BIT

12 : Continuous Integration (CI) and Continuous Delivery (CD)

IT6206-Software Quality Assurance

Level III - Semester 6

UCSC

eLC

Overview

- This chapter talks about Continuous Integration (CI) and Continuous Delivery (CD) and discusses Jenkins as a CI and CD tool.

- This chapter will also give an introduction to Version Control Systems (VCS) and discuss Git as an example VCS.

Intended Learning Outcomes

At the end of this lesson, you will be able to:

- Explain what are Continuous Integration (CI) and Continuous Delivery (CD) and their importance.
- Setup Jenkins and create a simple pipeline using Jenkins.
- Explain what is Version Control Systems and their importance.
- Setup and use Git to maintain a repository.

List of sub topics

Introduction to CI and CD tools

12.1 Introduction to CI and CD tools

- **Continuous Integration**
  - Continuous Integration (CI) is a method of continuously verifying the condition of a codebase through automated testing.
  - It is best achieved by integrating CI with version control.

- **Continuous Delivery/Deployment**
  - Approach to regularly deploying artifacts that successfully pass the CI phase to ensure confident around the deployment.

12.1 Introduction to CI and CD tools

- Continuous integration, continuous deployment, and continuous delivery are like vectors that have the same direction, but different magnitude.

- They have the same goal: make our software development and release process faster and more robust.

- The key difference between the three is in the scope of automation applied.

12.1 Introduction to CI and CD tools

CI & CD Phases

CI

- Code gets pushed the to a Version Control System.
- The build server checks the code as soon as it is pushed.
- Developers receive the feedback (passed or failed)
- Code get stored in a repository

CD

- Ensure the reliable release of software.
- Deployments are quick and often.
- Helps in automated deployment.

12.1 Introduction to CI and CD tools

CI & CD Pipelines

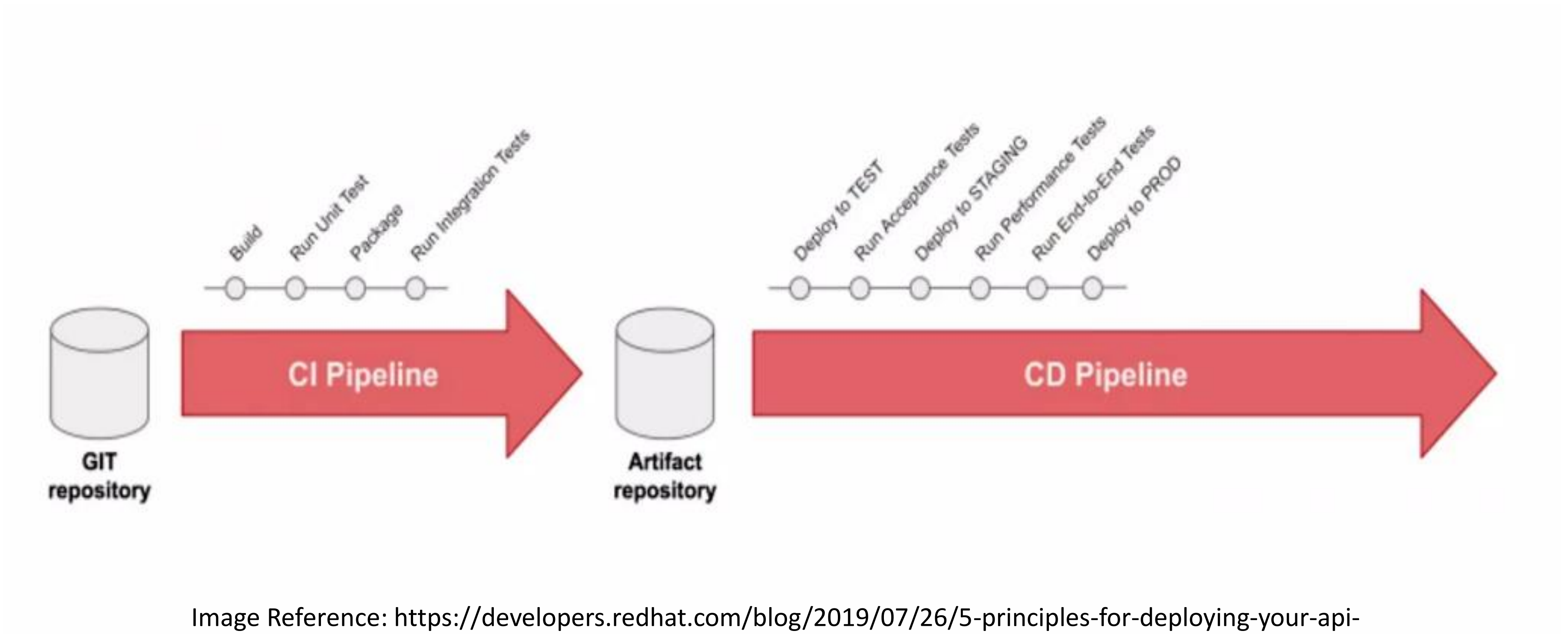


Image Reference: https://developers.redhat.com/blog/2019/07/26/5-principles-for-deploying-your-api-from-a-ci-cd-pipeline#an_api_is_not__just_another_piece_of_software_

12.1 Introduction to CI and CD tools

Signs of a Good CI

- Decoupled stages
  - The CI process should ensure that each step performs a specific and targeted task.
- Repeatable
  - The automation process should be designed in a manner that ensures consistency and repeatability.
  - The tools used for automation should be accessible to both local and remote developers, ensuring parity between different environments.
- Fail fast
  - Fail at the first sign of trouble.

12.1 Introduction to CI and CD tools

Signs of a GOOD CD

- Design with the system in mind
  - Cover as many parts of a deployment as possible (Application, Infrastructure, Configuration, Data)
- Pipelines
  - Continually increase confidence as you move towards production
- Globally unique versions
  - Know the state of the system at any time
  - Be able to demonstrate difference between current and future state

12.1 Introduction to CI and CD tools

- Who should know about CI/CD?
  - Developers
  - QAs
  - System Administrations

12.1 Introduction to CI and CD tools

AppVeyor
argo
Bamboo
BRIGADE
Buildkite
circleci
Skycap
cloudbees
codefresh
CODESHIP
Concourse
ContainerOps
DeployHub
Drone
GITKUBE
GitLab
go
Google Cloud Build
Habitus
Jenkins
JenkinsX
Octopus Deploy
Semaphore
shippable
Spinnaker
TeamCity
Travis CI
HYSCALE wavemaker
weave flux
wercker
XL DEPLOY
ZUUL

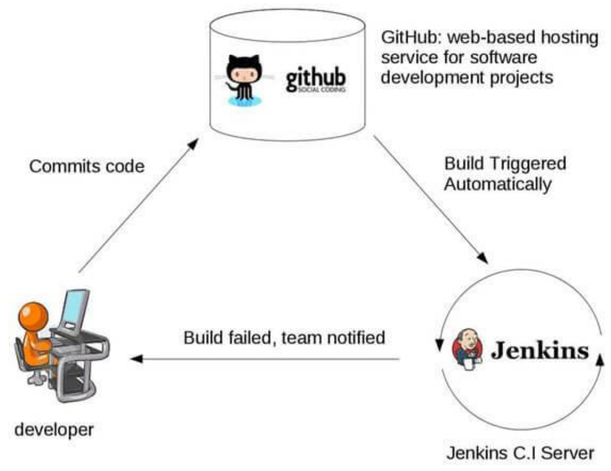
Image Reference: https://www.weave.works/blog/what-cicd-tool-should-i-use

Setting up Jenkins

12.2 Setting up Jenkins

Jenkins

- Jenkins is a widely used open-source tool for continuous integration and building of software development projects.
- It offers the ability to build projects manually, periodically, or automatically, and comes equipped with hundreds of plugins to support deployment and automation.
- Jenkins is utilized by teams of various sizes for projects written in different languages,
- Jenkins is programmed using Java.


github
GitHub: web-based hosting service for software development projects
Commits code
Build Triggered Automatically
Build failed, team notified
Jenkins
developer
Jenkins C.I Server

Reference: https://www.numpyninja.com/post/looking-to-understand-what-jenkins-is-want-to-get-hands-on-jenkins-then-this-blog-will-help-you

