

JENKINS

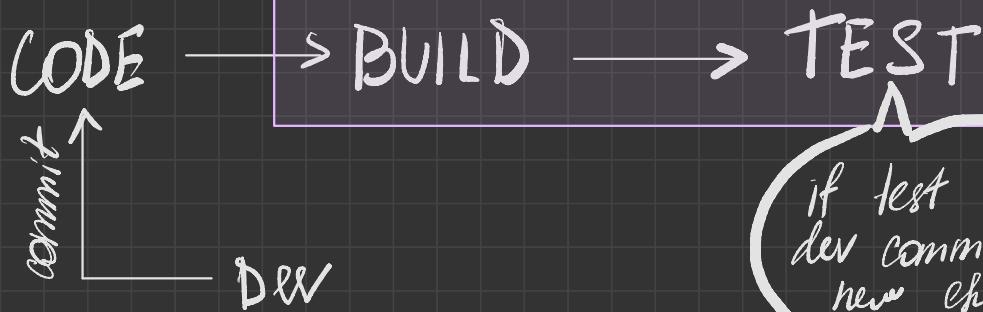
To run with Docker:

```
docker run -p 8080:8080 -p 50000:50000 --restart=on-failure -v  
jenkins_home:/var/jenkins_home jenkins/its-jdk17
```

- ↳ save admin pass
- ↳ go to localhost:8080
- ↳ install suggested plugins
- ↳ continue as admin

CI

App



- Not efficient to build on local machine.
- ↳ Jenkins turns the steps into pipeline

When dev commits, Jenkins scans for new commits and starts a build

CD

Where devops start getting involved

CD



CREATE NEW JOB

Enter an item name

» Required field



Freestyle project

blank slate

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.



Pipeline

e.g. CI/CD process

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

if we have to run the same job in a project with slightly diff params

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



Folder

a way to organise jobs

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.



Multibranch Pipeline

ref to github organization

Creates a set of Pipeline projects according to detected branches in one SCM repository.



Organization Folder

Creates a set of multibranch project subfolders by scanning for repositories.

If you want to create a new item from other existing, you can use this option:



Copy from

Type to autocomplete

OK

CREATE PIPELINE EDITOR

The screenshot shows the Jenkins Pipeline Editor interface. At the top, there's a navigation bar with 'Dashboard > My Pipeline > Configuration'. On the left, a sidebar titled 'Configure' lists 'General', 'Advanced Project Options' (which is selected), and 'Pipeline'. The main content area is titled 'Advanced Project Options' with a dropdown set to 'Advanced'. It contains a 'Pipeline' section with a 'Definition' dropdown set to 'Pipeline script'. Below this is a code editor window titled 'Script' with a single line of Groovy code: '1'. To the right of the code editor is a placeholder text 'try sample Pipeline...'. Underneath the code editor is a checkbox labeled 'Use Groovy Sandbox' which is checked. At the bottom of the page are two buttons: 'Save' and 'Apply'.

It's better to use **YAML** code bc
the pipeline editor is bad in
Jenkins

CREATE PIPELINE

```
❶ Jenkinsfile
1 // any jenkins pipeline starts with pipeline statement
2 pipeline {
3     // agent specifies where the pipeline will be executed
4     agent any
5     // stages block contains all the stages in the pipeline
6     stages {
7         // jenkins stage to clean up the workspace, because when jenkins runs the job it creates a folder
8         // better to put in the beginning otherwise if the job fails, the workspace won't be cleaned up
9         stage("Clean Up"){
10             steps {
11                 deleteDir() // deletes the workspace
12             }
13         }
14         // stage to clone the repository
15         stage("Clone Repo"){
16             steps {
17                 sh "git clone https://github.com/jenkins-docs/simple-java-maven-app.git"
18             }
19         }
20         // stage to build
21         stage("Build"){
22             steps {
23                 // each steps reverts back to the original workspace
24                 // change directory to the cloned repository
25                 // we don't use cd because it's persistent and every step will be inside that directory
26                 dir("simple-java-maven-app"){
27                     // the following builds java file and cleans libraries/binaries that may have been created before
28                     sh "mvn clean package"
29                 }
30             }
31         }
32     }
33     stage("Test"){
34         steps {
35             dir("simple-java-maven-app"){
36                 // run tests
37                 sh "mvn test"
38             }
39         }
40     }
41 }
42 }
```

