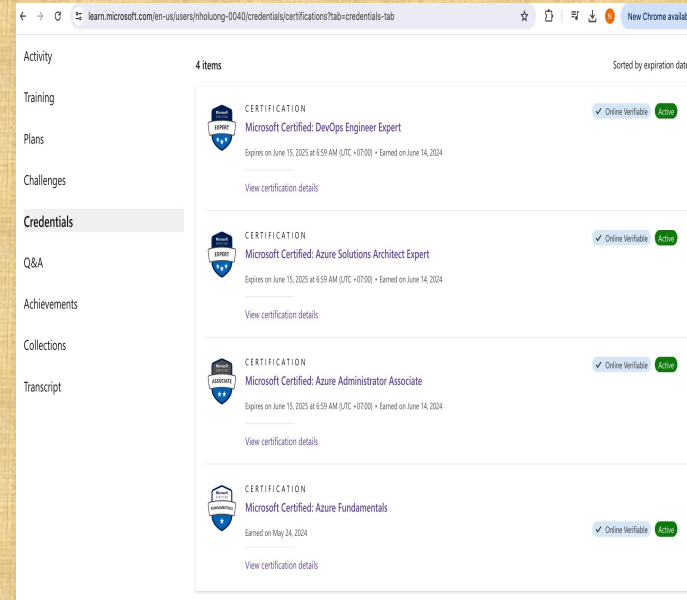
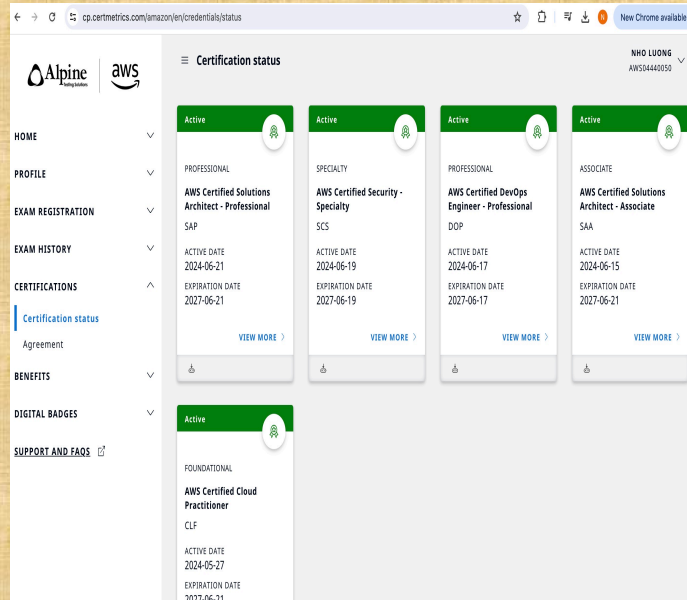


Jenkins Pipeline

Author: Nho Luong

Skill: DevOps Engineer Lead



What is a Jenkinsfile?

A Jenkinsfile is a text file that stores the entire workflow as code and it can be checked into a SCM on your local system. How is this advantageous? This enables the developers to **access, edit and check the code at all times**.

The Jenkinsfile is written using the Groovy DSL and it can be created through a text/groovy editor or through the configuration page on the Jenkins instance. It is written based on two syntaxes, namely:

1. **Declarative pipeline syntax**
2. **Scripted pipeline syntax**

Declarative pipeline is a relatively new feature that supports the pipeline as code concept. It makes the pipeline code easier to read and write. This code is written in a Jenkinsfile which can be checked into a source control management system such as Git.

Pipeline concepts

- **Pipeline**

This is a user defined block which contains all the processes such as build, test, deploy, etc. It is a collection of all the stages in a Jenkinsfile. All the stages and steps are defined within this block. It is the key block for a declarative pipeline syntax.

```
pipeline {  
  
}
```

- **Node**

A node is a machine that executes an entire workflow. It is a key part of the scripted pipeline syntax.

```
node {  
  
}
```

There are various mandatory sections which are common to both the declarative and scripted pipelines, such as stages, agent and steps that must be defined within the pipeline. These are explained below:

- **Agent**

An agent is a directive that can run multiple builds with only one instance of Jenkins. This feature helps to distribute the workload to different agents and execute several projects within a single Jenkins instance. It instructs Jenkins to **allocate an executor** for the builds.

A single agent can be specified for an entire pipeline or specific agents can be allotted to execute each stage within a pipeline. Few of the parameters used with agents are:

- **Any**

Runs the pipeline/ stage on any available agent.