12.2 Setting up Jenkins

Jenkins Plugins

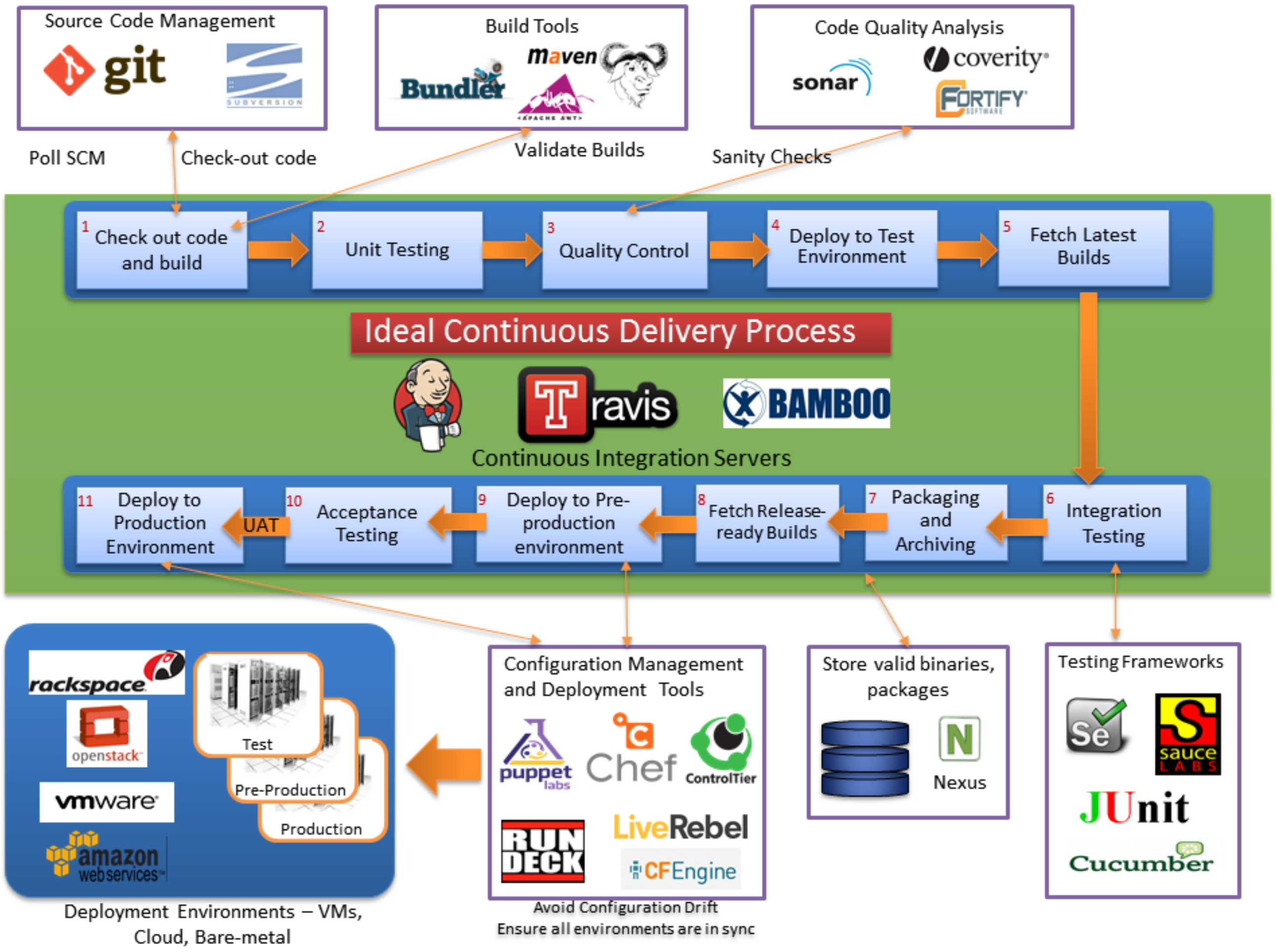
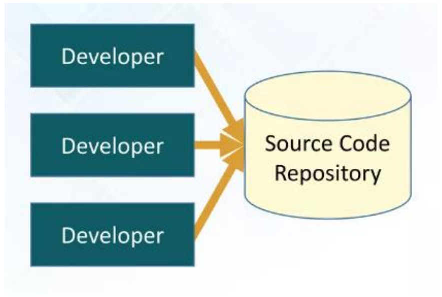

Source Code Management
git
Build Tools
maven
Bundler
Code Quality Analysis
coverity
sonar
FORTIFY
Poll SCM
Check-out code
Validate Builds
Sanity Checks
1 Check out code and build
2 Unit Testing
3 Quality Control
4 Deploy to Test Environment
5 Fetch Latest Builds
Ideal Continuous Delivery Process
Travis
BAMBOO
Continuous Integration Servers
11 Deploy to Production Environment
UAT
10 Acceptance Testing
9 Deploy to Pre-production environment
8 Fetch Release-ready Builds
7 Packaging and Archiving
6 Integration Testing
rackspace
openstack
vmware
amazon webservices
Test
Pre-Production
Production
Deployment Environments – VMs, Cloud, Bare-metal
Configuration Management and Deployment Tools
puppet labs
Chef
ControlTier
RUN DECK
LiveRebel
CFEngine
Avoid Configuration Drift
Ensure all environments are in sync
Store valid binaries, packages
Nexus
Testing Frameworks
Se
sauce LABS
JUnit
Cucumber

Image Reference: http://devops.pm/booting-on-devops/

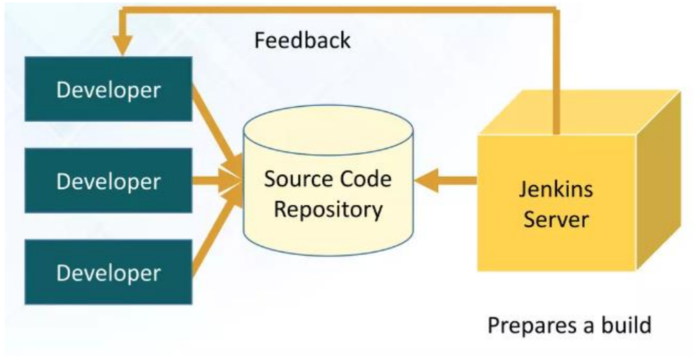
12.2 Setting up Jenkins

CI and CD using Jenkins

- Source code changes will be committed by the developer.

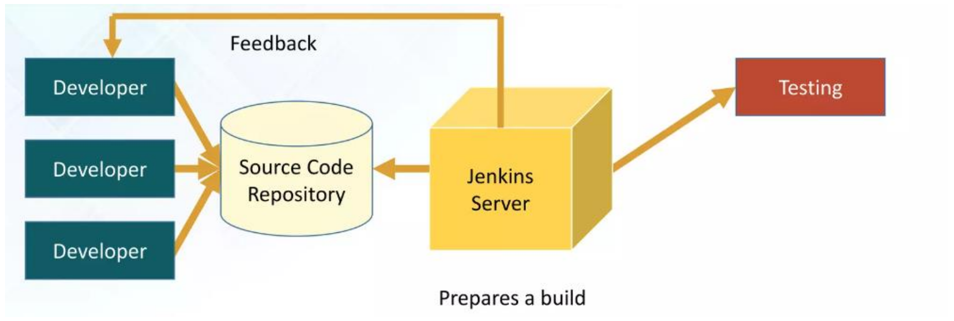


- CI server will pull that code and trigger a build.

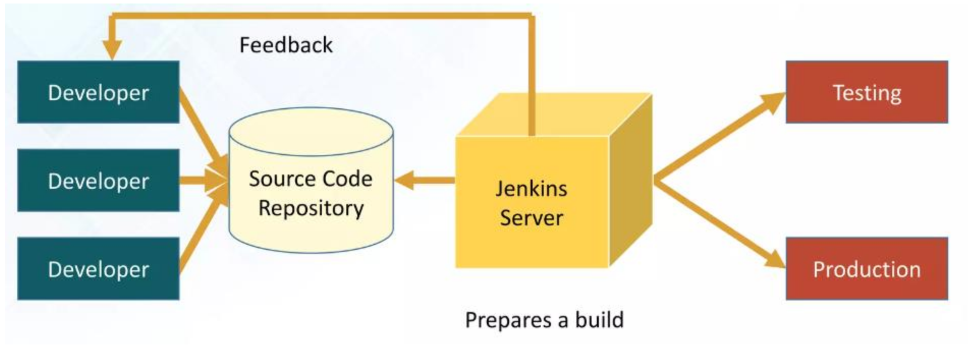


12.2 Setting up Jenkins

- Build application will then be developed for testing on the testing server.



- After testing, it will then be deployed on the production server. The concerned teams are constantly notified about the build and the results.



12.2 Setting up Jenkins

Jenkins' Master and Slave Architecture

- The tester has to ensure that every new build is tested in different web browsers and operating systems to avoid any issues.
- It is not feasible to manually test each new build on various platforms, as it is time-consuming and expensive.
- To overcome this, Jenkins use master and slave architecture to handle multiple machines and distribute the workload automatically, creating different build sections that support all necessary environments for building and testing.

12.2 Setting up Jenkins

Jenkins’ Master and Slave Architecture

Master:

- Schedule build jobs and dispatch to to the slaves for the job execution.
- Monitor the slaves and record the results.
- Can also execute build jobs directly.

Slave:

- Execute build jobs given by the master.

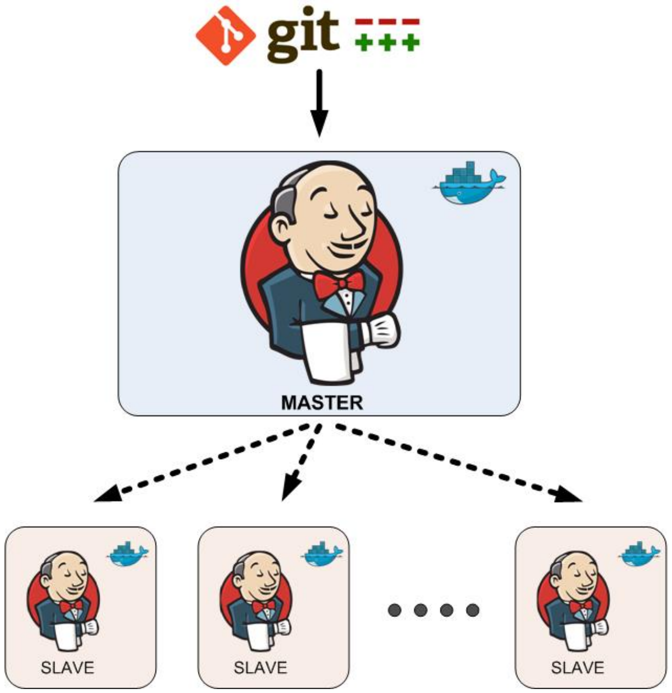


Image Reference: https://nimblehq.co/blog/dockerised-jenkins-for-mobile-testing

12.2 Setting up Jenkins

Prerequisites for Installing Jenkins

JDK	JDK 1.7 or above
Memory	2 GB RAM (recommended)
Disk Space	No minimum requirement. However, all builds will be stored on the Jenkins machines, it has to be ensured that sufficient disk space is available for build storage.
Operating System Version	Jenkins can be installed on Windows, Ubuntu/Debian, Red Hat/Fedora/CentOS, Mac OS X, openSUSE, FReeBSD, OpenBSD, Gentoo.
Java Container	The WAR file can be run in any container that supports Servlet 2.4/JSP 2.0 or later.(An example is Tomcat 5).

12.2 Setting up Jenkins

Installing

- Download Jenkins: Open the official website of Jenkins : https://jenkins.io/index.html Click on Download Jenkins and download the LTS Release.