→ then copy/paste to Jenkins

INSTALL MAVEN

1. docker container ls

↳ id of jenkins container

2. docker exec -it -u root id /bin/bash

3. apt-get update

4. apt-get install maven

CREATE PIPELINE

BUILD TRIGGERS

→ they say when to run this job

Build Triggers

- Build after other projects are built ?
- Build periodically ?
e.g. cleanup job (morning/evening)
- GitHub hook trigger for GITScm polling ?
e.g. runs the job when the change is made to the source code
- Poll SCM ?
- Quiet period ?
- Trigger builds remotely (e.g., from scripts) ?

CREATE PIPELINE

BUILD / REPLAY

The screenshot shows the Jenkins Pipeline Stage View for a project. The stage logs show a successful git clone operation. The stage view displays average times for Clean Up (152ms), Clone Repo (1s), Build (15s), and Test (3s). A handwritten note in a purple speech bubble says: "you can check details of every step by clicking on it".

Stage Logs (Clone Repo)

```
Shell Script -- git clone https://github.com/jenkins-docs/simple-java-maven-app.git (self time 1s)
+ git clone https://github.com/jenkins-docs/simple-java-maven-app.git
Cloning into 'simple-java-maven-app'...
```

</> Changes Add description
▷ Build Now Disable Project
⚙ Configure
Delete Pipeline
🔍 Full Stage View
🕒 Rename
Pipeline Syntax
Build History trend
Filter...
#1 Apr 22, 2024, 6:14 PM
Atom feed for all Atom feed for failures

Stage View

Average stage times:
(Average full run time: ~22s)

#1	Apr 22, 20:14	No Changes		
Clean Up	152ms	1s	15s	3s
Clone Repo	152ms	1s	15s	3s
Build	152ms	1s	15s	3s
Test	152ms	1s	15s	3s

Permalinks

REST API Jenkins 2.440.2

The screenshot shows the Jenkins Pipeline Replay page for build #1. It includes a Main Script section with Jenkins pipeline code, a Pipeline Syntax section, and a Run button. A handwritten note in orange says: "good tool for debugging".

Dashboard > My Pipeline > #1 > Replay

Status admin log out

Replay #1

Allows you to replay a Pipeline build with a modified script. If any load steps were run, you can also modify the scripts they loaded.

Main Script

```
1 // any jenkins pipeline starts with pipeline statement
2 pipeline {
3     // agent specifies where the pipeline will be executed
4     agent any
5     // stages block contains all the stages in the pipeline
6     stages {
7         // Jenkins stage to clean up the workspace, because when Jenkins runs the job it creates a folder
8         // better to put in the beginning otherwise if the job fails, the workspace won't be cleaned up
9         stage("Clean Up"){
10             steps {
11                 deleteDir() // deletes the workspace
12             }
13         }
14         // stage to clone the repository
15         stage("Clone Repo"){
```