- **None**

This parameter is applied at the root of the pipeline and it indicates that there is no global agent for the entire pipeline and each stage must specify its own agent.

▪ Label

Executes the pipeline/stage on the labelled agent.

▪ Docker

This parameter uses docker container as an execution environment for the pipeline or a specific stage. In the below example I'm using docker to pull an ubuntu image. This image can now be used as an execution environment to run multiple commands.

```
pipeline {  
  agent {  
    docker {  
      image 'ubuntu'  
    }  
  }  
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

- **Stages**

This block contains all the work that needs to be carried out. The work is specified in the form of stages. There can be more than one stage within this directive. Each stage performs a specific task. In the following example, I've created multiple stages, each performing a specific task.

```
pipeline {  
    agent any  
    stages {  
        stage ('Build') {  
            ...  
        }  
        stage ('Test') {  
            ...  
        }  
        stage ('QA') {  
            ...  
        }  
        stage ('Deploy') {  
            ...  
        }  
        stage ('Monitor') {  
            ...  
        }  
    }  
}
```

- **Steps**

A series of steps can be defined within a stage block. These steps are carried out in sequence to execute a stage. There must be at least one step within a steps directive. In the following example I've implemented an echo command within the build stage. This command is executed as a part of the 'Build' stage.

```
pipeline {
    agent any
    stages {
        stage ('Build') {
            steps {
                echo 'Running build phase...'
            }
        }
    }
}
```


Creating your first Jenkins pipeline.

Step 1: Log into Jenkins and select 'New item' from the dashboard.




Step 2: Next, enter a name for your pipeline and select 'pipeline' project. Click on 'ok' to proceed.

Jenkins > All >


Enter an item name

