

Jenkins

- [【Jenkins】相關連結](#)
- [【Jenkins】常用plug-in](#)
- [【Jenkins】Jenkinsfile](#)
- [【GitLab】push 觸發pipeline](#)
- [【Jenkins】docker-compose](#)
- [【Jenkins】如何執行排程作業](#)
- [【Jenkins】執行git專案上的pipeline](#)

【Jenkins】相關連結

Jenkins入门 pipeline (官方文檔)

<https://www.jenkins.io/zh/doc/book/pipeline/syntax/>

<https://www.jenkins.io/doc/book/pipeline/>

- 使用 Shared Libraries
 - [Getting Started With Shared Libraries in Jenkins](#)
 - [Using Resource Files From a Jenkins Shared Library](#)
- pipeline語法
 - [What Is the Difference Between Scripted and Declarative Pipeline](#)

【udemy】持續集成（Continuous Integration with Jenkins）---從初學到精通
[Jenkins.pdf](#)

【Docker Hub】Jenkins 官說明
[Jenkins - Official Image | Docker Hub](#)

Jenkins+Github觸發器
[Jenkins : GitHub Plugin \(jenkins-ci.org\)](#)

PMD Jenkins plugin:
<https://wiki.jenkins-ci.org/display/JENKINS/PMD+Plugin>

Findbugs Jenkins Plugin:
<https://wiki.jenkins-ci.org/display/JENKINS/FindBugs+Plugin>

[Postman+Newman+Git+Jenkins介面自動化測試 | IT人 \(iter01.com\)](#)

Jenkins pipeline

[\[Day 27\] Jenkins \(1\) - iT 邦幫忙::一起幫忙解決難題，拯救 IT 人的一天 \(ithome.com.tw\)](#)

在 Jenkins 容器中執行 docker 指令

[Jenkins 容器中執行 docker 指令：叢揚部落格 \(gss.com.tw\)](#)

```
docker run -d --name test-jenkins \  
--user root -p 8080:8080 \  
-p 50000:50000 \  
-v /var/run/docker.sock:/var/run/docker.sock \  
jenkins/jenkins
```

```
docker exec -it --user root test-jenkins /bin/bash
```

```
apt-get update && apt-get -y install apt-transport-https ca-certificates \  
curl gnupg2 software-properties-common && \  
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; \  
apt-key add /tmp/dkey && \  
add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") $(lsb_release -cs) stable" && \  
apt-get update && apt-get -y install docker-ce
```

【Jenkins】常用plug-in

Deploy to container Plugin



This plugin allows you to deploy a war to a container after a successful build.
Glassfish 3.x remote deployment

1.16

Copy Artifact Plugin



Adds a build step to copy artifacts from another project.

1.46.2

Build Pipeline

建置工具

其他建置後動作

使用者介面

This plugin renders upstream and downstream connected jobs that typically form a build pipeline. In addition, it offers the ability to define manual triggers for jobs that require intervention prior to execution, e.g. an approval process outside of Jenkins.



Warning: This plugin version may not be safe to use. Please review the following security notices:

- [Stored XSS vulnerability](#)

1.5.8

4 年 2 月 ago



Locale 546.v1609030511ca_

localization

User Interface

This plugin lets you control the language of Jenkins.

【Jenkins】 Jenkinsfile

Using a Jenkinsfile

```
pipeline {
    agent any
    stages{
        stage('Init'){
            steps {
                echo "Testing....."
            }
        }
        stage('Build'){
            steps {
                echo "Building....."
            }
        }
        stage('Deploy'){
            steps {
                echo "Code Deployed."
            }
        }
    }
}
```

```
pipeline {
    agent any

    tools{
        maven 'local maven'
    }

    parameters{
        string(name: 'tomcat_dev', defaultValue: '1.1.1.1', description: 'Staging Server')
        string(name: 'tomcat_prod', defaultValue: '2.2.2.2', description: 'Production Server')
    }

    triggers {
        pollSCM('* * * * *')
    }

    stages{
        stage('Build'){
            steps {
                sh 'mvn clean package'
            }
            post {
                success {
                    echo '开始打包...'
                    archiveArtifacts artifacts: '**/target/*.war'
                }
            }
        }
    }

    stage ('Deployments'){
        parallel{
            stage ('Deploy to Staging'){
                steps {
                    sh "scp -i /Users/gaoyan/Documents/SunnyDemo/dev/tomcat-demo.pem.txt **/target/*.war ec2-user@${params.tomcat_dev}:/var/lib/tomcat8/webapps"
                }
            }

            stage ("Deploy to Production"){
                steps {

