `${project_home}/target/site/checkstyle.html` 檢視 report 的產出。

而用於 jenkins 產出 report 所需的 xml 檔案路徑為 `target/checkstyle-result.xml`

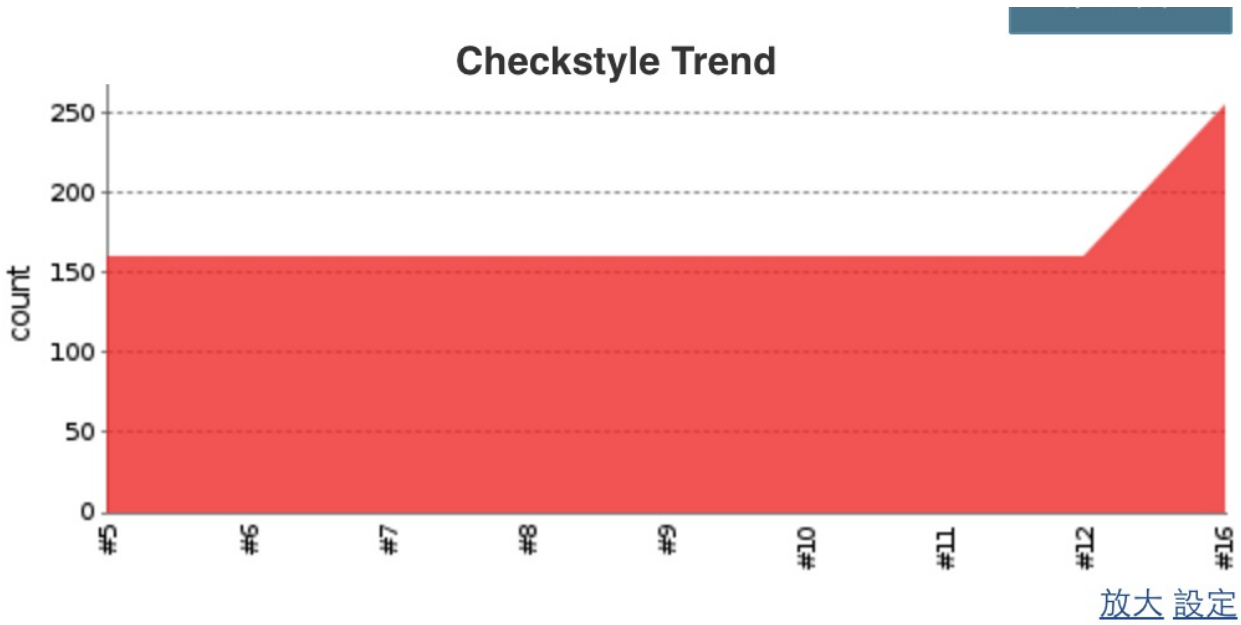
利用 Jenkins 產出 Report



若 `Checkstyle results` 沒有設置的情況下，預設為 `**/checkstyle-result.xml`

故這邊不做設置，剛好符合 report 產出位置。

產出相關報表如下：



CheckStyle Result

警告趨勢

所有警告	新增的警告	修正的警告
255	44	0

摘要

總計	高優先順序	一般優先順序	低優先順序
255	255	0	0

Details

套件 | 檔案 | 類別 | 類型 | 警告 | 詳細資料 | 新增

套件 (Package)	總計	分佈
sample.data.rest	8	
sample.data.rest.domain	168	
sample.data.rest.service	79	
總計	255	

sample.data.rest - 類別 Checks

摘要

總計	高優先順序	一般優先順序	低優先順序
1	1	0	0

Details

詳細資料

SampleDataRestApplication.java:25, FinalParametersCheck, 優先順序: 高

Parameter args should be final.

Check that parameters for methods, constructors, and catch blocks are final. Interface, abstract, and native methods are not checked: the final keyword does not make sense for interface, abstract, and native method parameters as there is no code that could modify the parameter.

Rationale: Changing the value of parameters during the execution of the method's algorithm can be confusing and should be avoided. A great way to let the Java compiler prevent this coding style is to declare parameters final.

檔案 SampleDataRestApplication.java 的內容

```
01 /*
02  * Copyright 2012-2015 the original author or authors.
03  *
04  * Licensed under the Apache License, Version 2.0 (the "License");
05  * you may not use this file except in compliance with the License.
06  * You may obtain a copy of the License at
07  *
08  *      http://www.apache.org/licenses/LICENSE-2.0
09  *
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  */
16
17 package sample.data.rest;
18
19 import org.springframework.boot.SpringApplication;
20 import org.springframework.boot.autoconfigure.SpringBootApplication;
21
22 @SpringBootApplication
23 public class SampleDataRestApplication {
24
25   public static void main(String[] args) throws Exception {
26     SpringApplication.run(SampleDataRestApplication.class, args);
27   }
28
29 }
```

參考資料

http://www.cnblogs.com/huang0925/p/3148389.html

PMD

PMD 是一套原始碼檢查的工具。我們可以它檢查原始碼，如：未使用的 variables, 空的 catch blocks, 不需要被建立的 object，並產生建議報告。

maven POM.xml 設定

```
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-pmd-plugin</artifactId>
   <version>3.6</version>
   <configuration>
      <linkXref>true</linkXref>
      <sourceEncoding>utf-8</sourceEncoding>
      <minimumTokens>20</minimumTokens>
      <format>xml</format>
      <failOnViolation>false</failOnViolation>
   </configuration>
   <executions>
      <execution>
        <phase>compile</phase>
        <goals>
          <goal>pmd</goal>
          <goal>check</goal>
        </goals>
      </execution>
   </executions>
</plugin>
```

設置好 POM.xml 之後，可以透過 `mvn pmd:pmd` 來產生 PMD 報表。

檢視 html 報表可以透過 `${project_home}/target/site/pmd.html` 來存取

若要透過套件產生 report 所需的 xml 檔案路徑為 `${project_home}/target/pmd.xml`

使用 jenkins 設置產生報表

參考下圖進行設置：

![()[assets/README-7f2b0.png]

指定到 PMD 產出原始 XML 報告檔的檔案集，例如 **/pmd.xml。檔案集起算目錄就是工作區根目錄。如果沒有設定，預設會是** /pmd.xml

報表呈現如下：

PMD 結果

警告趨勢

所有警告	新增的警告	修正的警告
3	0	0

摘要

總計	高優先順序	一般優先順序	低優先順序
3	0	3	0

Details

檔案 警告 詳細資料

檔案	總計	分佈
City.java	1	
CityTemp.java	1	
Hotel.java	1	
總計	3	

PMD 警告 - File City.java

摘要

總計	高優先順序	一般優先順序	低優先順序
1	0	1	0

Details

詳細資料

City.java:33, UnusedPrivateField, 優先順序: 一般

Avoid unused private fields such as 'id'.

