Build Automation & CI/CD with Jenkins

Key Takeaways

TECHWORLD WITH NANA

Introduction to Build Automation

TECHWORLD WITH NANA

What is Build Automation? - 1

Build automation is the **process of automating** the:

- *retrieval of source code*
- *execution of automated tests*
- *compiling into binary code/build docker image*
- *pushing of artifacts to a repository*
- *deploying of artifacts*

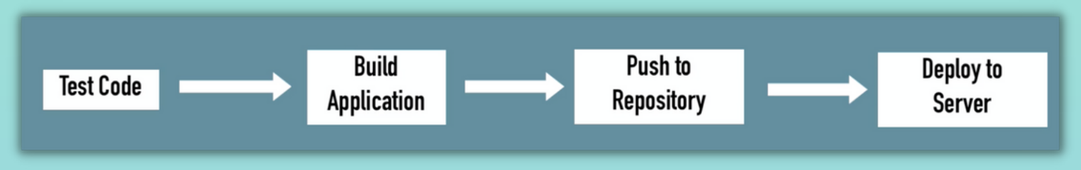
Build Automation and CI/CD is at the heart of DevOps

What is Build Automation? - 2

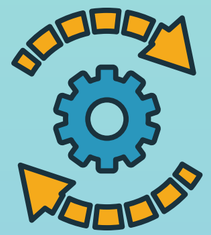
Doing this **manually** each time would be **super inconvenient**!

- Stash working changes and moving to main branch
- Logging in to correct docker repository
- Setting up the test environment to run tests
- Executing tests (meanwhile you CAN'T code)
- Building Docker Image and pushing to a repository

So we have a **dedicated server** executing this



- Test environment prepared
- Docker credentials configured
- All necessary tools installed
- Trigger this build automatically



What is CI / CD ?

- CI/CD stands for **Continuous Integration / Continuous Deployment**
- The goal of CI/CD is to "release early and often". And the way to do this, is by **automating the whole software release cycle**

Continuous Integration

- Means new code changes are continuously built, tested and merged to shared repository

Continuous Deployment

- Automating further stages of the pipeline
- Automatically deploying to different deployment environments including releasing to production

- Build automation is a basic component of CI/CD pipelines

Jenkins = most used in the industry

- Jenkins is currently one of the most used build automation CI/CD tools in the industry

Many others available:

GitHub Actions

Travis CI

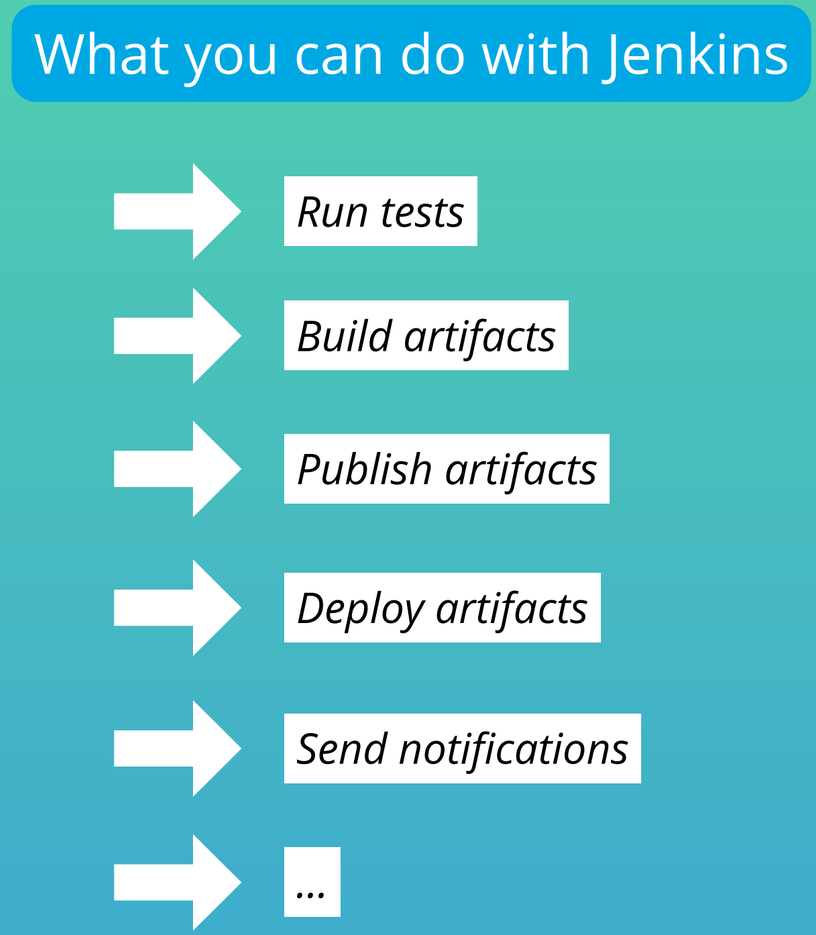
GitLab

Bamboo

TeamCity

Introduction to Jenkins - 1

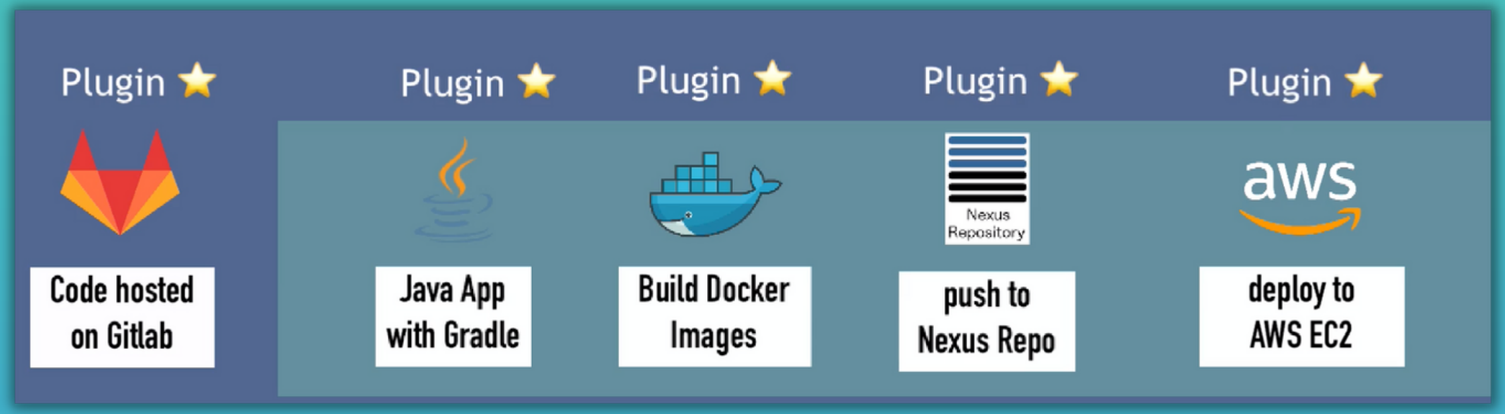
1. Jenkins is software that you **install on a dedicated server**
2. It has a **UI** to configure your builds
3. You need to **install all the tools** you need to get the tasks done (Docker, Gradle/Maven/npm, etc.)
4. Then you need to **configure the tasks** (run tests, build app, deployment etc.)
5. And **configure the automatic trigger** of the workflow



Introduction to Jenkins - 2

One of most important capabilities a build automation tool needs to have is **integration with other technologies**

- Jenkins makes these integrations possible with **Plugins.** Plugin = **extension to Jenkins functionality**
- There are plugins for Docker, Build Tools, Repositories, Deployment Servers etc.