- From the command prompt, browse to the directory where the jenkins.war file is present. Run the following command:

```
java -jar jenkins.war
```

- Extraction of the war file is done by an embedded webserver called winstone.

12.2 Setting up Jenkins

- Jenkins by default runs on port 8080, but we can run it on different port by issuing following command:

```
java -jar jenkins.war --httpPort=8181
```

- Open http://localhost:8080/jenkins to check the Jenkins Dashboard.

- We can now get the Jenkins-git setup enabled by clicking the Manage Jenkins option and then click the Available tab and enter the search keyword “git”.

12.2 Setting up Jenkins

- Install the git plugin and restart Jenkins by issuing this command in the browser:

  http://localhost:8080/jenkins/restart

For additional information on installing Jenkins visit:
https://www.jenkins.io/doc/book/installing/

Jenkins Pipeline Process

12.3 Jenkins Pipeline Process

- Jenkins Pipeline, also known as "Pipeline," is a group of plugins that allows for the implementation and integration of continuous delivery pipelines into Jenkins.

- The Pipeline suite offers a flexible range of tools for creating delivery pipelines using the Pipeline domain-specific language (DSL) syntax.

- To create a Jenkins Pipeline, the pipeline's definition is written in a text file called a Jenkinsfile, which can then be saved to the project's source control repository.

- This approach is called "Pipeline-as-code," where the CD pipeline is treated as an integral part of the application to be versioned and reviewed, like any other code.

12.4 Jenkins Pipeline Process

- Creating a Jenkinsfile and committing it to source control provides a number of benefits:
    - Automatically creates a Pipeline build process for all branches and pull requests.
    - Code review/iteration on the Pipeline (along with the remaining source code).
    - Audit trail for the Pipeline.
    - Single source of truth for the Pipeline, which can be viewed and edited by multiple members of the project.
- Although the syntax for creating a Pipeline in Jenkins, whether through the web UI or a Jenkinsfile, is the same, it is commonly recommended to define the Pipeline using a Jenkinsfile and save it to a source control repository. This practice is regarded as a best practice.

12.4 Jenkins Pipeline Process

Declarative vs. Scripted Pipeline Syntax

- A Jenkinsfile can be written using two types of syntax - Declarative and Scripted.
- Declarative and Scripted Pipelines are constructed fundamentally differently. Declarative Pipeline is a more recent feature of Jenkins Pipeline which is designed to make writing and reading Pipeline code easier.
- Declarative Pipeline Provides richer syntactical features over Scripted Pipeline syntax.

12.4 Jenkins Pipeline Process

- Although the specific syntactical components, (aka"steps”) used in a Jenkinsfile can vary depending on the type of Pipeline, there are many commonalities between Declarative and Scripted Pipeline syntax.

12.4 Jenkins Pipeline Process

Jargon used in Jenkins Pipeline Process

- Pipeline
  - The Pipeline's code encompasses the complete build process, including stages for creating, testing, and deploying an application.
  - In addition, a pipeline block is an essential element of Declarative Pipeline syntax.

- Node
  - A node refers to a machine that is part of the Jenkins environment and has the ability to run a Pipeline.
  - Additionally, a node block is a crucial component of Scripted Pipeline syntax.

12.4 Jenkins Pipeline Process

- Stage
  - A stage block represents a logically distinct set of tasks executed throughout the entire Pipeline, such as "Build," "Test," and "Deploy" stages.
  - Many plugins utilize this information to display or present the status or progress of the Jenkins Pipeline.

- Step
  - A step refers to an individual action or command to be executed by Jenkins at a specific point in the Pipeline.
  - For instance, the ‘s’ step is used to execute a shell command, such as 'make.'
  - When a plugin extends the Pipeline DSL, it generally indicates that the plugin has introduced a new step.

12.4 Jenkins Pipeline Process

Prerequisites for setting up Jenkins Pipeline

- To use Jenkins Pipeline, you will need:
  - Jenkins 2.x or later (older versions back to 1.642.3 may work but are not recommended)
  - Pipeline plugin, which is installed as part of the "suggested plugins" (specified when running through the Post-installation setup wizard after installing Jenkins).

12.4 Jenkins Pipeline Process

Creating a Basic Pipeline through the Jenkins Classic UI

- If required, ensure you are logged in to Jenkins.
- From the Jenkins home page (i.e. the Dashboard of the Jenkins classic UI), click **New Item** at the top left.

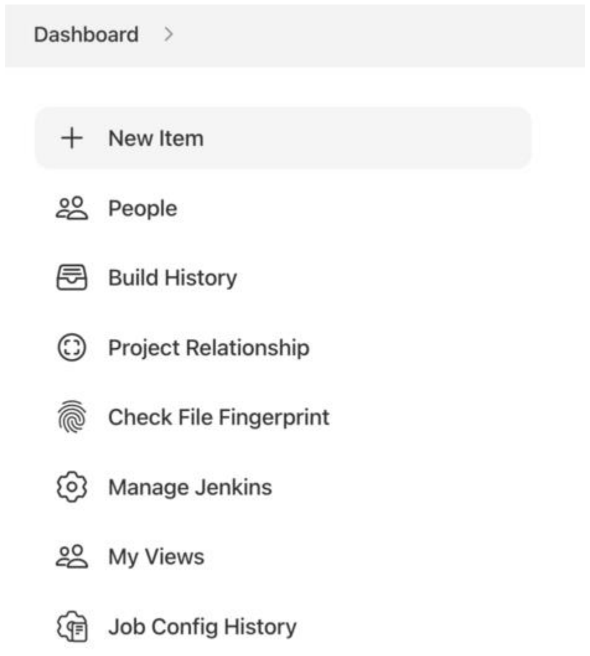


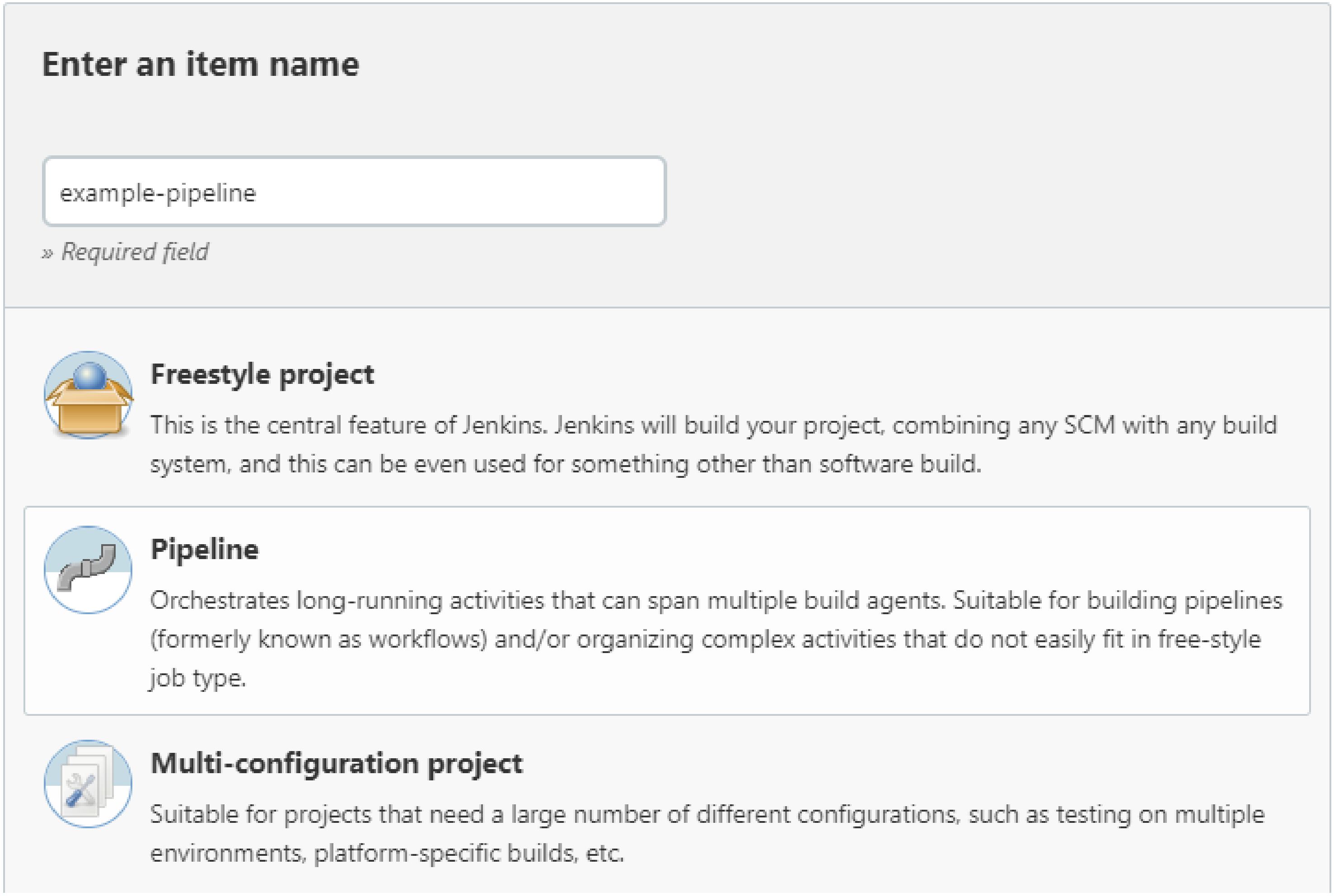
Image Reference: https://www.jenkins.io/doc/book/pipeline/getting-started/

12.4 Jenkins Pipeline Process

- In the **Enter an item name** field, specify the name for your new Pipeline project.

- Scroll down and click **Pipeline**, then click **OK** at the end of the page to open the Pipeline configuration page (whose **General** tab is selected).

12.4 Jenkins Pipeline Process


Enter an item name

example-pipeline

» *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.