Detects when a private field is declared and/or assigned a value, but not used.

```
public class Something {
  private static int FOO = 2; // Unused
  private int i = 5; // Unused
  private int j = 6;
  public int addOne() {
    return j++;
  }
}
```

```
26 @Entity
27 public class City implements Serializable {
28
29   private static final long serialVersionUID = 1L;
30
31   @Id
32   @GeneratedValue
33   private Long id;
34
35   @Column(nullable = false)
36   private String name;
37
38   @Column(nullable = false)
39   private String state;
40
41   @Column(nullable = false)
42   private String country;
43
44   @Column(nullable = false)
45   private String map;
46
47   protected City() {
48   }
49
```

Avoid unused private fields such as 'id'. Detects when a private field is declared and/or assigned a value, but not used.

```
public class Something {
  private static int FOO = 2; // Unused
  private int i = 5; // Unused
  private int j = 6;
  public int addOne() {
    return j++;
  }
}
```

dry / CPD

檢查源碼內重覆的部分

maven POM.xml 設定

```
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-pmd-plugin</artifactId>
   <version>3.6</version>
   <configuration>
      <linkXref>true</linkXref>
      <sourceEncoding>utf-8</sourceEncoding>
      <minimumTokens>20</minimumTokens>
      <format>xml</format>
      <failOnViolation>false</failOnViolation>
   </configuration>
   <executions>
      <execution>
        <phase>compile</phase>
        <goals>
          <goal>cpd</goal>
          <goal>cpd-check</goal>
        </goals>
      </execution>
   </executions>
</plugin>
```

設置好之後，我們可以透過 `mvn pmd:cpd-check` 指令來進行檢查

其中 `<minimumTokens>20</minimumTokens>` 可以讓你設置最小重覆的字數，以便設定檢查範圍

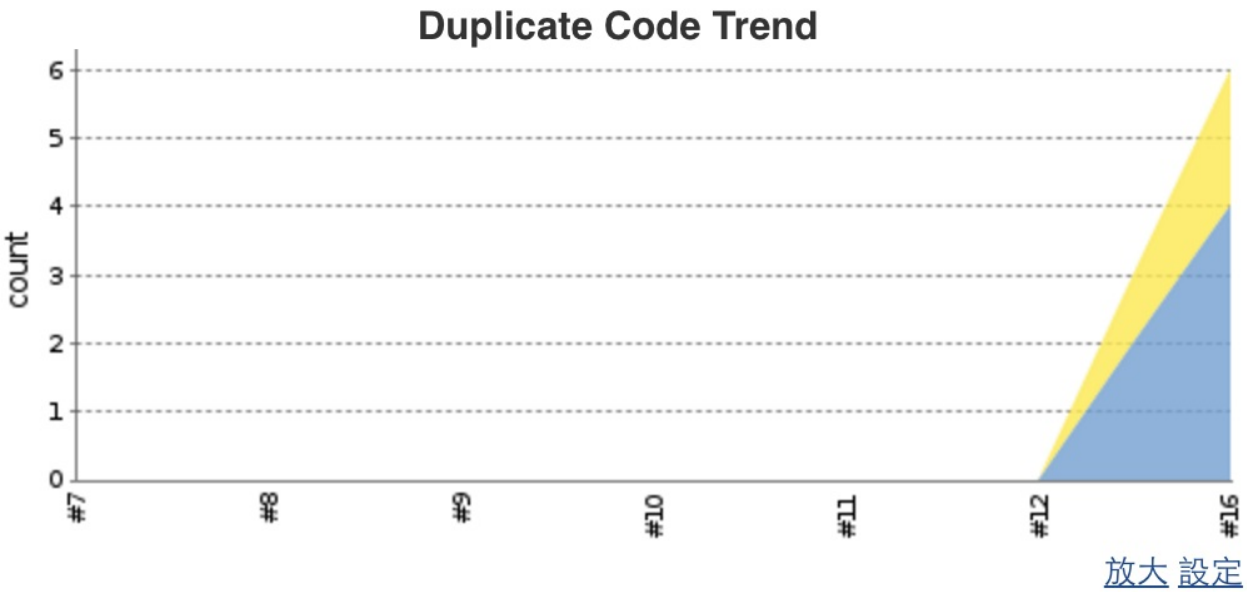
運行完檢查後，我們可以透過 `${project_home}/target/site/cpd.html` 來查看報表

若要透過套件產生 report 所需的 xml 檔案路徑為 `${project_home}/target/cpd.xml`

使用 Jenkins 進行 check 並產生報表

指定到 CPD 產出原始 XML 報告檔的檔案集，例如 **/cpd.xml。檔案集起算目錄就是工作區根目錄。如果沒有設定，預設會是** /cpd.xml

相關報表畫面如下：



Duplicate Code Result

警告趨勢

所有警告	新增的警告	修正的警告
6	6	0

摘要

總計	高優先順序	一般優先順序	低優先順序
6	0	2	4

Details

套件 | 檔案 | 警告 | 詳細資料 | 新增 | 一般 | 低

套件 (Package)	總計	分佈
sample.data.rest.domain	4	
sample.data.rest.service	2	
總計	6	

sample.data.rest.domain - File City.java

摘要

總計	高優先順序	一般優先順序	低優先順序
2	0	1	1

Details

警告 | 詳細資料 | 一般 | 低

File	Number of lines	Duplicated in
City.java:27	21	CityTemp.java:27
City.ja		CityTemp.java:50

```
public class City implements Serializable {

  private static final long serialVersionUID = 1L;

  @Id
  @GeneratedValue
  private Long id;

  @Column(nullable = false)
  private String name;

  @Column(nullable = false)
  private String state;

  @Column(nullable = false)
  private String country;

  @Column(nullable = false)
  private String map;
```

```
19 import java.io.Serializable;
20
21 import javax.persistence.Column;
22 import javax.persistence.Entity;
23 import javax.persistence.GeneratedValue;
24 import javax.persistence.Id;
25
26 @Entity
27 public class City implements Serializable {
28
29   private static final long serialVersionUID = 1L;
30
31   @Id
32   @GeneratedValue
33   private Long id;
34
35   @Column(nullable = false)
36   private String name;
37
38   @Column(nullable = false)
39   private String state;
40
41   @Column(nullable = false)
42   private String country;
43
44   @Column(nullable = false)
45   private String map;
46
47   protected City() {
48   }
```