- Plugins help you to execute each of those steps

TECHWORLD WITH NANA

Introduction to Jenkins - 3

- **A common DevOps Engineer task is to administer Jenkins** (plugins, setup etc.), create or help developers to set up build automation jobs in Jenkins

How it works:

RUN TESTS → Build Tools need to be available

BUILD ARTIFACTS → Build Tools or Docker need to be available

PUBLISH ARTIFACTS → Store credentials in Jenkins

 → Jenkins User must have access to all these technologies and platforms

What to configure:

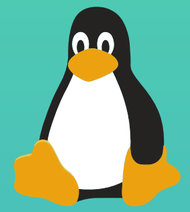
- Install Jenkins and prepare everything
- The setup needs to be done only once
- Plugins, Credentials etc. can be used for different projects

Jenkins Installation

2 ways to install Jenkins on a server:

1) Run Jenkins as Docker container

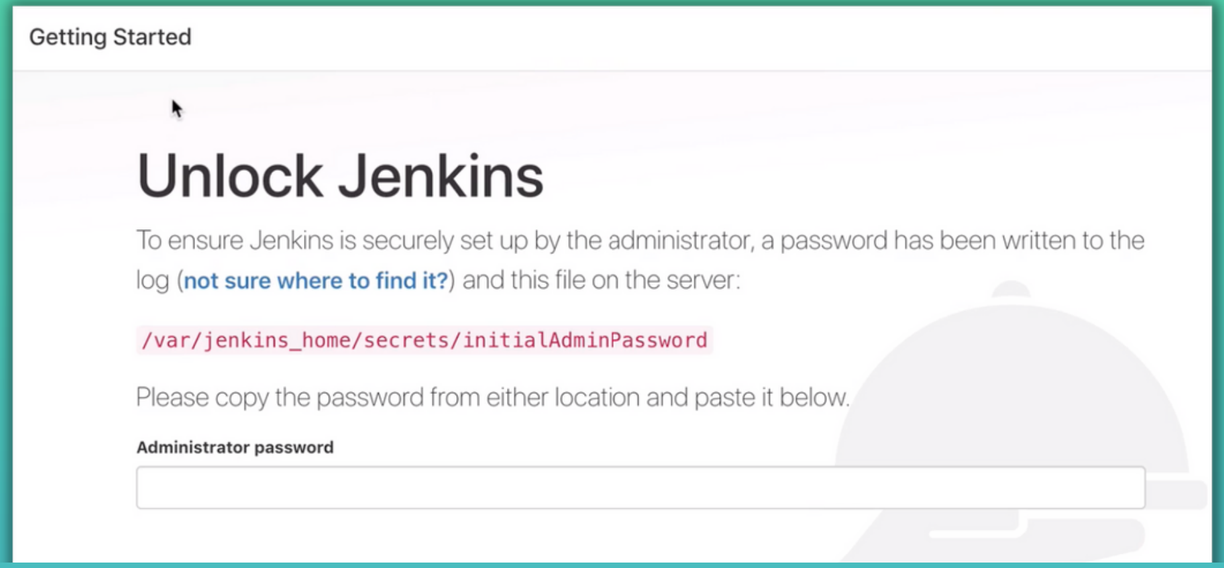
2) Install Jenkins directly on OS



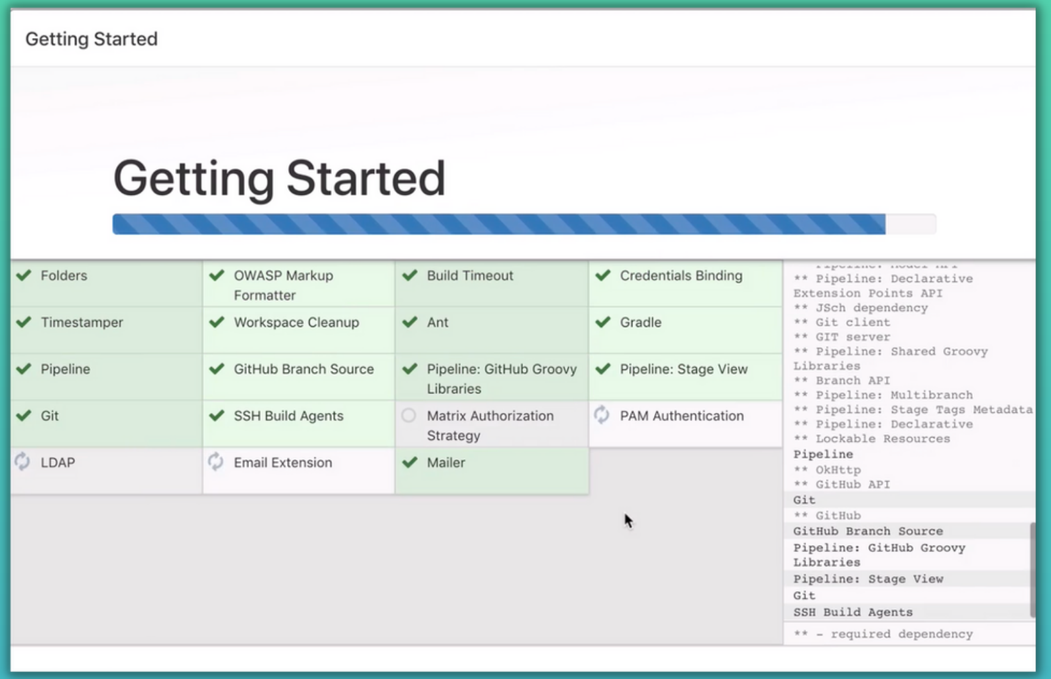
```
nana — root@ubuntu-s-2vcpu-4gb-fra1-01: ~ — ssh root@165.232.114.245 — 119×32
root@ubuntu-s-2vcpu-4gb-fra1-01:~# docker run -p 8080:8080 -p 50000:50000 -d \
> -v jenkins_home:/var/jenkins_home jenkins/jenkins:lts
```