Image Reference:
https://www.jenkins.io/doc/book/pipeline/getti
ng-started/

12.4 Jenkins Pipeline Process

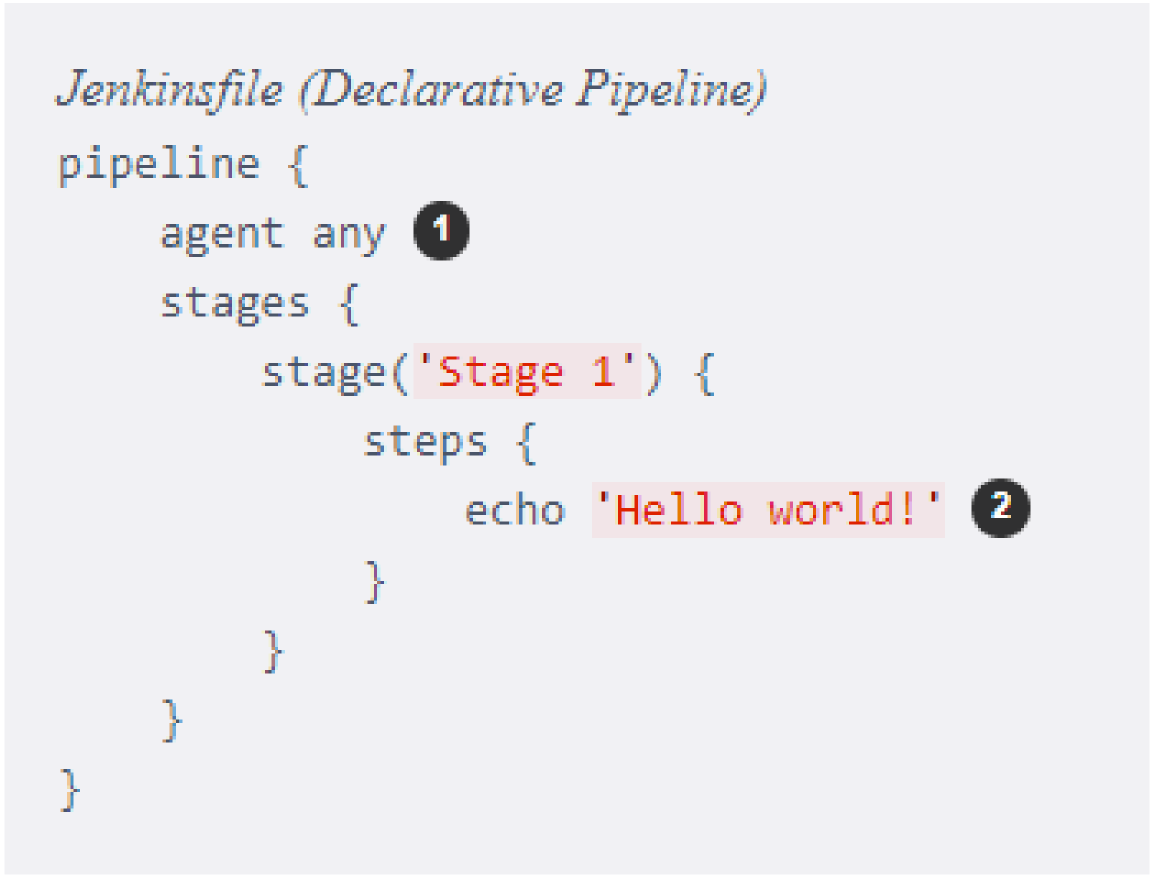
- Click the Pipeline tab at the top of the page to scroll down to the Pipeline section.

- In the Pipeline section, ensure that the Definition field indicates the Pipeline script option.

- Enter your Pipeline code into the Script text area.

12.4 Jenkins Pipeline Process

- Example code:



Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any 1
    stages {
        stage('Stage 1') {
            steps {
                echo 'Hello world!' 2
            }
        }
    }
}
```

Image Reference: https://www.jenkins.io/doc/book/pipeline/getting-started/

12.4 Jenkins Pipeline Process

1. ‘**agent**’ instructs Jenkins to allocate an executor (on any available agent/node in the Jenkins environment) and workspace for the entire Pipeline.

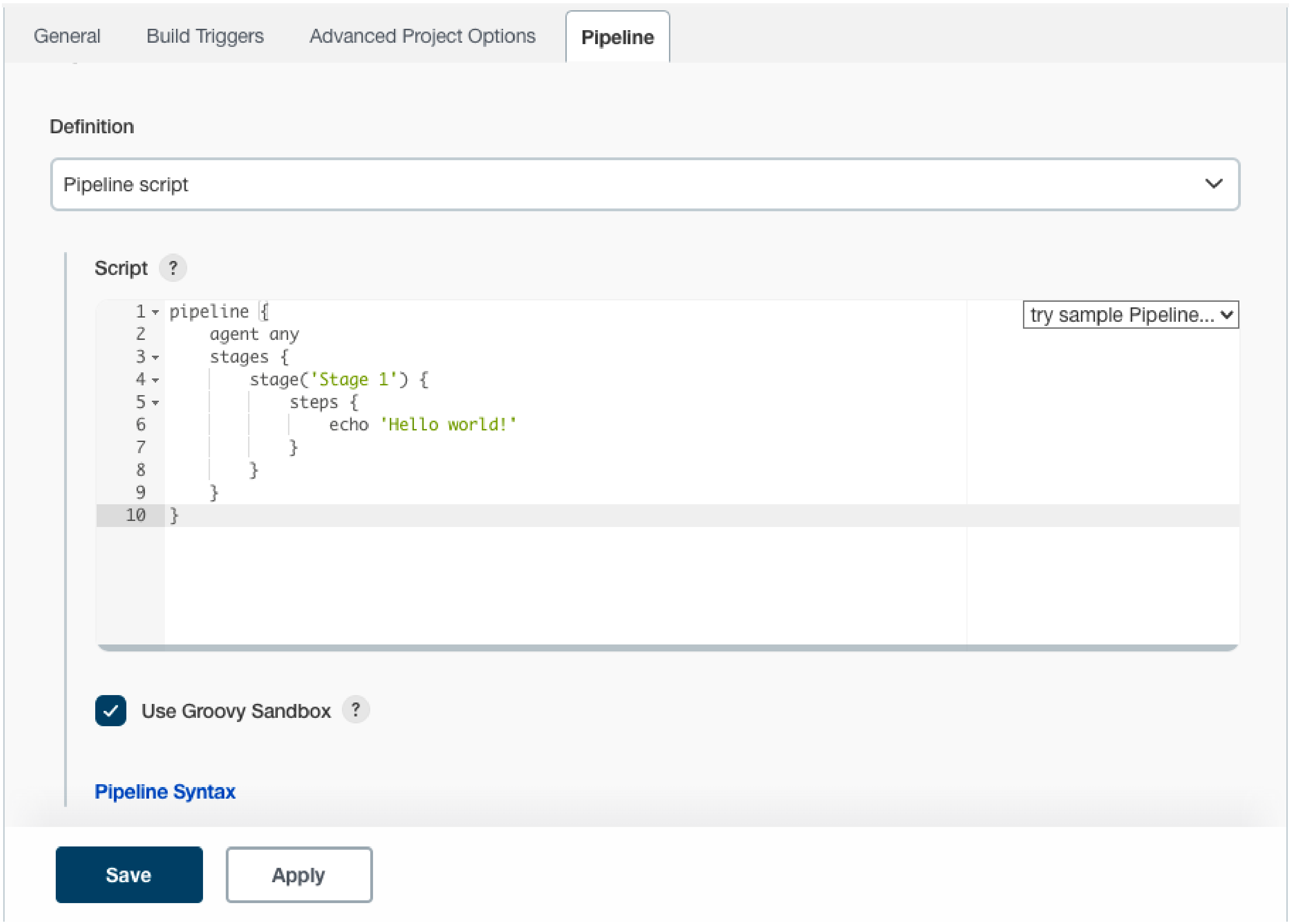
2. ‘**echo**’ writes simple string in the console output.

- You can also select from *Scripted* Pipeline examples from the **try sample Pipeline** option at the top right of the **Script** text area.

- Click **Save** to open the Pipeline project/item view page.

- On this page, click **Build Now** on the left to run the Pipeline.

12.4 Jenkins Pipeline Process



Definition

Pipeline script

Script