21 lines of duplicate code.

Duplicated in:

- sample.data.rest.domain.CityTemp (27)

參考資料

http://kentyeh.github.io/mavenStartup

Simian - Similarity Analyser

Simian 與 CPD 功能類似，只支援 maven 1 所以在比較心得專案沒辦法透過設定 POM.xml 的方式來進行。

不過我們還是可以透過 command 的方式來產生相關報表，指令如下

```
wget -nc http://www.harukizaemon.com/simian/simian-2.4.0.tar.gz
tar xvzf simian-2.4.0.tar.gz bin/simian-2.4.0.jar
mv bin/simian-2.4.0.jar ./simian.jar
java -jar simian.jar \
-formatter=xml:target/simian.xml \
-threshold=10 \
-language=java \
**/*.java
```

如此我們就可以產出 `${project_home}/target/simian.xml` 令 jenkins 可以進行報表輸出

jenkins 報表輸出設置

建置

執行 Shell

指令

```
wget -nc http://www.harukizaemon.com/simian/simian-2.4.0.tar.gz
tar xvzf simian-2.4.0.tar.gz bin/simian-2.4.0.jar
mv bin/simian-2.4.0.jar ./simian.jar
java -jar simian.jar \
-formatter=xml:target/simian.xml \
+failOnDuplication \
-threshold=10 \
-language=java \
**/*.java
```

可以參考可用環境變數清單

新增建置步驟

建置後動作

Publish duplicate code analysis results

Duplicate code results: **/simian.xml

Fileset includes setting that specifies the generated raw XML report files, such as **/cpd.xml or **/simian.xml. Basedir of the fileset is the workspace root. If no value is set, then the default **/cpd.xml is used. Be sure not to include any non-report files into this pattern.

High priority threshold: 50

Minimum number of duplicated lines for high priority warnings.

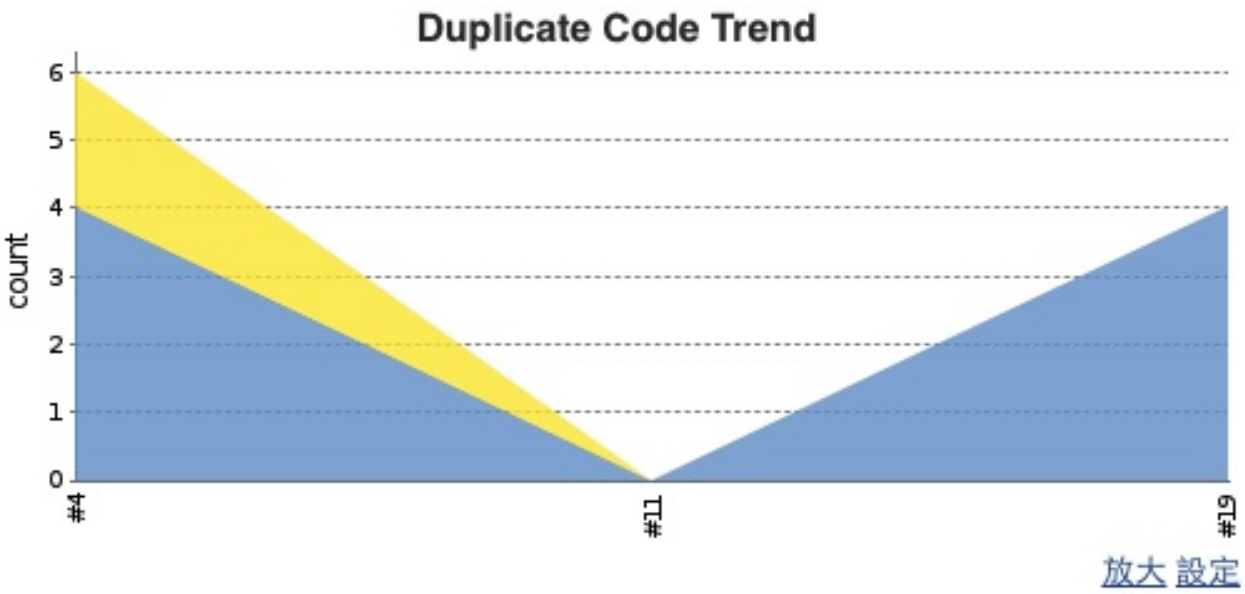
Normal priority threshold: 25

Minimum number of duplicated lines for normal priority warnings.

永遠執行 ☑

預設情況下，本外掛程式只會在穩定或是不穩定的建置上跑，不會在失敗的建置上執行。要是就算建置失敗也要執行，請啟用本功能。

相關報表產出如下：



Duplicate Code Result

警告趨勢

所有警告	新增的警告	修正的警告
4	4	0

摘要

總計	高優先順序	一般優先順序	低優先順序
4	0	0	4

Details

檔案 | 警告 | 詳細資料 | 新增

檔案	總計	分佈
City.java	2	
CityTemp.java	2	
總計	4	

Duplicate Code - File City.java

摘要

總計	高優先順序	一般優先順序	低優先順序
2	0	0	2

Details

警告 | 詳細資料

File	Number of lines	Duplicated in
City.java:29	12	CityTemp.java:29
City.java:51	14	CityTemp.java:51

```
26 @Entity
27 public class City implements Serializable {
28
29   private static final long serialVersionUID = 1L;
30
37
38   @Column(nullable = false)
39   private String state;
40
41   @Column(nullable = false)
42   private String country;
```

12 lines of duplicate code.