TECHWORLD WITH NANA

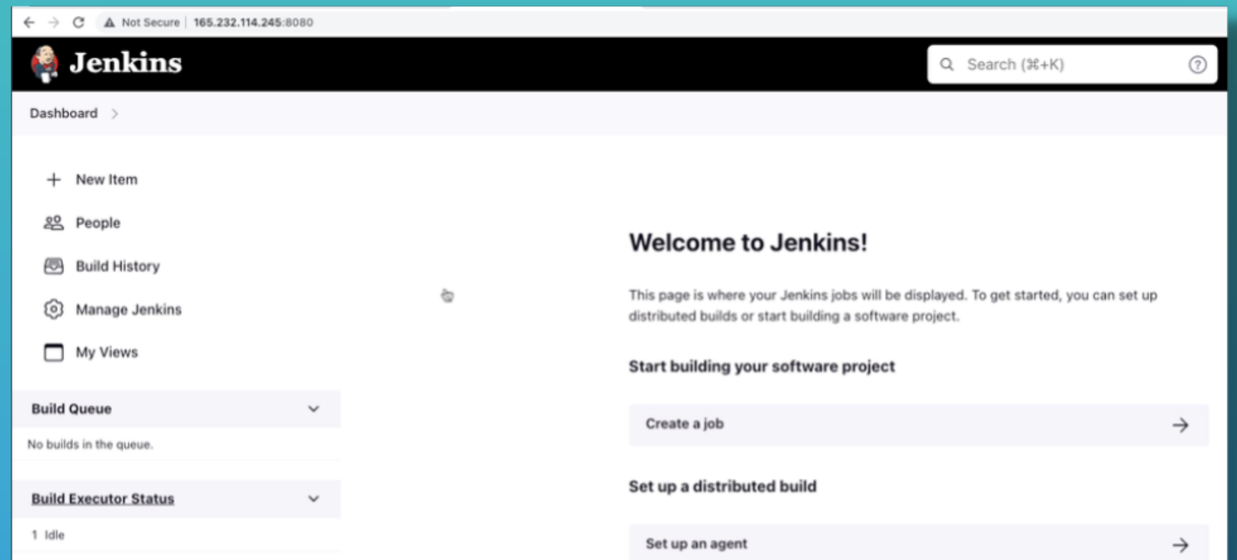
Getting Started



Enter password



Some plugins are installed automatically



Successfully logged in

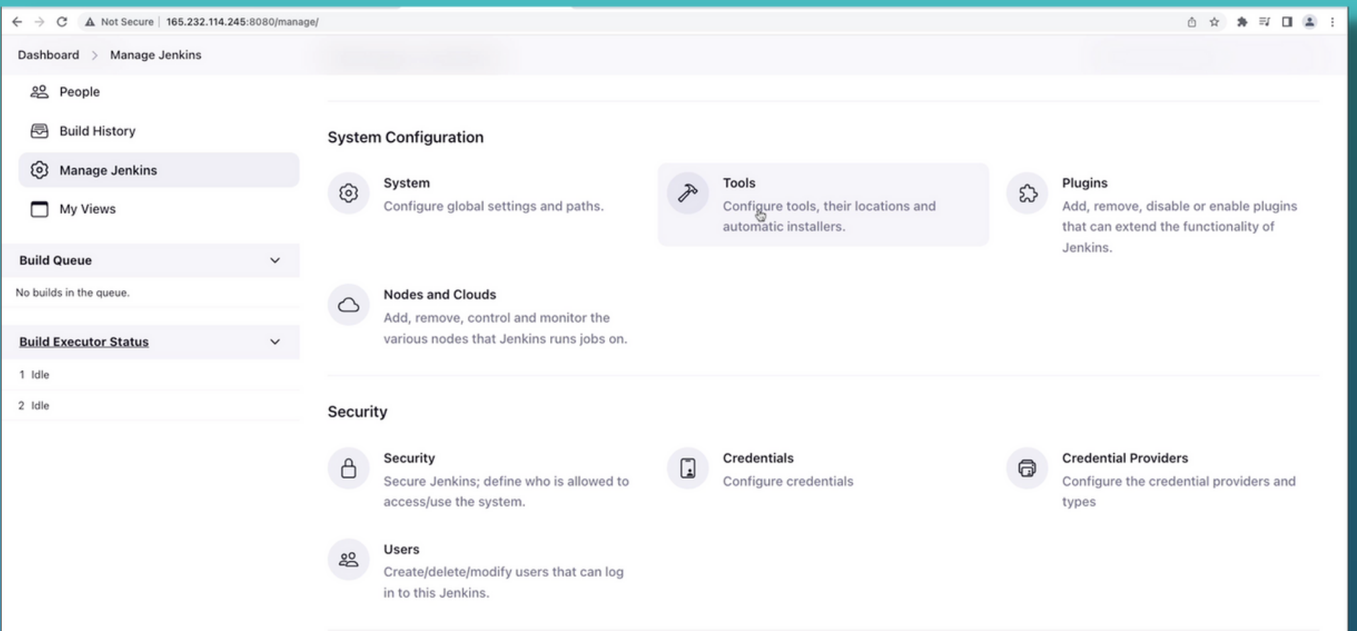
TECHWORLD WITH NANA