```
pipeline {
    agent any
    stages {
        stage('Stage 1') {
            steps {
                echo 'Hello world!'
            }
        }
    }
}
```

Image Reference:
https://www.jenkins.io/doc/book/pipeline/getting-started/

12.4 Jenkins Pipeline Process

- Under **Build History** on the left, click **1st option** to access the details for this particular Pipeline run.

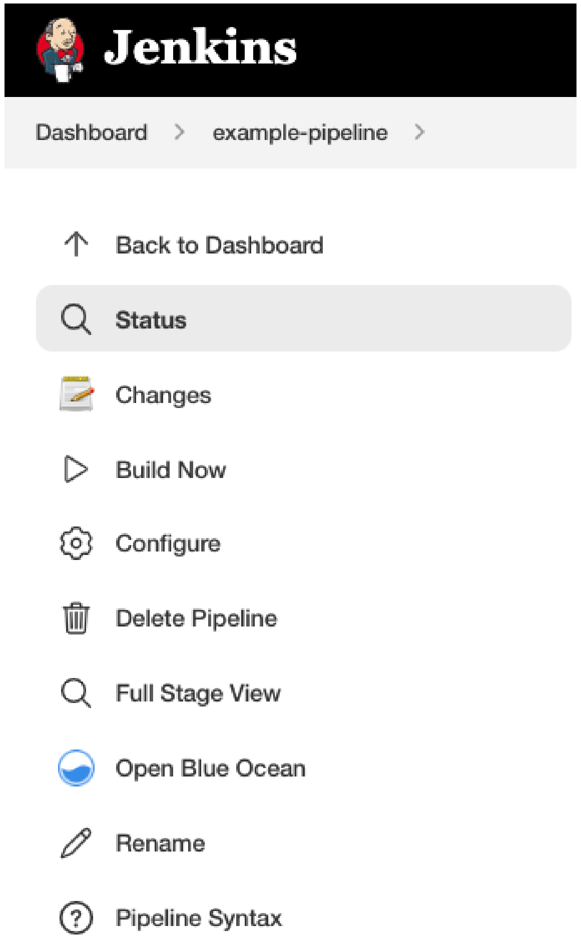
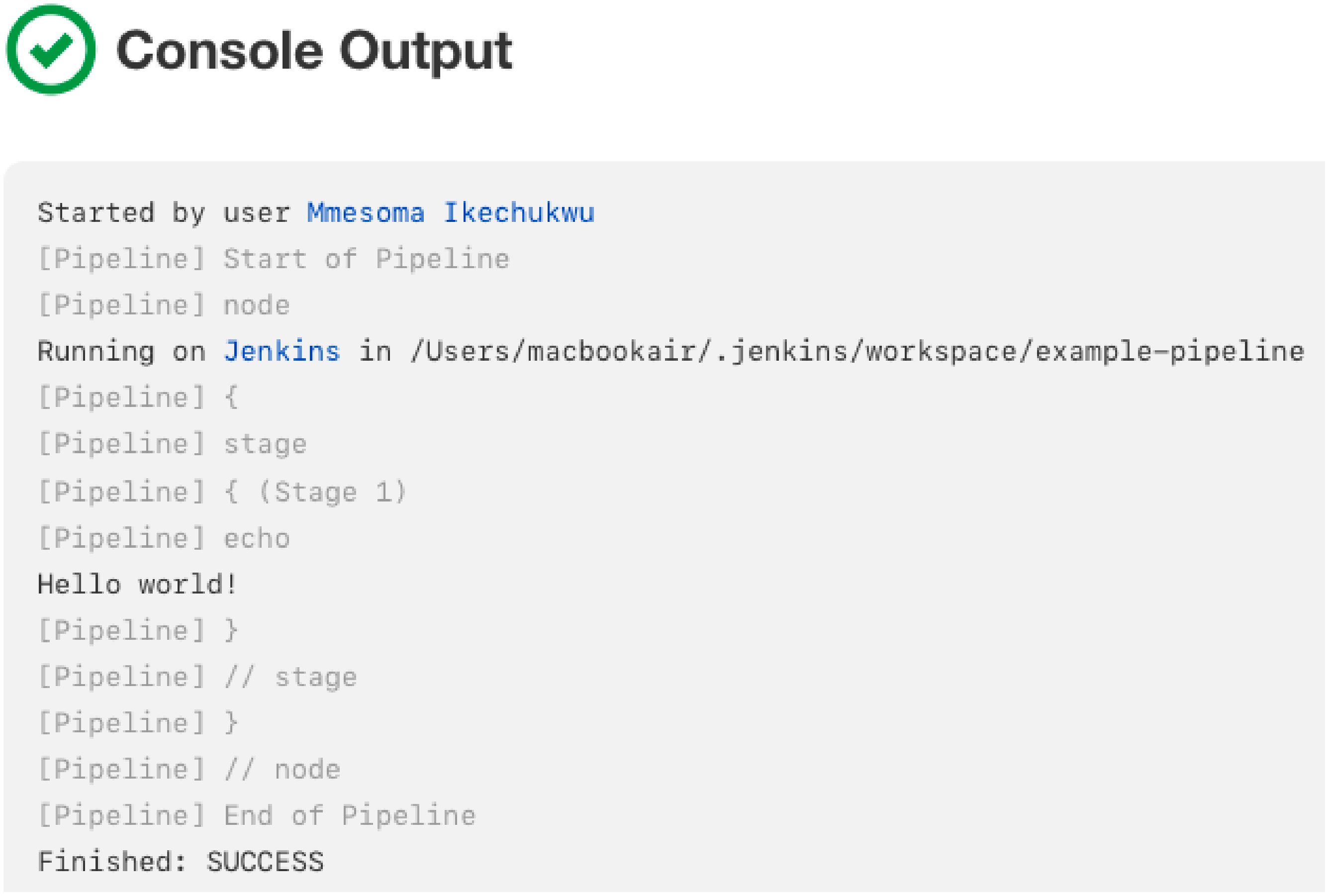


Image Reference: https://www.jenkins.io/doc/book/pipeline/getting-started/

12.4 Jenkins Pipeline Process

- Click **Console Output** to see the full output from the Pipeline run. The following output shows a successful run of your Pipeline.


Console Output

```
Started by user Mmesoma Ikechukwu
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /Users/macbookair/.jenkins/workspace/example-pipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Stage 1)
[Pipeline] echo
Hello world!
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Image Reference: https://www.jenkins.io/doc/book/pipeline/getting-started/

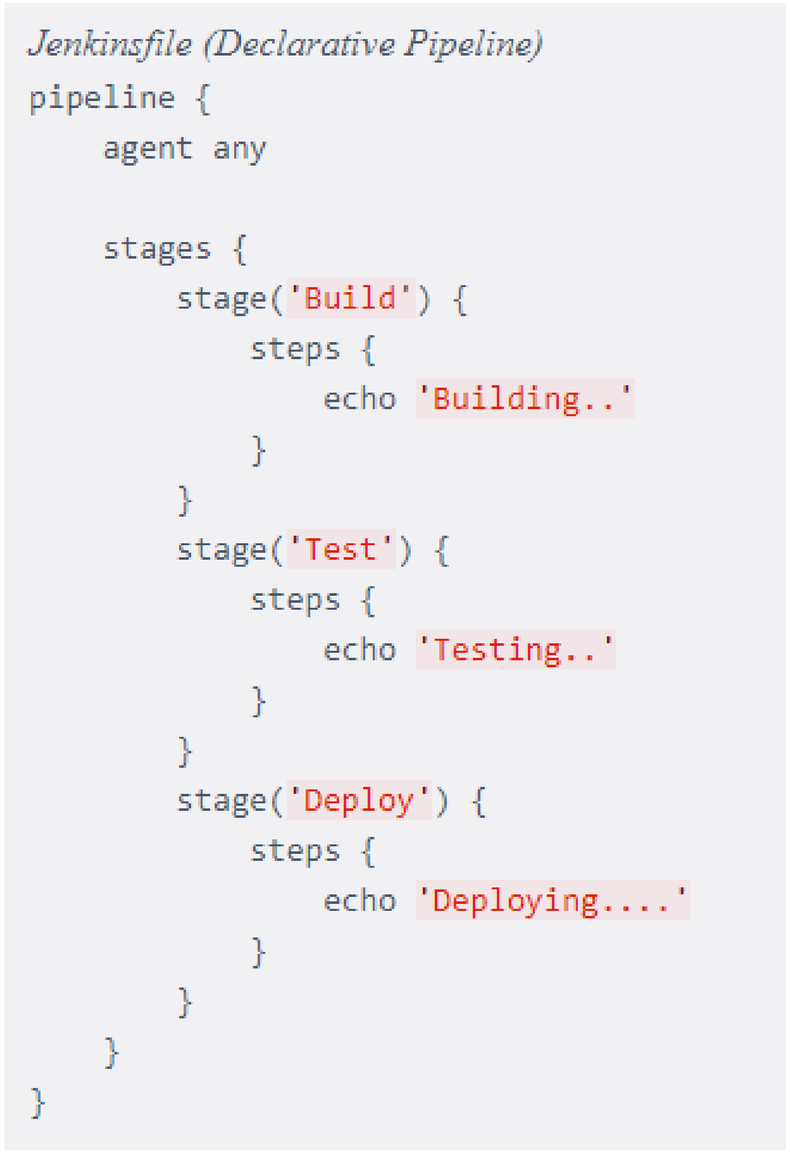
For more information on setting up a pipeline visit: https://www.jenkins.io/doc/book/pipeline/

12.4 Jenkins Pipeline Process

Creating a Jenkinsfile

- A Jenkinsfile is a type of file that includes the definition of a Jenkins Pipeline and is saved in a source control repository. An example of a basic three-stage continuous delivery pipeline implemented in a Pipeline is shown in the next slide.