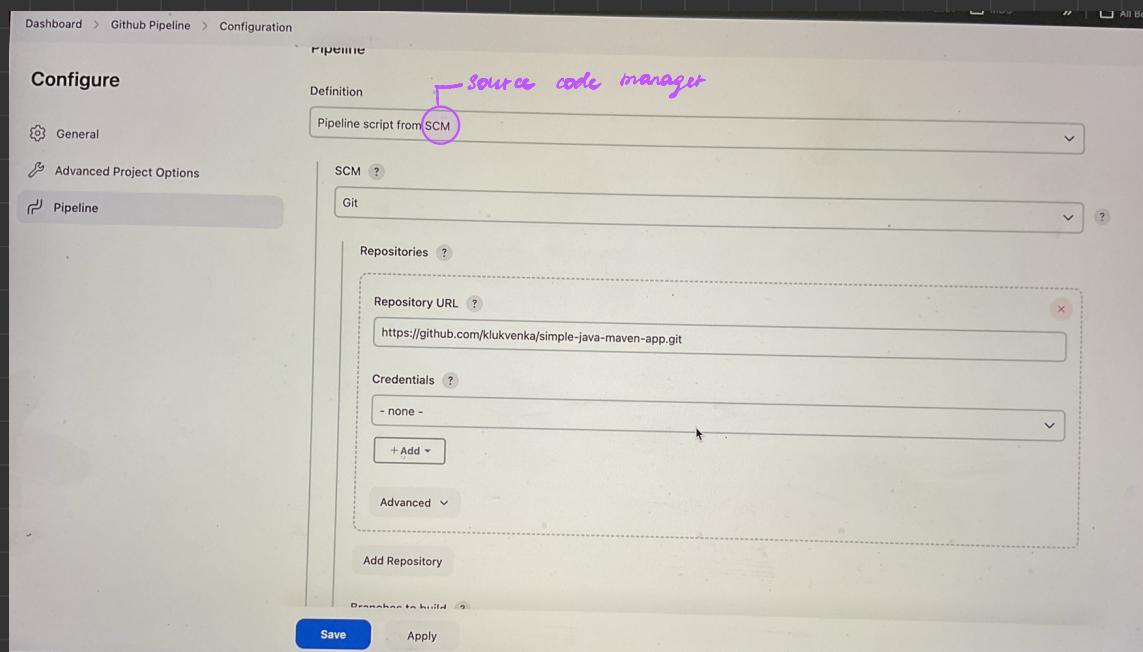
good tool for debugging

Pipeline Syntax

Run

PIPELINE . GITHUB

* You should already have a github repo with jenkinsfile in it



1. Now jenkins automatically will clone the repo
2. Then will load jenkinsfile that creates the pipeline

PIPELINE . GITHUB

auto builds

* H means it would pick a random hour

Build Triggers

Build after other projects are built ?

Build periodically ?

Schedule ?

H/2 * * * *

Would last have run at Monday, April 22, 2024 at 8:19:45 PM Coordinated Universal Time; would next run at Monday, April 22, 2024 at 8:19:45 PM Coordinated Universal Time.

MULTIBRANCH PIPELINE

Allows to implement different
Jenkinsfiles for different branches
of same project

PARAMETERIZED PIPELINES

Allows to get input before the job runs

- ↳ checkbox
- ↳ user input
- ↳ dropdowns
- ↳ ...

BOOLEAN

① Create pipeline

Advanced Project Options

Advanced

Pipeline

Definition

Pipeline script

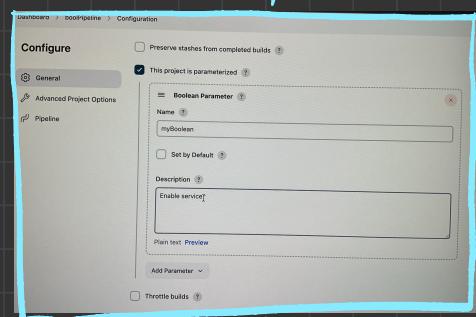
```
Script ?  
1+ pipeline {  
2+   agent any  
3+  
4+   parameters {  
5+     booleanParam(defaultValue: false, description: "Enable service?", name: "myBoolean")  
6+   }  
7+  
8+   stages {  
9+     stage("Demo"){  
10+       steps {  
11+         echo "booleanParam is set to: ${params.myBoolean}"  
12+       }  
13+     }  
14+   }  
15+ }
```

Use Groovy Sandbox

Pipeline Syntax

Save Apply

② Build w/o params for the 1st time → ③ Then you see params in config (Jenkins picked up the info)



↳ ④ Click "Build with params"

PARAMETERIZED PIPELINES

STRING

```
params > input > Jenkinsfile
1 pipeline []
2   agent any
3
4   parameters {
5     // there is also a text parameter that supports multi-line text
6     string(defaultValue: "TEST", description: "Which environment to deploy in?", name: "deployEnv")
7   }
8
9   stages {
10     stage("Demo"){
11       steps {
12         echo "string is set to: ${params.deployEnv}"
13       }
14     }
15   }
16 }
```