Duplicated in:

- sample.data.rest.domain.CityTemp (29)

FindBugs

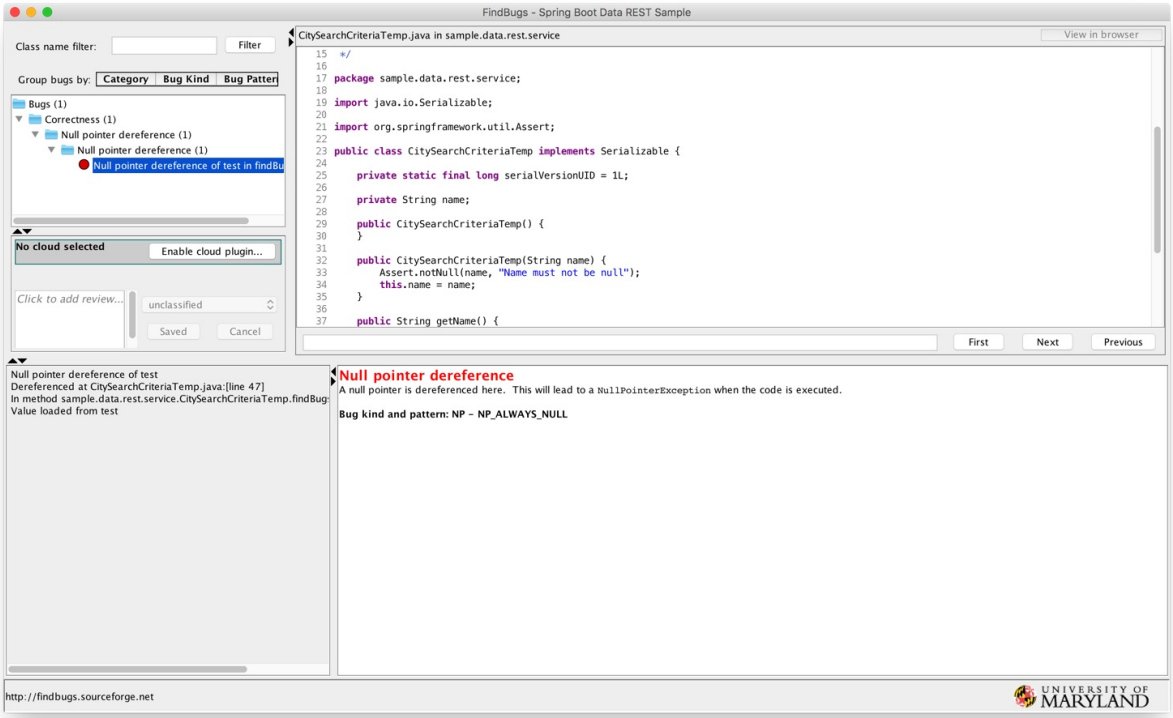
FindBugs 檢測的標的不是源碼，而是 binary code，所以一定要先打包好專案才能執行檢查。

maven POM.xml 設定

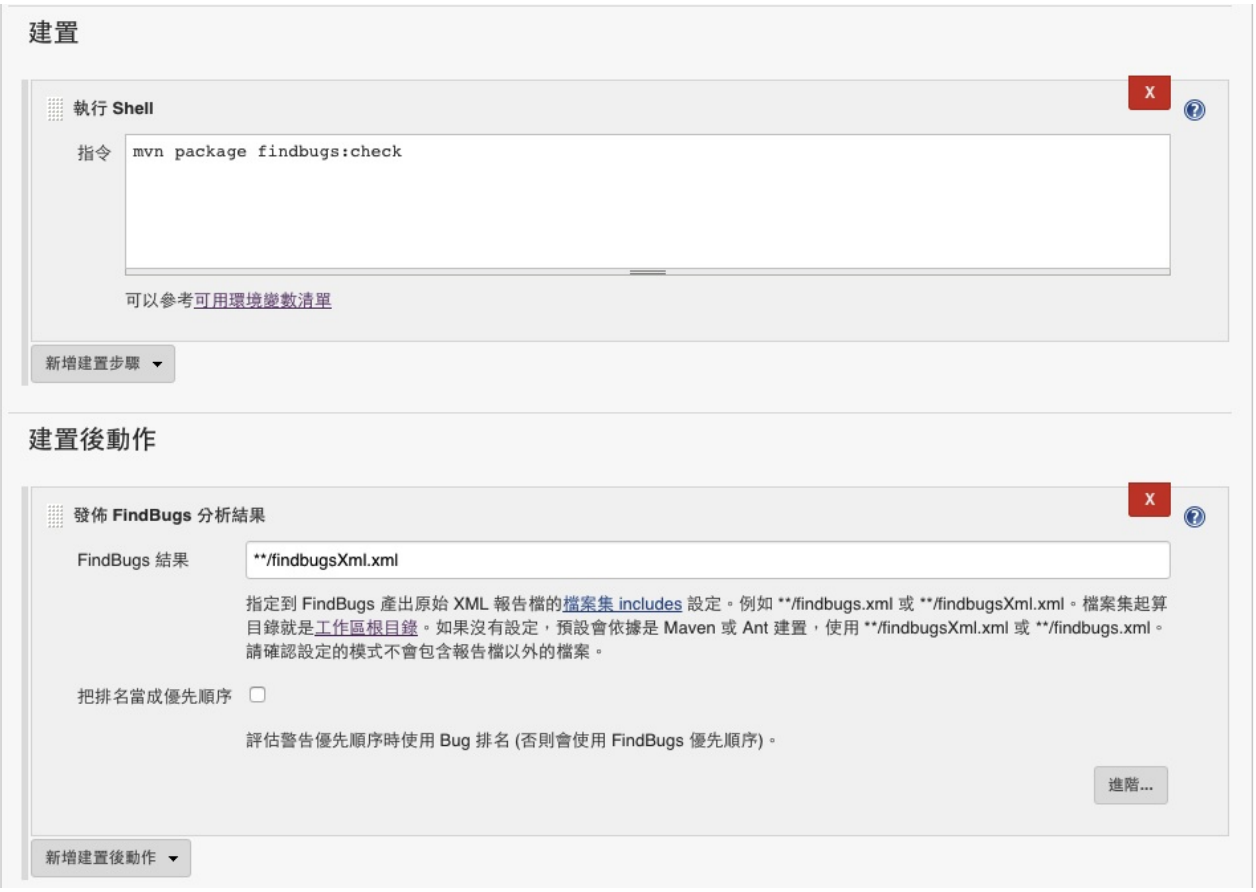
```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>findbugs-maven-plugin</artifactId>
    <version>3.0.1</version>
    <executions>
        <execution>
            <goals>
                <goal>check</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <effort>Max</effort><!--檢測的程度，值可為Min、Default或是Max-->
        <xmlOutput>true</xmlOutput>
        <failOnError>false</failOnError>
    </configuration>
</plugin>
```