12.4 Jenkins Pipeline Process



Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building..'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing..'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying....'
            }
        }
    }
}
```

Image Reference: https://www.jenkins.io/doc/book/pipeline/jenkinsfile/

12.4 Jenkins Pipeline Process

- Although not all Pipelines will require the same three stages, they are generally a good starting point for defining most projects.

- It is expected that a source control repository has already been established for the project, and a Pipeline has been defined in Jenkins according to the given instructions.

- To create a new Jenkinsfile, open a text editor that supports Groovy syntax highlighting and save the file in the root directory of the project.

12.4 Jenkins Pipeline Process

- The Declarative Pipeline example shown above includes the essential elements necessary to create a continuous delivery pipeline.

- The agent directive is necessary and instructs Jenkins to assign an executor and workspace to the Pipeline. Without an agent directive, the Declarative Pipeline would be invalid and unable to perform any tasks. By default, the agent directive ensures that the source repository is checked out and accessible for steps in the following stages.

- The stages and steps directives are also required for a valid Declarative Pipeline as they indicate to Jenkins what tasks to perform and in which stage they should be executed.

12.4 Jenkins Pipeline Process

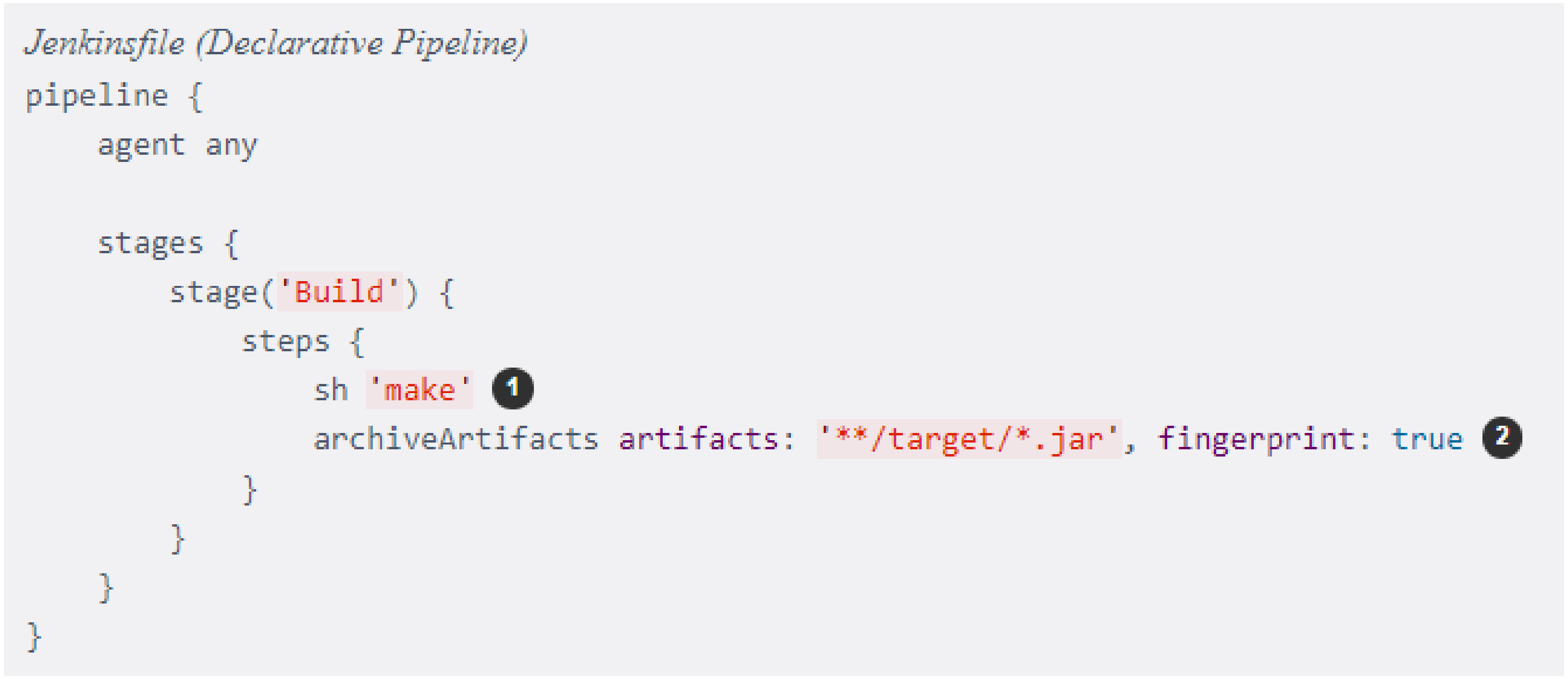
Build

- In many projects, the "build" stage marks the start of work in the Pipeline. This stage is typically where source code is compiled, assembled, or packaged.

- The Jenkinsfile is not meant to replace existing build tools like GNU/Make, Maven, Gradle, etc., but can be seen as a binding layer that connects the different phases of a project's development lifecycle (build, test, deploy, etc.).

- Jenkins offers various plugins that can invoke almost any build tool in general use, but in following example, we will use a shell step (‘sh’) to invoke make. Please note that the ‘sh’ step assumes that the system is Unix/Linux-based; for Windows-based systems, the ‘bat’ step can be used instead.

12.4 Jenkins Pipeline Process



Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                sh 'make' ❶
                archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true ❷
            }
        }
    }
}
```

Image Reference: https://www.jenkins.io/doc/book/pipeline/jenkinsfile/

1. The ‘sh’ step invokes the make command and will only continue if a zero exit code is returned by the command. Any non-zero exit code will fail the Pipeline.
2. ‘archiveArtifacts’ captures the files built matching the include pattern (**/target/*.jar) and saves them to the Jenkins controller for later retrieval.

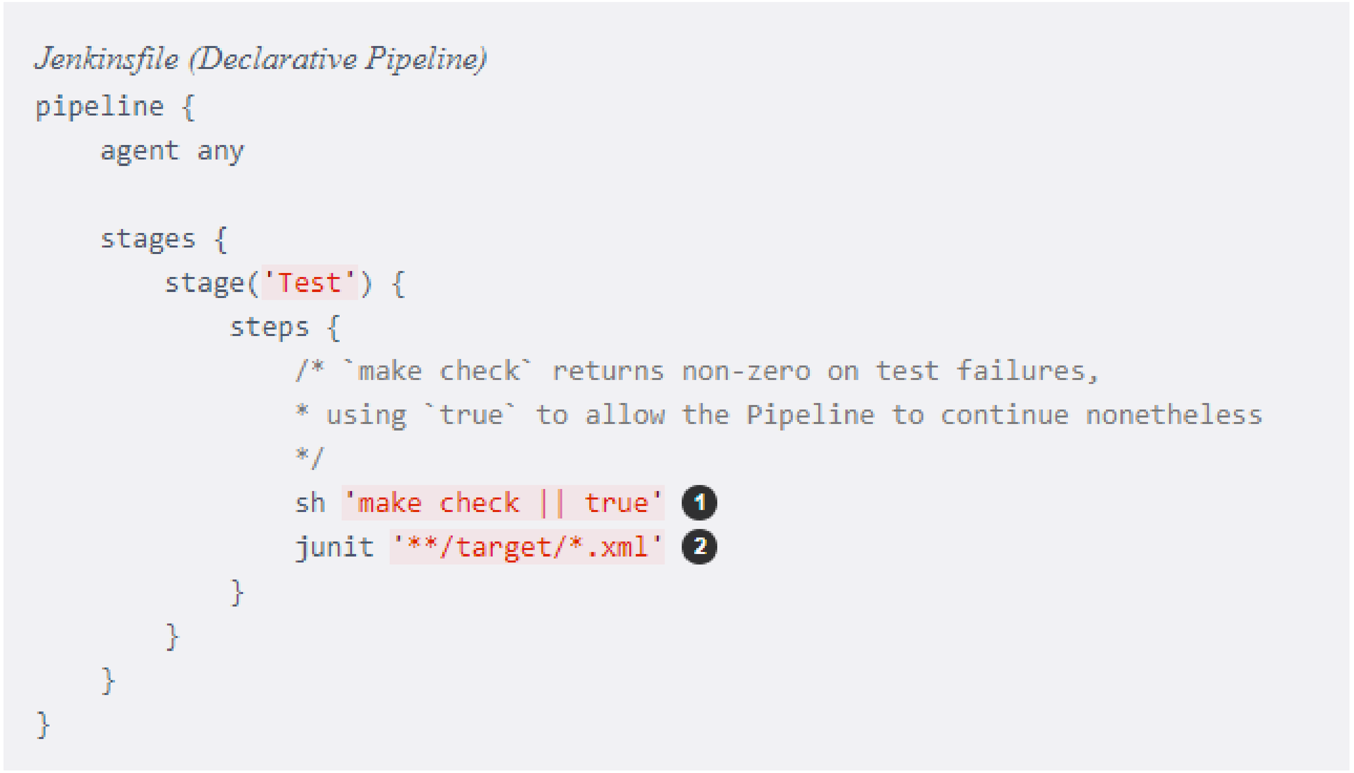
12.4 Jenkins Pipeline Process

Test

- Automated testing plays a critical role in the continuous delivery process. Jenkins offers various plugins to record, report, and visualize test results.

- Recording failures is important for reporting and viewing purposes. The junit step, provided by the JUnit plugin, is used in the example below. If there are test failures, the Pipeline will be marked "unstable," which will be indicated by a yellow ball in the web UI.

12.4 Jenkins Pipeline Process



Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any

    stages {
        stage('Test') {
            steps {
                /* `make check` returns non-zero on test failures,
                * using `true` to allow the Pipeline to continue nonetheless
                */
                sh 'make check || true' ❶
                junit '**/target/*.xml' ❷
            }
        }
    }
}
```

Image Reference: https://www.jenkins.io/doc/book/pipeline/jenkinsfile/

1. Using an inline shell conditional ensures that the ‘sh’ step always sees a zero exit code, giving the ‘junit’ step the opportunity to capture and process the test reports. Alternative approaches to this are covered in more detail in the Handling failure section below.

12.4 Jenkins Pipeline Process

2. junit captures and associates the JUnit XML files matching the inclusion pattern (**/target/*.xml).

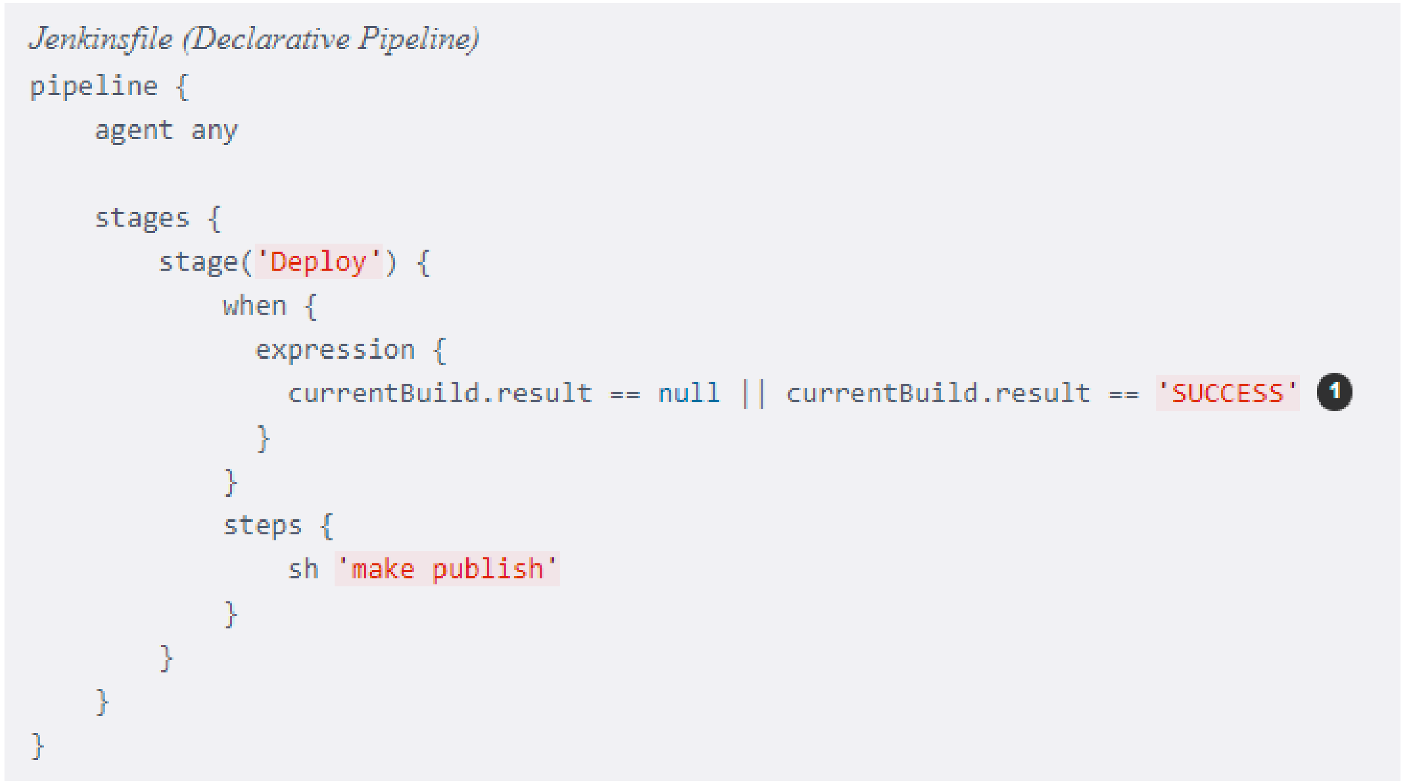
12.4 Jenkins Pipeline Process

Deploy

- The process of deployment can involve different actions depending on the specific requirements of the project or organization, which can range from publishing built artifacts to a Artifactory server, to uploading code to a production system.

- In this particular example Pipeline, the previous stages, "Build" and "Test", have been executed successfully. The "Deploy" stage will only proceed if the previous stages completed without errors; otherwise, the Pipeline would have stopped before reaching this stage.

12.4 Jenkins Pipeline Process



Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any

    stages {
        stage('Deploy') {
            when {
              expression {
                currentBuild.result == null || currentBuild.result == 'SUCCESS' (1)
              }
            }
            steps {
                sh 'make publish'
            }
        }
    }
}
```