```

```
sh "scp -i /Users/gaoyan/Documents/SunnyDemo/dev/tomcat-demo.pem.txt **/target/*.war ec2-  
user@${params.tomcat_prod}:/var/lib/tomcat8/webapps"  
}  
}  
}  
}  
}  
}
```

- [\[Day 25\] Pipeline 專案 \(上\) - iT 邦幫忙::一起幫忙解決難題，拯救 IT 人的一天 \(ithome.com.tw\)](#)
- [\[Day 26\] Pipeline 專案 \(中\) - iT 邦幫忙::一起幫忙解決難題，拯救 IT 人的一天 \(ithome.com.tw\)](#)
- [\[Day 27\] Pipeline 專案 \(下\) - iT 邦幫忙::一起幫忙解決難題，拯救 IT 人的一天 \(ithome.com.tw\)](#)

問題1： `**\target*.war: No such file or directory`

解決辦法：

Jenkins無法根據正則匹配找到war文件，我們需要在Jenkinsfile中指定到war文件的完整路徑，例如/var/jenkins_home/workspace/package_pipeline/webapp/target/*.war(更改成你機器上的實際路徑)。而不是**\target*.war

如果出現下面錯誤提示：

```
[Deploy to Production] Host key verification failed.  
[Deploy to Production] lost connection
```

解決思路如下：

請改用如下命令

```
1.  
scp -o StrictHostKeyChecking=no -i /Users/gaoyan/Documents/SunnyDemo/dev/tomcat-demo.pem.txt  
webapp/target/webapp.war ec2-user@52.15.183.253:/var/lib/tomcat8/webapps
```

加了-o StrictHostKeyChecking=no參數，這樣主機密鑰就會在連接時自動加入到known_hosts中去

2.

命令行也用加了-o參數的命令手動登錄試試

```
ssh -o StrictHostKeyChecking=no ec2-user@13.58.179.83 -i tomcat-demo.pem.txt
```

這樣主機密鑰就會在連接時自動加入到known_hosts中去

3.保證用jenkins執行的時候 tomcat-demo.pem.txt這個文件其他用戶對他有操作權，但是注意執行上面第二個命令時，會提示其他人不能對這個文件有訪問權，所以用sudo chmod 700 tomcat-demo.pem.txt，執行完這個命令以後，再把權限改成 644，sudo chmod 644 tomcat-demo.pem.txt，然後去jenkins執行第一條命令

4.還不成功刪除known_hosts試試，cd ~/.ssh

```
rm known_hosts
```

如果出現：

```
Permissions 0644 for 'jenkinskey.pem' are too open.  
It is required that your private key files are NOT accessible by others.  
This private key will be ignored.  
Load key "jenkinskey.pem": bad permissions
```

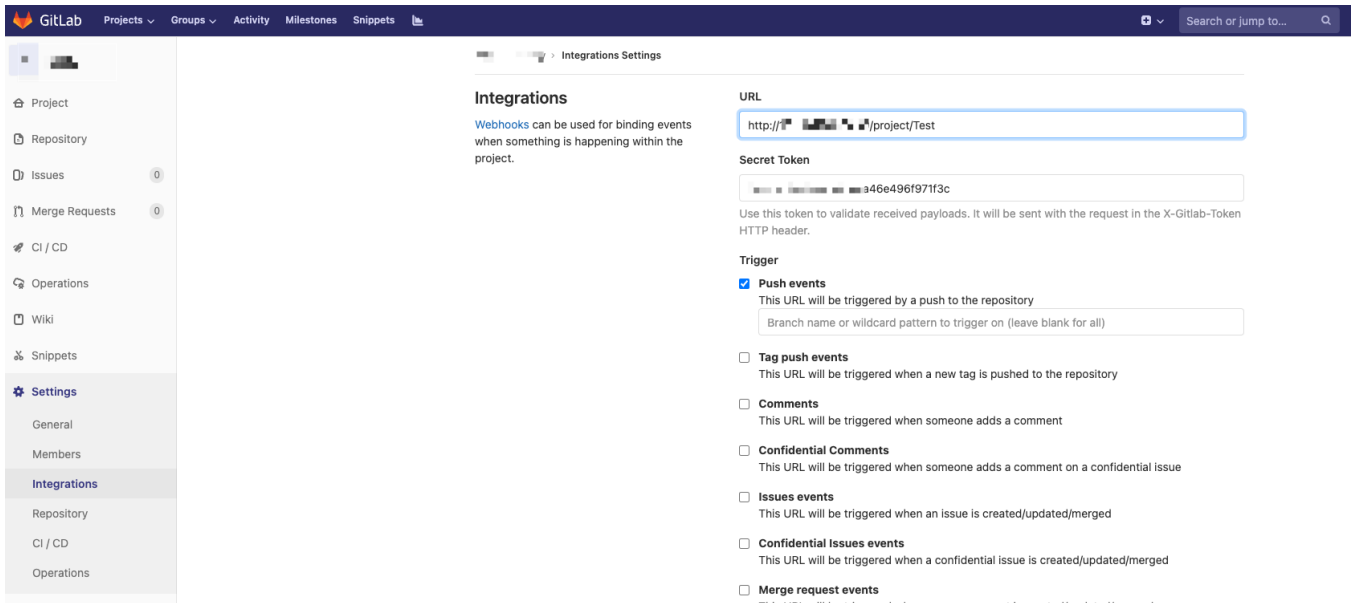
解決辦法：

```
chmod 400 pem_filename.pem
```