Roles

Jenkins Administrator

Operations or DevOps teams:

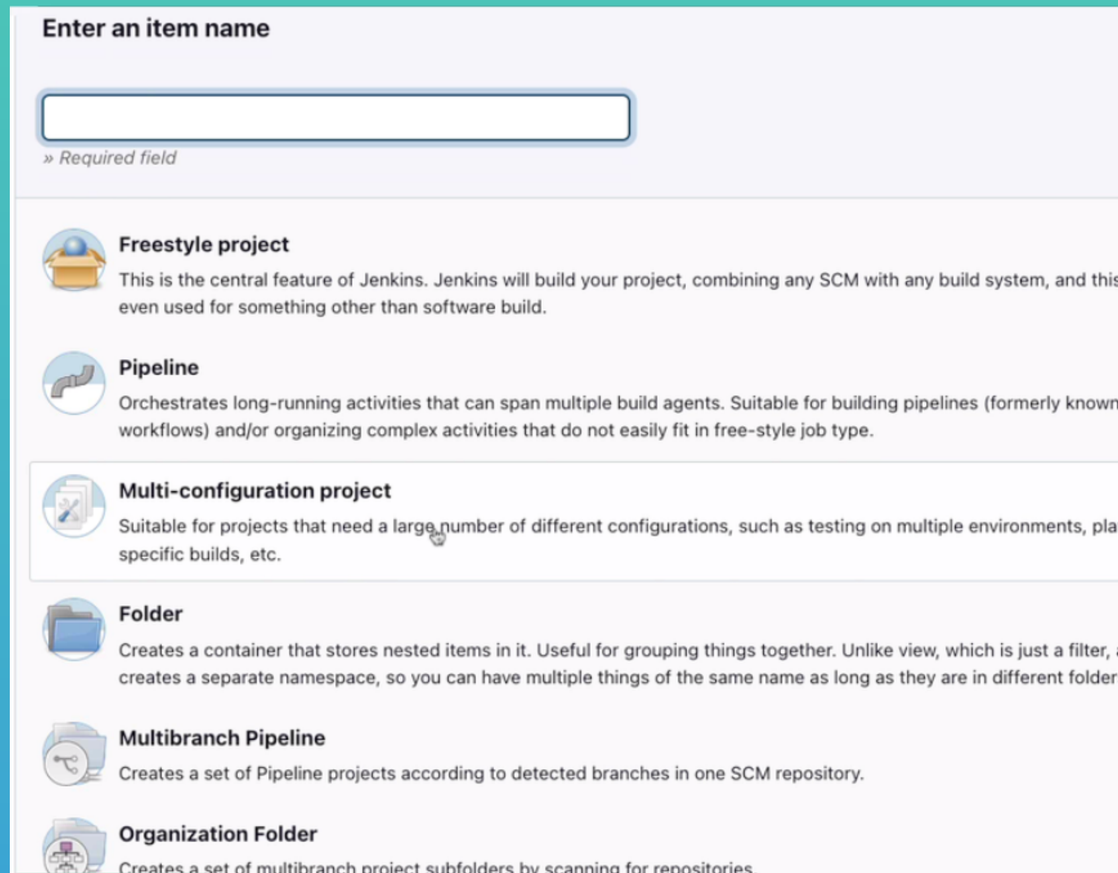
- **Administers** and manages Jenkins
- Sets up Jenkins cluster
- Installs plugins
- Backs up Jenkins data
- ...