Image Reference: https://www.jenkins.io/doc/book/pipeline/jenkinsfile/

1. Accessing the 'currentBuild.result' variable allows the Pipeline to determine if there were any test failures. In which case, the value would be UNSTABLE.

Introduction to Version Control Systems

12.4 Introduction to Version Control Systems

- Version control or Source control refers to the process of monitoring and handling modifications made to software code.

- Software teams employ Version Control Systems (VCS), which are software applications that assist in overseeing alterations made to source code over a period of time.

12.4 Introduction to Version Control Systems

Features of VCS

- Back-up methodology
- Record increments - know which version is live
- Manitain change history - when features were added or amended
- Allow multiple developers (in remote locations) to work on same code base
- Merge changes across same files - handle collisions
- Answers who did what
- Point in time marking aka. Tagging
- Branching - release versions maintained & main development can continue

12.4 Introduction to Version Control Systems

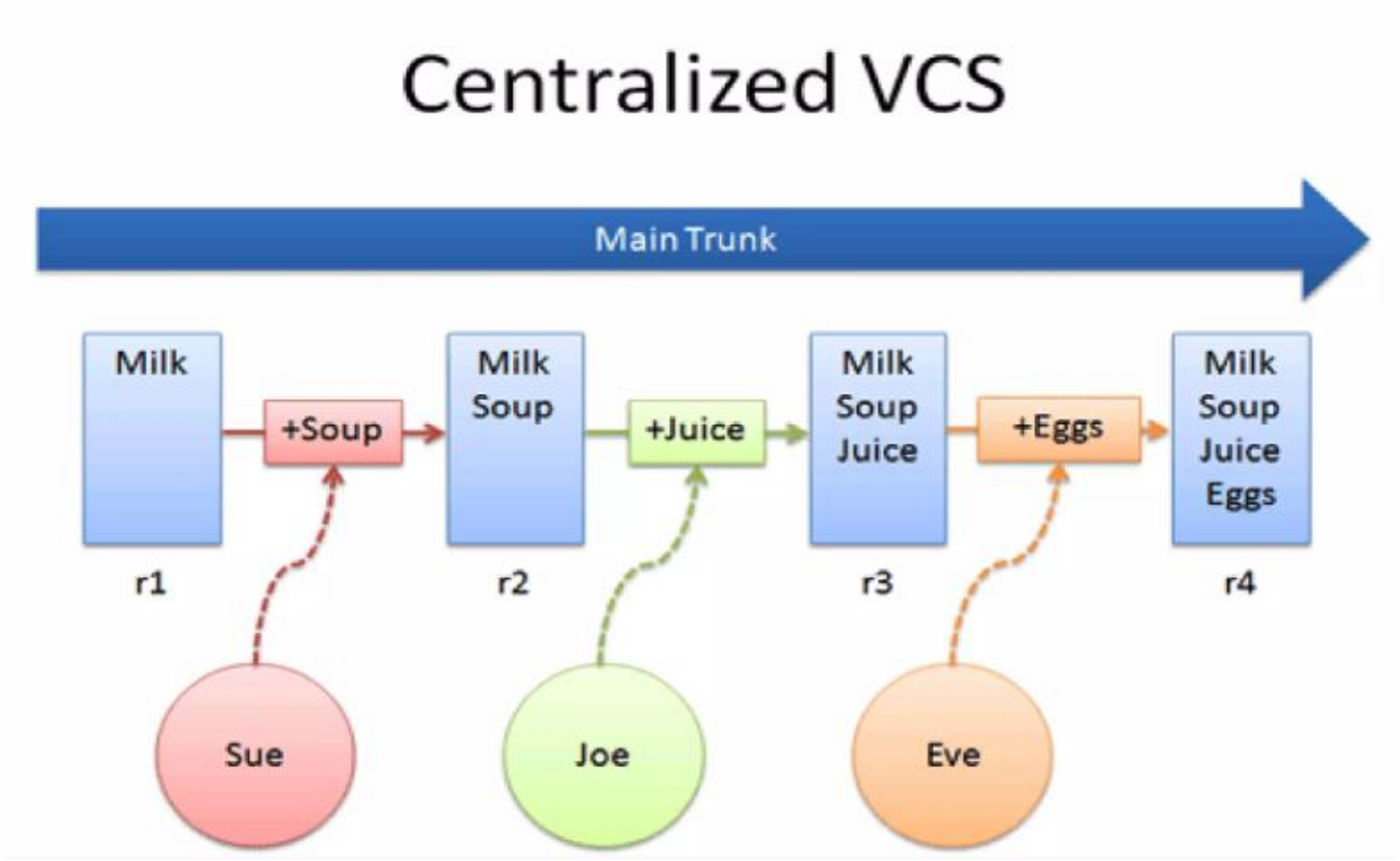


Image Reference: https://mastersofmedia.hum.uva.nl/blog/2009/11/01/git-virtue-github-and-commons-based-peer-production/

12.4 Introduction to Version Control Systems

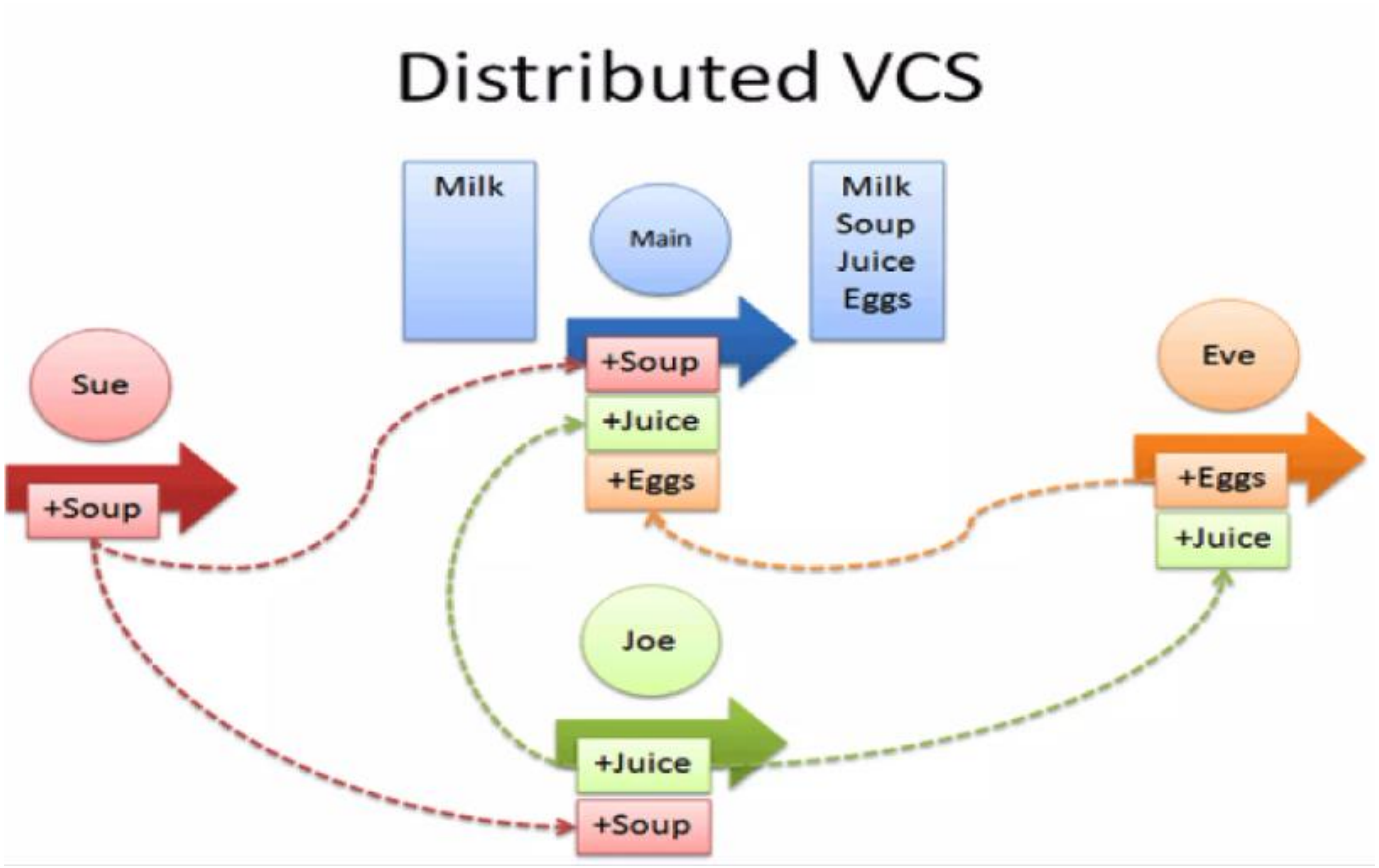


Image Reference: https://mastersofmedia.hum.uva.nl/blog/2009/11/01/git-virtue-github-and-commons-based-peer-production/