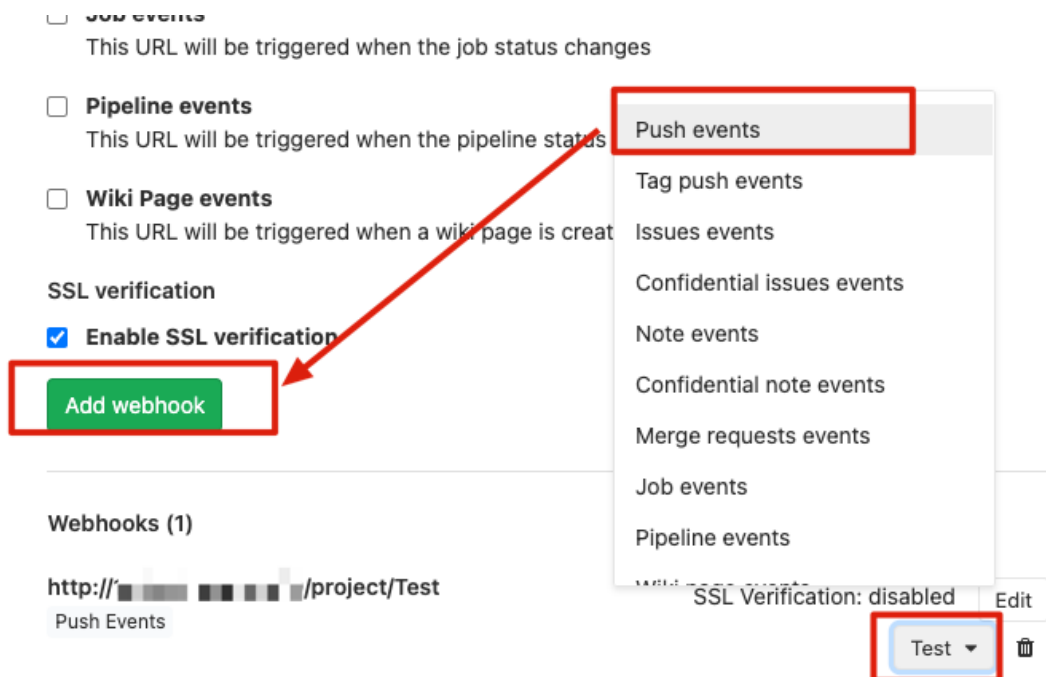
【GitLab】push 觸發pipeline

Gitlab 專案的 settings -> integrations

設定 【url】 【token】 (見下方jenkins 設定)