DROPDOWN

```
params > choice > Jenkinsfile
1 pipeline {
2   agent any
3
4   parameters {
5     choice(choices: ["TEST", "DEV", "QA", "PRE_PROD", "PROD"], description: "Which environment to deploy in?", name: "deployEnv")
6   }
7
8   stages {
9     stage("Demo"){
10       steps {
11         echo "Choice is set to: ${params.deployEnv}"
12       }
13     }
14   }
15 }
```

VARIABLES

This is how to use regular variables:

```
variables > Jenkinsfile
1 pipeline {
2   agent any
3
4   environment {
5     // define variables here
6     def myString = "Hello World"
7     def myNumber = 10
8     def myBool = true
9   }
10
11  stages {
12    stage("Demo") {
13      steps {
14        echo "myString: ${myString}"
15        echo "myNumber: ${myNumber}"
16        echo "myBool: ${myBool}"
17      }
18    }
19  }
20 }
```

ENV VARS

<https://www.jenkins.io/doc/book/pipeline/jenkinsfile/#using-environment-variables>

List of Jenkins env vars available

Script ?

```
1 pipeline {
2   agent any
3   stages {
4     stage('Example') {
5       steps {
6         // e.g. you can use build number as a tag when working with docker
7         echo "Build number: ${env.BUILD_NUMBER}"
8       }
9     }
10   }
11 }
```

GROOVY

Docs →

<https://groovy-lang.org/syntax.html>

Jenkins uses Groovy behind the scenes
Really helpful for complex pipelines

```
params > launch > 🏠 Jenkinsfile
1 pipeline [ ]
2   agent any
3
4   parameters {
5     booleanParam(defaultValue: false, description: "Enable service?", name: "myBoolean")
6   }
7
8   stages {
9     stage("Demo"){
10       steps {
11         // anything with groovy needs to go inside the script block
12         script {
13           if(params.myBoolean == false){
14             currentBuild.result = "SUCCESS"
15             return // return stops the pipeline completely
16           }
17           else {
18             echo "booleanParam is set to: TRUE"
19           }
20         }
21       }
22     }
23   }
24 }
```



FUNCTIONS

```
func > 🏛 Jenkinsfile
1  pipeline {
2    agent any
3
4    env {
5      def myNumber = 123
6    }
7
8    stages{
9      stage("Demo"){
10        steps {
11          myFunc("Hello from the demo stage!")
12        }
13      }
14    }
15  }
16
17  def myFunc(String myText) {
18    echo "myText is set to: ${myText}"
19 }
```

can't be used outside of the pipeline **(SCOPE)**

e.g. can't be used here

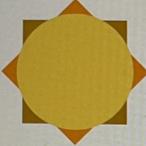
Vars defined inside a stage also can be used only in that stage

MULTILINE BASH

```
1  pipeline {
2    agent any
3
4    stages{
5      stage("Demo"){
6        steps [
7          sh """
8            echo "Hello from the shell"
9            """
10       ]
11     }
12   }
13 }
```

BUILD HEALTH

A useful indication on builds status.
E.g. if goes from Sunny to Rainy we can notice the problem there

Icon	Health
	Sunny , more than 80% of Runs passing
	Partially Sunny , 61% to 80% of Runs passing
	Cloudy , 41% to 60% of Runs passing
	Raining , 21% to 40% of Runs passing
	Storm , less than 21% of Runs passing

CREDS

<https://www.jenkins.io/doc/book/pipeline/jenkinsfile/#handling-credentials>

[There are many guides depending on what platform we are using]

Advice for AWS:

When Jenkins's running on EC2 instances on AWS, instead of using credentials, assign the role to the node

JOB FROM A JOB

```
41 v     stage("Build Remote"){
42 v       steps {
43 v         build 'boolPipeline'
44 v       }
45 v     }
46 v   }
47 }
```



name of the job

To set a parameter value:

```
28 v     stage("Build Remote"){
29 v       steps {
30 v         build job: 'boolPipeline', parameters: [[${class: 'BooleanParameterValue', name: 'myBoolean', value: true}]]}
31 v       }
32 v     }
33 v   }
34 v }
35 }
```