設置好 POM.xml 後，我們可以透過 `mvn package findbugs:check` 來產出報表。

當檢測出缺點時，我們可以用指令 `mvn findbugs:gui` 透過 GUI 查看 xml 的輸出，畫面如下



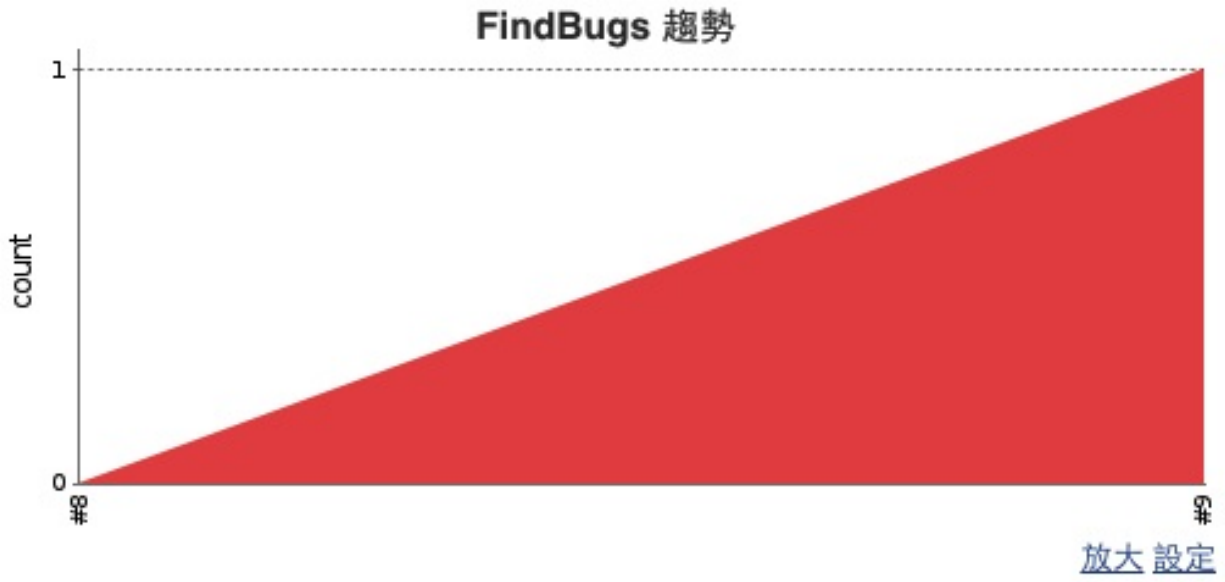
使用 Jenkins 進行 check 並產生報表



指定到 FindBugs 產出原始 XML 報告檔的檔案集，例如 **/findbugs.xml 或** /findbugsXml.xml。檔案集起算目錄就是工作區根目錄。如果沒有設定，預設會依據是 Maven 或 Ant 建置，使用 **/findbugsXml.xml 或** /findbugs.xml。

因為我們是使用 maven 所以這邊需要設置為 `**/findbugsXml.xml`

相關報表畫面如下：



FindBugs 警告

警告趨勢

全部警告	本次建置新增	修正的警告
1	1	0

摘要

總計	高優先順序	一般優先順序	低優先順序
1	1	0	0

Details

詳細資料 | 新增

CitySearchCriteriaTemp.java:47, NP_ALWAYS_NULL, 優先順序: 高

Null pointer dereference of test in sample.data.rest.service.CitySearchCriteriaTemp.findBugsTest()

A null pointer is dereferenced here. This will lead to a `NullPointerException` when the code is executed.

```
40
41  public void setName(String name) {
42    this.name = name;
43  }
44
45  public String findBugsTest() {
46    String test = null;
47    return test.toUpperCase();
48  }
49
```

Null pointer dereference of test in sample.data.rest.service.CitySearchCriteriaTemp.findBugsTest()

A null pointer is dereferenced here. This will lead to a `NullPointerException` when the code is executed.