PIPELINE DEMO


» Required field



Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.



Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



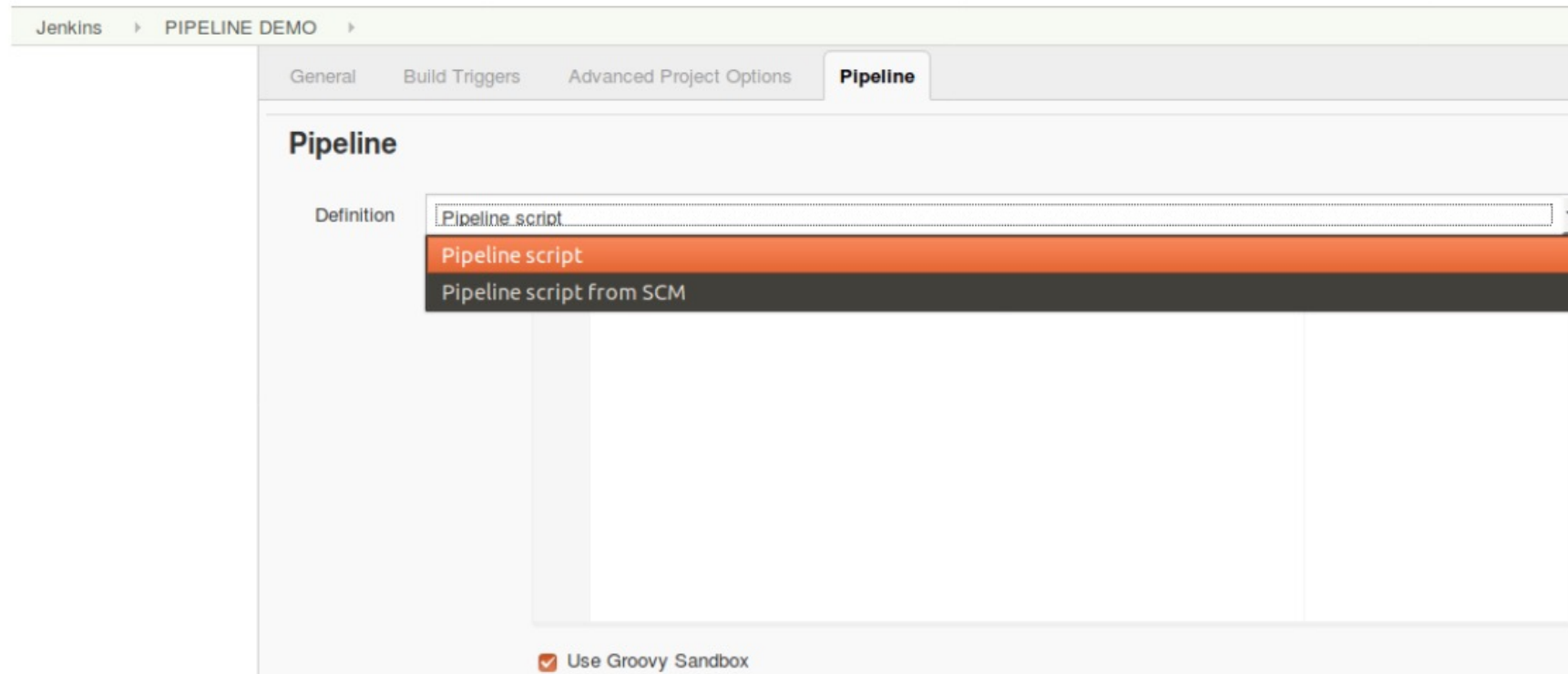
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Enter the project name – Jenkins Pipeline Tutorial

Author: Nho Luong

Skill: DevOps Engineer Lead

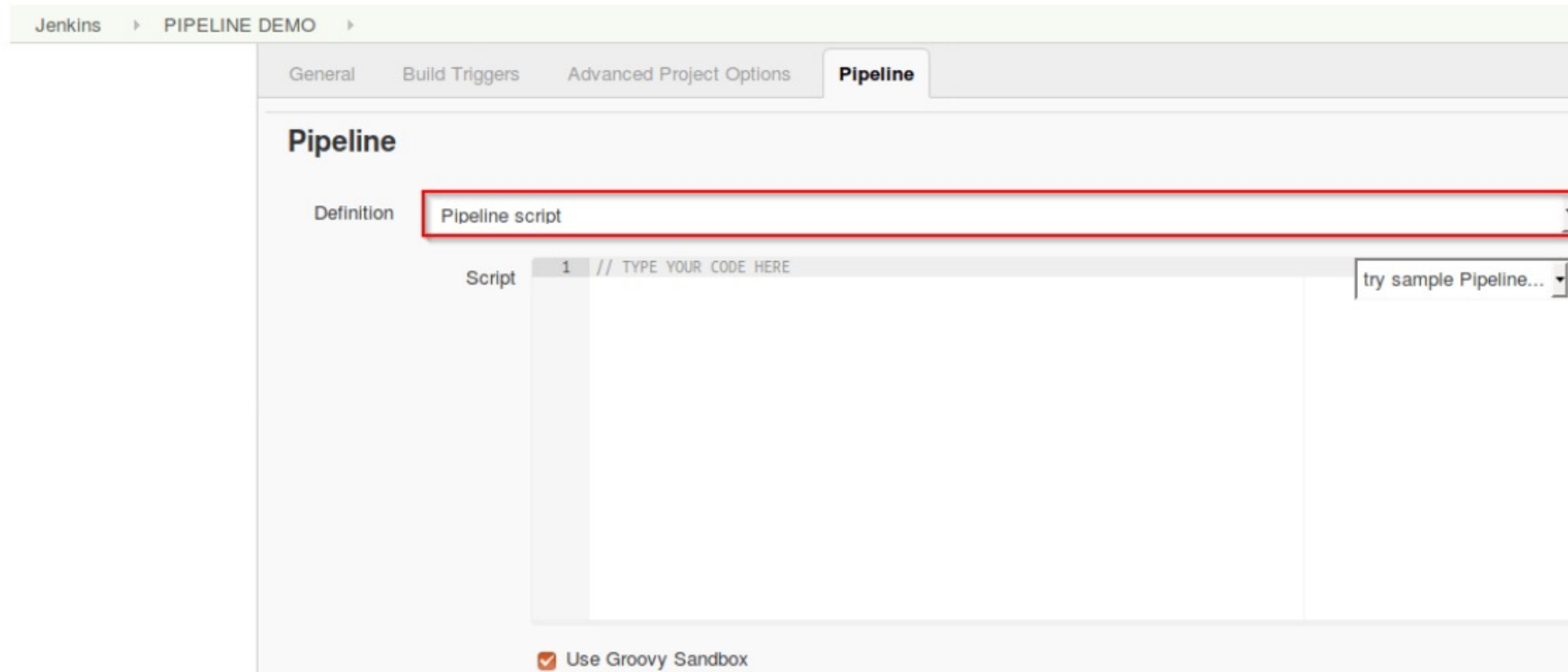
Step 3: Scroll down to the pipeline and choose if you want a declarative pipeline or a scripted one.



The screenshot shows the Jenkins web interface for configuring a pipeline. The breadcrumb navigation at the top reads 'Jenkins > PIPELINE DEMO >'. Below this, there are four tabs: 'General', 'Build Triggers', 'Advanced Project Options', and 'Pipeline', with the 'Pipeline' tab currently selected. The main section is titled 'Pipeline'. Under the 'Definition' label, there is a dropdown menu. The dropdown is open, showing three options: 'Pipeline script' (highlighted in orange), 'Pipeline script', and 'Pipeline script from SCM'. At the bottom of the configuration area, there is a checkbox labeled 'Use Groovy Sandbox' which is checked.

Declarative or scripted pipeline – Jenkins Pipeline Tutorial

Step 4a: If you want a scripted pipeline then choose 'pipeline script' and start typing your code.



The screenshot shows the Jenkins web interface for configuring a pipeline. At the top, there's a breadcrumb trail: 'Jenkins > PIPELINE DEMO >'. Below this is a tabbed interface with four tabs: 'General', 'Build Triggers', 'Advanced Project Options', and 'Pipeline'. The 'Pipeline' tab is selected. Under the 'Pipeline' tab, the title 'Pipeline' is displayed. Below the title, there's a 'Definition' section. In this section, a dropdown menu is open, showing 'Pipeline script' as the selected option. This dropdown is highlighted with a red rectangular border. Below the dropdown, there's a 'Script' section. It contains a text area with a line number '1' and the text '// TYPE YOUR CODE HERE'. To the right of the text area is a button labeled 'try sample Pipeline...'. At the bottom of the 'Script' section, there's a checkbox labeled 'Use Groovy Sandbox' which is checked.