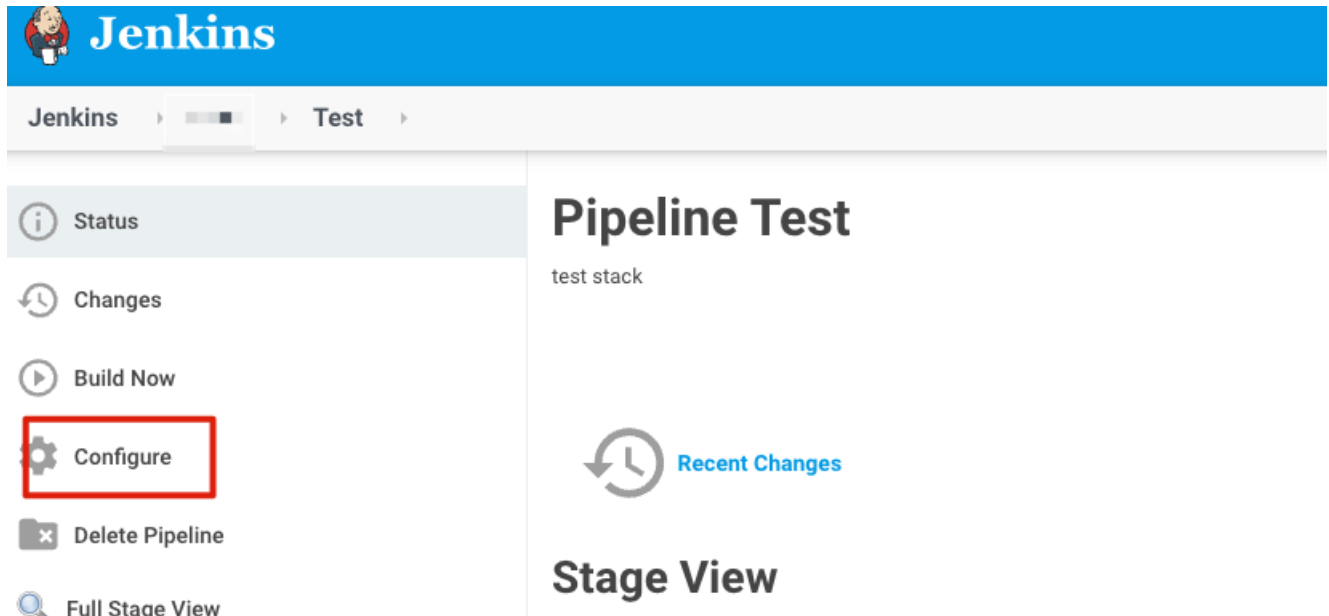


點選下方 【Test】 【Push events】 測試

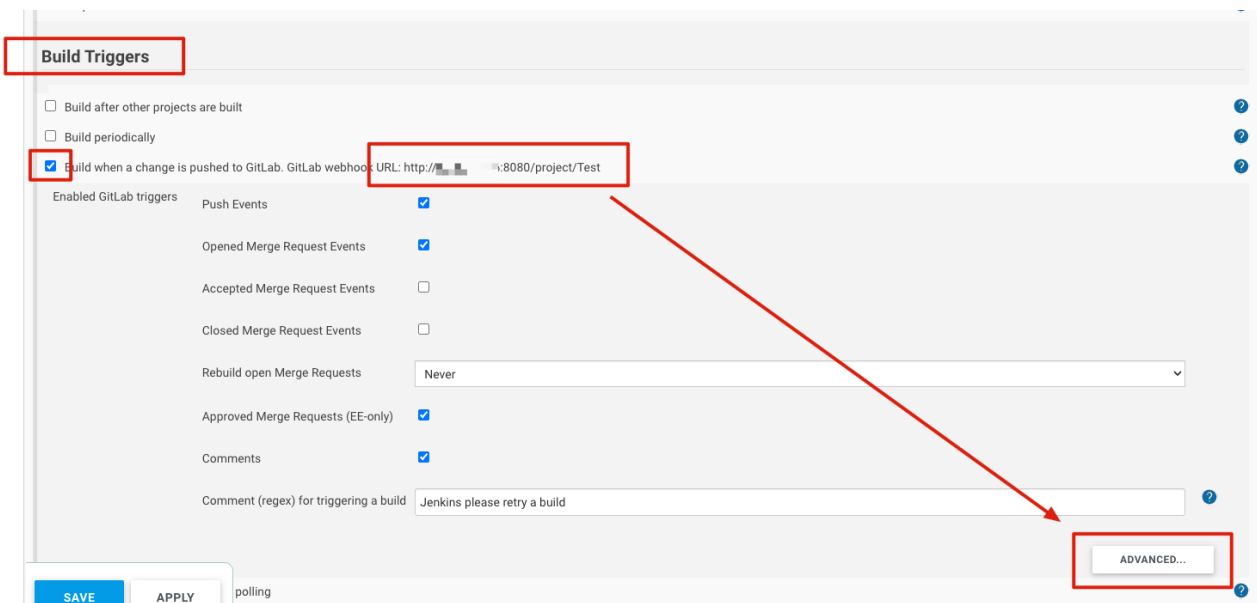


成功上方會出現藍色200字樣

Jenkins (Url, token)



The image shows the Jenkins web interface for configuring a pipeline. The top navigation bar is blue with the Jenkins logo and name. Below it, a breadcrumb trail shows 'Jenkins' and 'Test'. The left sidebar contains several options: 'Status', 'Changes', 'Build Now', 'Configure' (highlighted with a red box), 'Delete Pipeline', and 'Full Stage View'. The main content area is titled 'Pipeline Test' and shows 'test stack'. Below this, there is a 'Recent Changes' section with a clock icon. The 'Stage View' section is partially visible at the bottom.