參考資料

http://kentyeh.github.io/mavenStartup

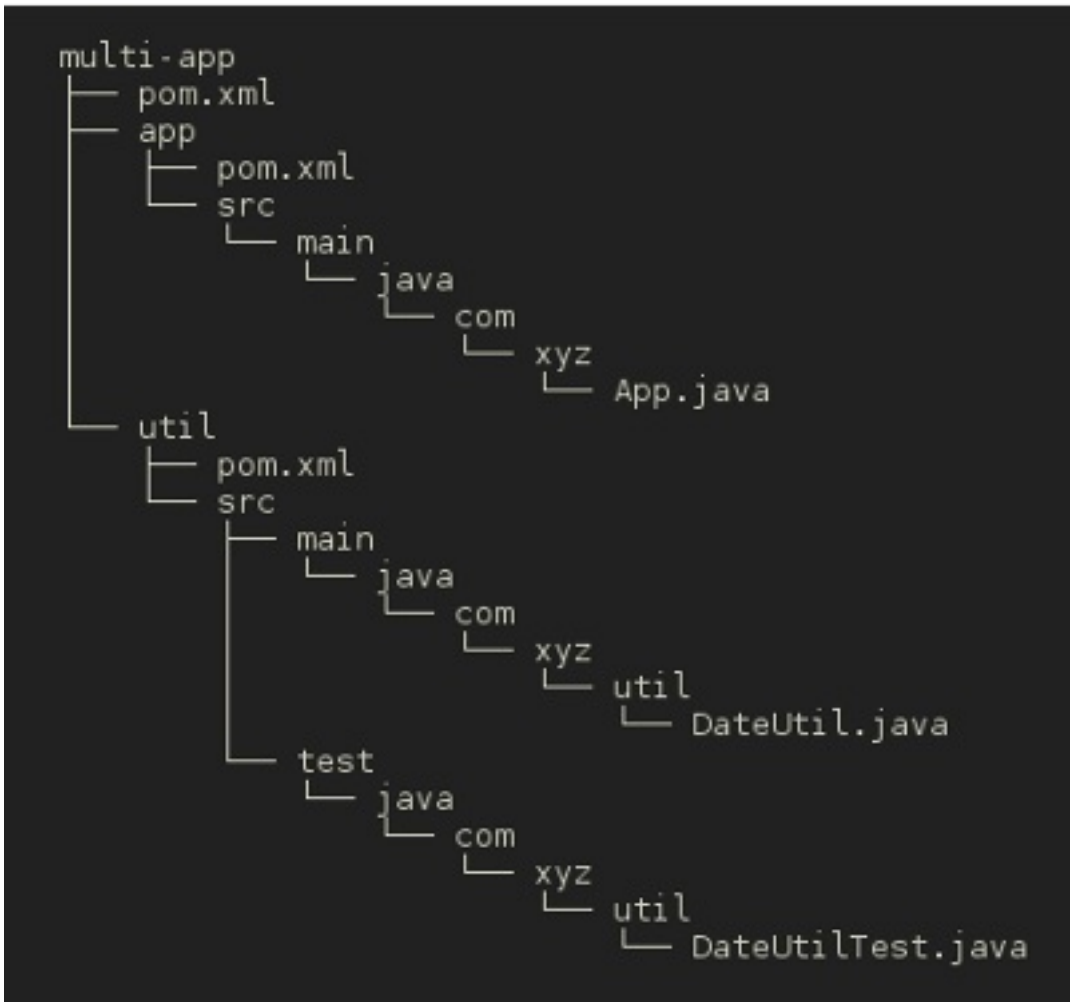
Maven Multi Module Project

範例專案

https://github.com/agileworks-tw/maven-multi-app-sample

範例專案檔案結構

由下圖可以看到，此範例專案是有 app 及 util 所組成，相關設定將會由 `multi-app/pom.xml` 進行組合



multi-app/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org

        <modelVersion>4.0.0</modelVersion>

        <!-- project coordinates -->
        <groupId>com.xyz</groupId>
        <artifactId>multi-app</artifactId>
        <version>1.0</version>
        <packaging>pom</packaging>

        <modules>
                <module>app</module>
                <module>util</module>
        </modules>

</project>
```

其中

```
<modules>
        <module>app</module>
        <module>util</module>
</modules>
```

而其中 app 及 util 之 pom.xml 則與一般標準 pom.xml 設置上無異。

在 `multi-app/pom.xml` 可以透過 modules 來設定建置此專案需要的其他專案。

運行

```
git clone https://github.com/agileworks-tw/maven-multi-app-sample
cd maven-multi-app-sample
mvn test
```

運行結果如下

```
[INFO] Reactor Summary:
[INFO]
[INFO] multi-app ........................................... SUCCESS [  0.002 s]
[INFO] util ................................................ SUCCESS [  1.469 s]
```