Scripted Pipeline – Jenkins Pipeline Tutorial

Step 4b: If you want a declarative pipeline then select 'pipeline script from SCM' and choose your SCM. In my case I'm going to use Git throughout this demo. Enter your repository URL.

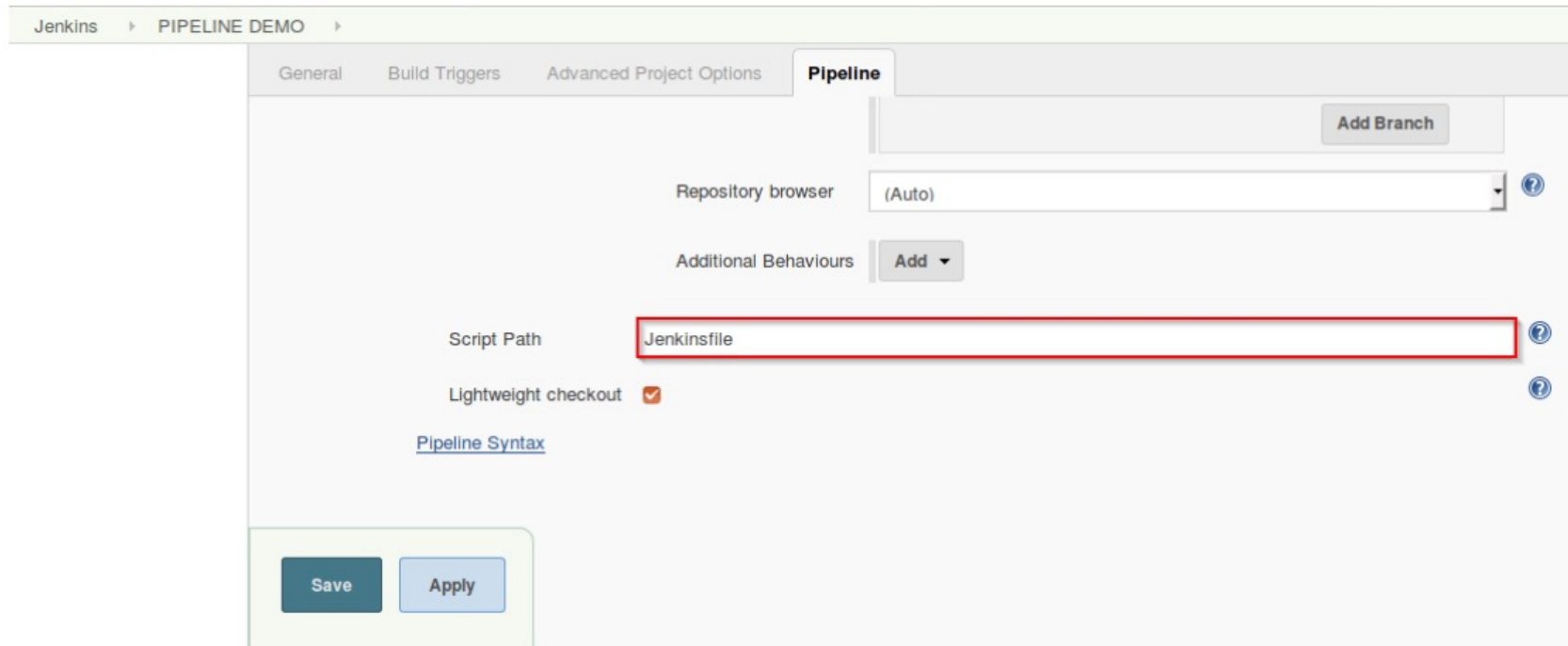
The screenshot shows the Jenkins configuration interface for a pipeline. The breadcrumb navigation at the top indicates the path: Jenkins > PIPELINE DEMO > Pipeline. The 'Pipeline' tab is selected, showing options for General, Build Triggers, Advanced Project Options, and Pipeline. Under the 'Pipeline' section, the 'Definition' is set to 'Pipeline script from SCM' and the 'SCM' is set to 'Git'. The 'Repositories' section contains a single entry with the 'Repository URL' set to 'https://github.com/Zulaikha12/git-test.git'. A red error message, 'Please enter Git repository.', is displayed below the URL field. The 'Credentials' dropdown is set to '- none -' with an 'Add' button next to it. At the bottom right of the configuration area, there are buttons for 'Advanced...' and 'Add Repository'.

Declarative pipeline – Jenkins Pipeline Tutorial

Author: Nho Luong

Skill: DevOps Engineer Lead

Step 5: Within the script path is the name of the Jenkinsfile that is going to be accessed from your SCM to run. Finally click on 'apply' and 'save'. You have successfully created your first Jenkins pipeline.

A screenshot of the Jenkins web interface showing the 'Pipeline' configuration page for a project named 'PIPELINE DEMO'. The breadcrumb navigation at the top shows 'Jenkins > PIPELINE DEMO >'. Below this, there are tabs for 'General', 'Build Triggers', 'Advanced Project Options', and 'Pipeline', with 'Pipeline' being the active tab. The 'Pipeline' section contains several fields: 'Repository browser' is set to '(Auto)' with a dropdown arrow and a help icon; 'Additional Behaviours' has an 'Add' button with a dropdown arrow; 