12.4 Introduction to Version Control Systems

Jargon used in VCS

- Repository (repo): Database where files are stored
- Server: The computer storing the repository
- Client: The computer connecting to the repository
- Working Copy: Your local directory of files
- Trunk / Main: Master location for code in the repository
- Head: The latest revision in the repository

12.4 Introduction to Version Control Systems

Actions performed in VCS

- Add: Place a file under version control
- Check in: Send local changes to the repo
- Check out: Download from a repo to your working copy •
- Ignore: Allows files to exist in your working copy but not in the repo
- Revert: Throw away your working copy and restore last revision
- Update / Sync: Update your working copy to the latest revision
- Diff / Change: Specific modification to a document
- Branch: Duplicate copy of code used for feature development
- Merge: Integrate changes from two different branches
- Conflict: Inability to reconcile changes to a document

12.4 Introduction to Version Control Systems

- Resolve: Manual fixing of conflicted document changes
- Locking: Prevents other developers from making changes

12.4 Introduction to Version Control Systems

Example VCSs

- Concurrent Version System (CVS) - http://www.nongnu.org/cvs/
- Subversion (SVN) - http://subversion.tigris.org/
- Git - http://git.or.cz/
- Bazaar - http://bazaar-vcs.org/
- Mercurial -http://www.selenic.com/mercurial/
- Monotone - http://www.monotone.ca/
- Visual Source Safe (VSS) - Microsoft visual tool - Distributed version control

12.4 Introduction to Version Control Systems

Git

- Git is a distributed revision control system developed in 2005 by Linus Torvalds as open source software.
- Git provides strong support for non-linear development.
- Git supports macOS, Linux, and Windows and compatible with existing systems and protocols like HTTP, FTP, ssh.
- Git can efficiently handle small to large sized projects.

12.4 Introduction to Version Control Systems

Git Project Sections

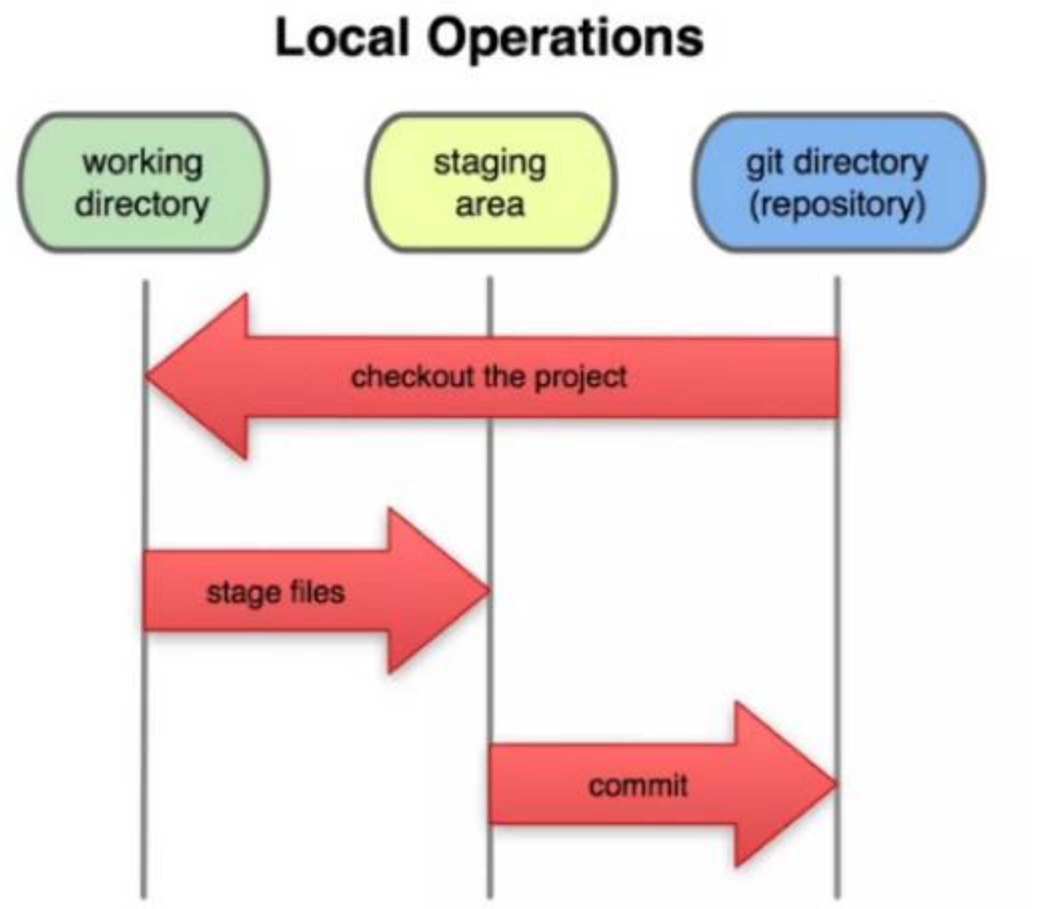


Image Reference: https://textbooks.cs.ksu.edu/cis400/b-git-and-github/05-staging-and-committing/

12.4 Introduction to Version Control Systems

Installing Git

- Installing on Linux
  - If you’re on Fedora (or any closely-related RPM-based distribution, such as RHEL or CentOS), you can use dnf:

```
$ sudo dnf install git-all
```

  - If you’re on a Debian-based distribution, such as Ubuntu, try apt:

```
$ sudo apt install git-all
```

  - For more options, there are instructions for installing on several different Unix distributions on the Git website, at https://git-scm.com/download/linux.

12.4 Introduction to Version Control Systems

- Installing on macOS
  - There are several ways to install Git on a Mac. The easiest is probably to install the Xcode Command Line Tools. On Mavericks (10.9) or above you can do this simply by trying to run git from the Terminal the very first time.

```
$ git –version
```

  - If you don’t have it installed already, it will prompt you to install it.
  - If you want a more up to date version, you can also install it via a binary installer. A macOS Git installer is maintained and available for download at the Git website, at https://git-scm.com/download/mac.

12.4 Introduction to Version Control Systems

- Installing on Windows
  - There are also a few ways to install Git on Windows. The most official build is available for download on the Git website. Just go to https://git-scm.com/download/win and the download will start automatically. Note that this is a project called Git for Windows, which is separate from Git itself; for more information on it, go to https://gitforwindows.org.

  - To get an automated installation you can use the Git Chocolatey package. Note that the Chocolatey package is community maintained.

12.4 Introduction to Version Control Systems

Initializing a Git Repository

- If you have a project directory that is currently not under version control and you want to start controlling it with Git, you first need to go to that project's directory. If you've never done this, it looks a little different depending on which system you're running:

```
$ git init
```

- This creates a new subdirectory named .git that contains all of your necessary repository files — a Git repository skeleton. At this point, nothing in your project is tracked yet. See Git Internals for more information about exactly what files are contained in the .git directory you just created.

12.4 Introduction to Version Control Systems

- If you want to start version-controlling existing files you should probably begin tracking those files and do an initial commit. You can accomplish that with a few ‘git add’ commands that specify the files you want to track, followed by a git commit:

```
$ git add *.c
$ git add LICENSE
$ git commit -m 'Initial project version‘
```

- We’ll go over what these commands do in just a minute. At this point, you have a Git repository with tracked files and an initial commit.

12.4 Introduction to Version Control Systems

Viewing the Commit History

- Use the 'git log' command

```
$ git log

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

   Initial commit
```

12.4 Introduction to Version Control Systems

Creating a New Branch

- Use the 'git branch' command

```
$ git branch testing
$ git checkout testing      (To switch to that branch)
```

- Now you can commit to that branch
- For switching back to main branch

```
$ git checkout master
```

12.4 Introduction to Version Control Systems

Merging

- Use ‘git merge’ command,

```
$ git checkout master   (Switched to branch 'master‘)
$ git merge testing     (Merge made by the 'recursive' strategy.)

index.html |   1 +
1 file changed, 1 insertion(+)
```

For more information on Git visit: https://git-scm.com/docs

Summary

- Continuous Integration is an approach to be continually validating the state of codebase through automated testing. It is best achieved through integration with version control.
- Continuous Delivery/Deployment is and approach to regularly deploying artifacts that successfully pass the CI phase to ensure confident around the deployment.
- Jenkins is a widely used open-source tool for CI which offers the ability to build software projects manually, periodically, or automatically, and comes equipped with hundreds of plugins to support deployment and automation.
- Version Control Systems (VCS) are software applications that assist in overseeing alterations made to source code over a period of time. They record increments, maintain change history and also allow multiple developers to work on same code base.

Summary

- Git is a distributed revision control system developed in 2005 by Linus Torvalds as open source software. It supports macOS, Linux, and Windows and compatible with existing systems and protocols like HTTP, FTP, ssh.