```
[INFO] app ..................................... SUCCESS [  0.035 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.594 s
[INFO] Finished at: 2016-08-01T08:39:39+08:00
[INFO] Final Memory: 15M/211M
[INFO] ------------------------------------------------------------------------
```

參考資料

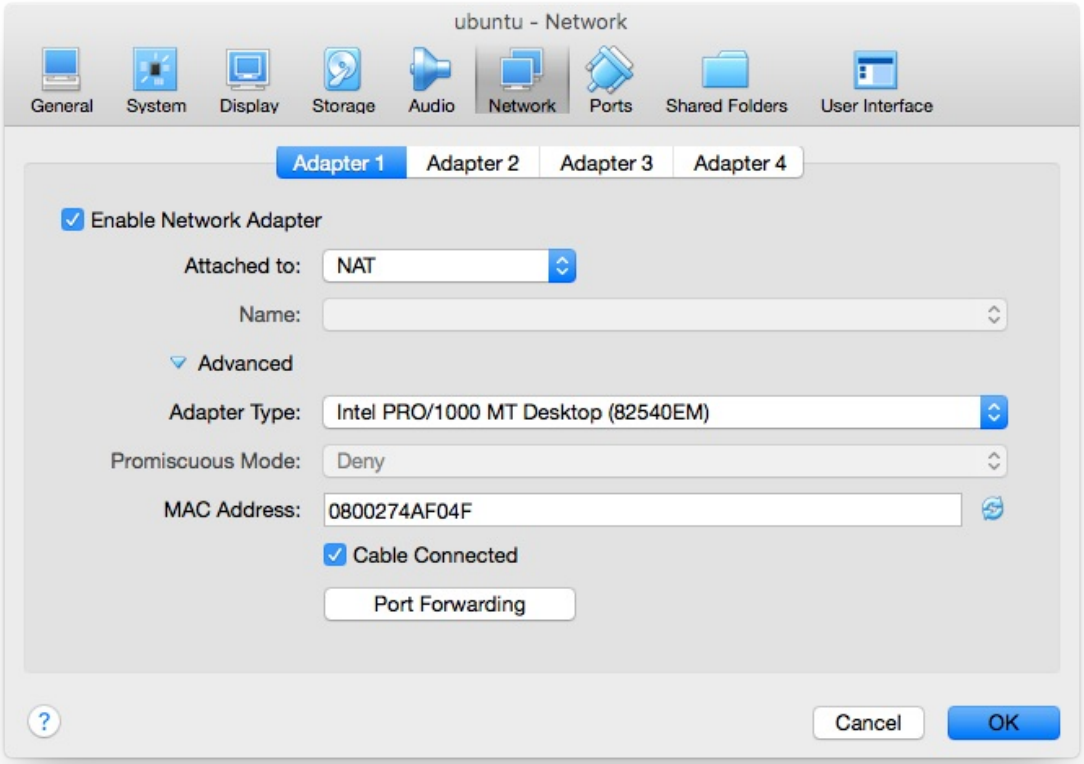
http://books.sonatype.com/mvnex-book/reference/multimodule-sect-intro.html http://www.codetab.org/apache-maven-tutorial/ http://www.codetab.org/apache-maven-tutorial/maven-multi-module-project/

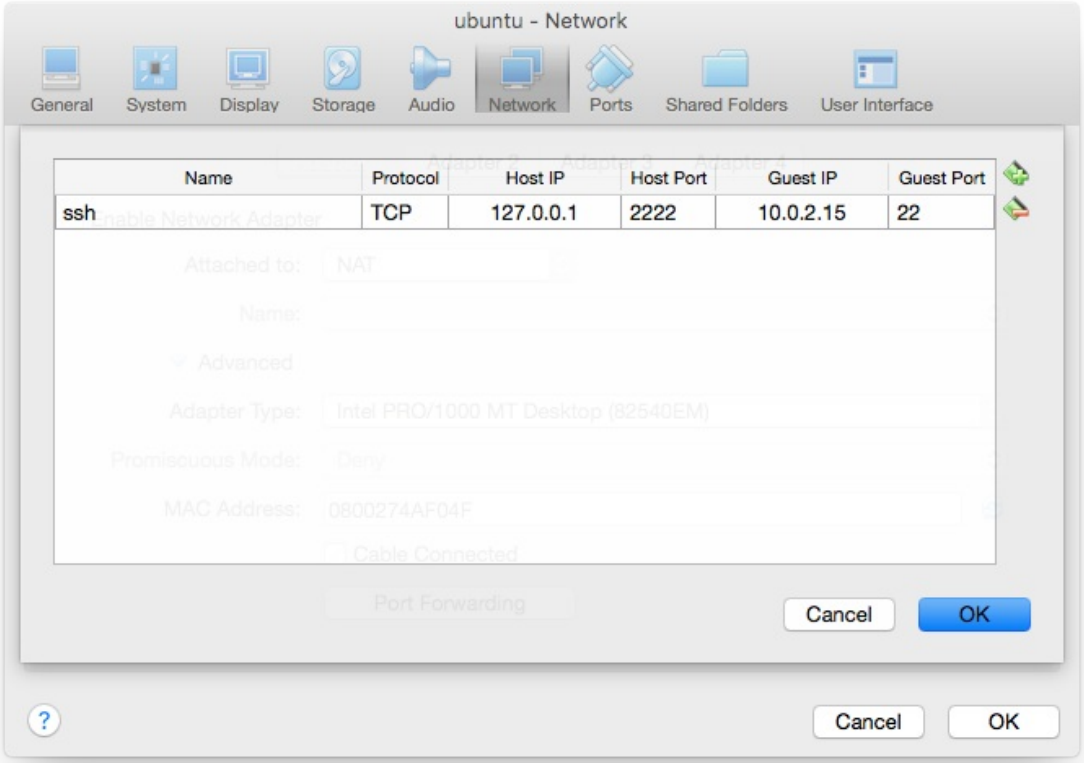
一般問題

VM 問題排解

上課時使用 VM Bridged Adapter network setting 但後來連不到，為了讓課程上的更順利，先改成 NAT 但如此 Host 就沒有辦法連到 Guest VM ，本來要用 Windows Cmder 進行相關操作，只好讓學員用 ubuntu 內的 terminal 進行操作。

後來查了一下資料，用 NAT 要連進 Guest VM 其實很簡單，只要設置一下 Port Forwarding 即可，如下：





給需要的人參考。

關於分支使用的策略

下面連結可以參考：

COSCUP 開源工作坊: Git Workflows

有更詳細的介紹。

希望 N 個 task 完成後才進行目標 task

套件如下：

Join Plugin

environment 替換

套件如下：

EnvInject Plugin

Jenkins做為CI與CD功能，本身有沒有什麼API可以提供做為資訊揭露，有沒有實際使用這些API的例子

- jenkins api doc: https://jenkinsapi.readthedocs.org/en/latest/using_jenkinsapi.html
- https://www.npmjs.com/package/jenkins-api
- 之前的經驗會透過 CI 的 api 取得版本相關資訊
- 也可以搭配 Jenkins script console 執行 https://wiki.jenkins-ci.org/display/JENKINS/Jenkins+Script+Console

Jenkins每次更新都會留下歷史記錄，目前有沒有什麼現有方法可以做house keeping的

jenkins 本身就可以設置希望可以保留多少 task，若要定期 archive 有相關的 plugin 可以處理

本身是Java project，有沒有推薦使用的plug-in以及如何離線安裝plug-in

- config 相關
- ssh 傳檔案相關
- 版本控制相關
- task 管理相關
- test report 相關
- 離線安裝透過 jenkins web UI 即可進行
- 在 command-line 模式可以這樣做：

Download the Plugin (*.hpi File) and put it in the following directory: %JENKINS_HOME%/plugins/

有可能Jenkins做完測試後，自動幫我回覆issue tracking system說已解嗎? 有plug-in解還是要自己寫code?

- e.g. Jenkins + Jira

除了CI與CD外，Jenkins還可以做到什麼程度，譬如產報表?

- 測試覆蓋率報表
- 記錄每次發版的 change log
- 壓力測試報表
- Unit Testing 報表: JUnit XML Report supported
- 自動監測服務狀態、...一些經常需要被重複執行的任務，特別是執行結果需要被記錄（或檢驗），只要能用 command-line / scripting 方式定義，就可以交給 Jenkins 來處理。

做CI與CD有什麼雷是千萬不要踩的，譬如說不要做在Production上?

- 準備一個跟 production 一樣的環境該環境除了拿來進行 preview 之外，在上 production 之前可以確認新的 deploy 有沒有其他潛在問題。
- jenkins 建置所需 library 如 jdk 避免使用 jenkins 的套件安裝，為了 debug 方便，建議使用 jenkins linux user 環境設置進行相關套件引入。
- CI 和 CD 只是開發流程的一環節，僅是輔助的用途。

JAVA project 問題

以一個Java Project來說，做什麼樣的調整會最符合Jenkins希望的樣子

- 利用 maven 或是 gradle 進行 auto build
- test build and production build 盡可能讓一個或少數幾個 command 完成
- 利用 maven task 來定義相關的 task
- 盡可能用 maven repository、減少 local 放置 JAR libraries 檔案
- 若是 java 專案，且又是 web 的話可內嵌 tomcat 或 jetty 方便開發測試
- 部分專案設定配置搭配環境變數作定義
- 盡量讓開發建置環境單純化，避免使用太多需要特別賦予網路存取權限的其他服務

很多測試環境需要很多機器的參與，最明顯的就是DB，假設Jenkins要做自動化測試對於資料庫的處置有什麼建議?

- 若有使用 orm (hibernate) 建議可以開 drop 來處理，也就是每次都新建資料庫
- 自動化測試必須要讓資料庫隨時有新的開始
- 沒有 orm 也應該要使用 sql script 來初始資料
- 如果專案允許，搭配 H2 / sqlite 等輕量化資料庫引擎並搭配 test 環境的配置與初始化資料

vt-x amd-v 異常造成 VM 無法開啟

在執行 VM 時出現下列訊行

> 電腦問題(0022) - VirtualBox 錯誤訊息 VT-x/AMD-V 硬體加速在您的系統不可用

我們可以透過下面步驟來進行排除

BIOS 設定

Intel

在 Intel 電腦的 BIOS 中，(有可能是在 Advanced，或者 CPU Configuration 裡面) 找一個名稱如 Intel Virtual Technology 或 Virtualization 之類的選項，如果原設定是 Disabled，則將它改為 Enabled 之後再儲存，離開 BIOS 重開機就可以了。

AMD

在 AMD 電腦的 BIOS 中，(有可能是在 Advanced，或者 CPU/System Configuration 裡面) 找一個名稱是 SVM Mode 或 Virtualization Technology 或 Virtualization 之類的去改為 Enabled，儲存之後離開 BIOS，再重開機就可以了。

參考資料

https://notepad-of-steven.blogspot.tw/2015/08/0022-virtualbox-vt-xamd-v.html

ubuntu docker install