'Script Path' is set to 'Jenkinsfile' and is highlighted with a red rectangular border, with a help icon to its right; 'Lightweight checkout' is checked with an orange checkbox and a help icon to its right. Below these fields is a link labeled 'Pipeline Syntax'. At the bottom left of the configuration area, there are two buttons: 'Save' (dark blue) and 'Apply' (light blue).

Jenkins > PIPELINE DEMO >

General Build Triggers Advanced Project Options **Pipeline**

Add Branch

Repository browser (Auto) ?

Additional Behaviours Add ▾

Script Path Jenkinsfile ?

Lightweight checkout ☒ ?

[Pipeline Syntax](#)

Save Apply

Script path – Jenkins Pipeline Tutorial

Declarative Pipeline Demo

The first part of the demo shows the working of a declarative pipeline. Refer the above 'Creating your first Jenkins pipeline' to start. Let me start the demo by explaining the code I've written in my Jenkinsfile.

Since this is a declarative pipeline, I'm writing the code locally in a file named 'Jenkinsfile' and then pushing this file into my global git repository. While executing the 'Declarative pipeline' demo, this file will be accessed from my git repository. The following is a simple demonstration of building a pipeline to run multiple stages, each performing a specific task.

- The declarative pipeline is defined by writing the code within a pipeline block. Within the block I've defined an agent with the tag 'any'. This means that the pipeline is run on any available executor.
- Next, I've created four stages, each performing a simple task.
- Stage one executes a simple echo command which is specified within the 'steps' block.
- Stage two executes an input directive. This directive allows to **prompt a user input** in a stage. It displays a message and waits for the user input. If the input is approved, then the stage will trigger further deployments.
- In this demo a simple input message 'Do you want to proceed?' is displayed. On receiving the user input the pipeline either proceeds with the execution or aborts.



Thank You