The image shows the 'Build Triggers' configuration page in Jenkins. The 'Build Triggers' section is highlighted with a red box. It contains several options: 'Build after other projects are built' (unchecked), 'Build periodically' (unchecked), and 'Build when a change is pushed to GitLab, GitLab webhook' (checked, highlighted with a red box). The 'Build when a change is pushed to GitLab, GitLab webhook' option has a URL field set to 'http://[redacted]:8080/project/Test'. Below this, there is a table of 'Enabled GitLab triggers' with checkboxes for 'Push Events', 'Opened Merge Request Events', 'Accepted Merge Request Events', 'Closed Merge Request Events', 'Rebuild open Merge Requests' (set to 'Never'), 'Approved Merge Requests (EE-only)', 'Comments', and 'Comment (regex) for triggering a build' (set to 'Jenkins please retry a build'). A red arrow points from the URL field to the 'ADVANCED...' button, which is also highlighted with a red box. At the bottom, there are 'SAVE' and 'APPLY' buttons, and a 'polling' checkbox.

Approved Merge Requests (EE-only) ☒

Comments ☒

Comment (regex) for triggering a build

Enable [ci-skip] ☒

Ignore WIP Merge Requests ☒

Set build description to build cause (eg. Merge request or Git Push) ☒

Build on successful pipeline events ☐

Pending build name for pipeline

Cancel pending merge request builds on update ☐

Allowed branches

- ☒ Allow all branches to trigger this job
- ☐ Filter branches by name
- ☐ Filter branches by regex
- ☐ Filter merge request by label

Secret token

GENERATE

【Jenkins】 docker-compose

```
version: '3.1'

services:

  jenkins:
    deploy:
      resources:
        limits:
          cpus: '0.50'
          # memory: 500M
    image: jenkins/jenkins:2.462.3-lts-jdk11
    # image: jenkins/jenkins:lts-jdk11
    # restart: always
    container_name: com_jenkins
    environment:
      - TZ=Asia/Taipei #設定時區
    volumes:
      - ./data/jenkins:/var/jenkins_home
      # - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - 8096:8080
      - 8097:50000
    networks:
      - jenkins
  networks:
    jenkins:

# /var/jenkins_home/secrets/initialAdminPassword
```

【Jenkins】如何執行排程作業

- [How to Schedule a Jenkins Job to Run Every Hour](#)
- [Jenkins排程作業](#)

1. Jenkins 作業可以手動執行，可以由某個Web 鉤子觸發，也可以定時運行；
2. 定時執行Jenkins 作業，是經由「宣告式指令產生器」中的「triggers: Triggers」指令，「cron: Build periodically」完成的；
3. 這裡的 cron 與*nix 中 cron 有略微差異，包括了 H 及 @midnight 等語法，其中 H 指 hash，可避免在某個時刻過多的作業（作業競爭），@midnight 這樣的寫法也可以避免作業競爭；
4. TZ=Asia/Taipei 這種寫法可以給Jenkins cron 加入時區；
5. 將聲明式指令產生器產生的程式碼片段：

```
pipeline {
  agent any
  environment {
    TZ = "Asia/Taipei"
  }
  triggers {
    cron '* * * * *'
  }
  stages {
    stage('demo') {
      steps {
        // do....
      }
    }
  }
}
```

【Jenkins】執行git專案上的pipeline

參考：<https://ithelp.ithome.com.tw/m/articles/10287403>

- Definition: Pipeline script from SCM
- SCM: Git
- Repository URL: `https://github.com/ben4932042/jenkins-ithome.git`
- Branch Specifier (blank for 'any'): `main`
- **Script Path:** `jobs/ithome-day3-sample-job/Jenkinsfile` => pipeline 檔案

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

`http://[redacted]-alerts.git`

Credentials ?

+ Add

Advanced

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

`master`

Add Branch

Repository browser ?

gitlab

URL ?

`http://[redacted]/fetch_apierror_[redacted].groovy`

Version ?

Additional Behaviours

Add

Script Path ?

`jenkinsfile/[redacted].groovy`

☒ Lightweight checkout ?

Pipeline Syntax

Save Apply