```
sudo apt-key adv --keyserver hkp://pgp.mit.edu:80 --recv-keys 58118E89F3A912897C070AD
sudo add-apt-repository "deb https://apt.dockerproject.org/repo ubuntu-$(lsb_release
sudo apt-get update
sudo apt-get install docker-engine
```

設定 user 可以執行 docker

```
sudo groupadd docker
sudo usermod -a -G docker {username}
sudo service docker restart
```

確認安裝是否完成

```
docker -v
docker ps
docker run hello-world
```

注意事項

如果還是不行，可以重開 ubuntu 再次[確認安裝是否完成]。

docker-compose install

下述指令請用 `root` 執行

```
curl -L https://github.com/docker/compose/releases/download/1.6.2/docker-compose-`una
chmod +x /usr/local/bin/docker-compose
```

docker-machine install

下述指令請用 `root` 執行

```
curl -L https://github.com/docker/machine/releases/download/v0.5.0/docker-machine_lin
unzip machine.zip && \
rm machine.zip && \
mv docker-machine* /usr/local/bin
```

OSX or Windows install docker

安裝 docker-tools

docker-tools: https://www.docker.com/toolbox

安裝 docker-tools 將會自動安裝 Docker，docker-machine，docker-compose

透過 docker-machine 建立 docker engine vm

`docker-machine create -d virtualbox dev`

啟動 docker-machine

```
docker-machine start dev
```

預期結果

```
Starting VM...
Started machines may have new IP addresses. You may need to re-run the `docker-machir
```

透過 docker-machine 連結 docker engine vm

執行下列指令完成連結

```
eval $(docker-machine env default)
```

啟動 bash 時自動連結 docker engine vm

執行下列指令將連結

```
eval $(docker-machine env default)
```

確認 docker 運作正常

若要確認是否正常運行，可以執行下列指令：

```
docker ps
```

正常將出現

```
CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES
```

若出現下列訊息

```
Get http:///var/run/docker.sock/v1.20/containers/json: dial unix /var/run/docker.sock
* Are you trying to connect to a TLS-enabled daemon without TLS?
* Is your docker daemon up and running?
```

就有可能需要執行 `$(docker-machine env default)` 來進行連接。

若再不行，請

1. 確定 Virtual Box 是否最新 ，5.0.x 以上
2. 刪除 Virtual Box 管理介面的 default VM
3. 重新執行 Docker Quickstart Terminal

因為是使用 VM 的關係，若網路斷斷續續也會造成 Docker 網路不穩定，所以若有網路連不上的情形可以再重啟 VM

參考資料

- 《Docker —— 從入門到實踐》正體中文版

參考資料

- 連猴子都能懂的Git入門指南

相關練習解答

製作 docker-compose 之 dev mode service

```
data:
  image: ubuntu
  volumes:
    - /root/.m2

server:
  image: maven:3-jdk-8
  volumes:
    - ./:/app
  volumes_from:
    - data
  working_dir: /app
  ports:
   - "8000:8000"
  command: "/bin/bash"
```

docker production 發布流程練習

```
docker login -e test@gmail.com -p testpass -u testuser
docker-compose run package
docker build -t smlsunxie/sample_prod .
docker push smlsunxie/sample_prod
```

使用 production image 運行專案進行 preview

```
docker stop preview
docker rm preview
docker run -d -p 8000:8000 --name preview smlsunxie/sample_prod
```

jenkins 建立 production deploy task

task shell script

```
docker login -e test@gmail.com -p testpass -u testuser
docker-compose run package
docker build -t smlsunxie/sample_prod .
docker push smlsunxie/sample_prod
```

透過 ssh 連線到 production 機器運行建置好的 image

```
ssh jenkins@localhost ' \

  docker stop prod && \
  docker rm prod && \
  docker run -d -p 8800:8800 --restart always --name prod smlsunxie/sample_prod \

'
```