Jenkins User

Developers or DevOps teams:

- Creating the **actual jobs** to run workflows



Deep Dive into Jenkins

Install Build Tools in Jenkins - 1

- Jenkins is used to **build applications**
- So first we need to **install and configure the build tools** in Jenkins

Java App

- Java App with Maven Build Tool
- Run Tests
- Build Jar File

→ ***Maven** needs to be available on Jenkins*

JavaScript App

- Node App
- Run Tests
- Package and Push to Repo

→ ***npm** needs to be available on Jenkins*

Depending on your app (**programming language**) you need to have **different tools** installed and configured on Jenkins

Install Build Tools in Jenkins - 2

- **2 ways** to install and configure those tools:

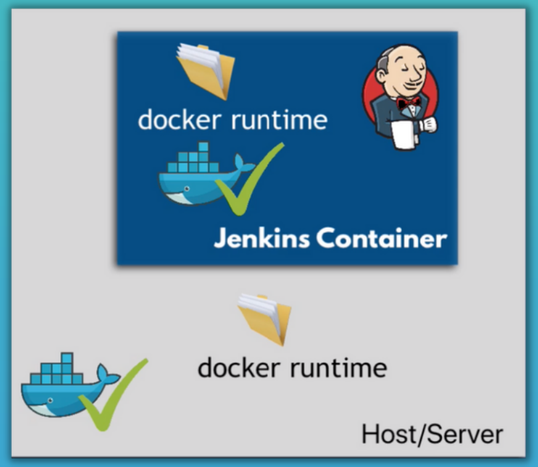
1) Jenkins Plugins

- Just install a plugin (via UI) for your tool

2) Install Tools directly on server

- Connect to remote server and install
- Inside the Docker container, when Jenkins runs as a container

How to make Docker available in Jenkins container



- To execute Docker commands inside Jenkins, we need Docker available. One way is to **mount Docker from host into Jenkins**:

```
root@ubuntu-s-2vcpu-4gb-fra1-01:~# docker run -p 8080:8080 -p 50000:50000 -d \
> -v jenkins_home:/var/jenkins_home \
> -v /var/run/docker.sock:/var/run/docker.sock jenkins/jenkins:lts
```

TECHWORLD WITH NANA

Jenkins Job Types - 1

- **Build jobs** are at the heart of Jenkins build process
- A job is a task or step in your build process
- "Job" is a deprecated term, but still widely used (called "Project" now)
- **2 of 3 most important jobs and their differences**:

Freestyle job

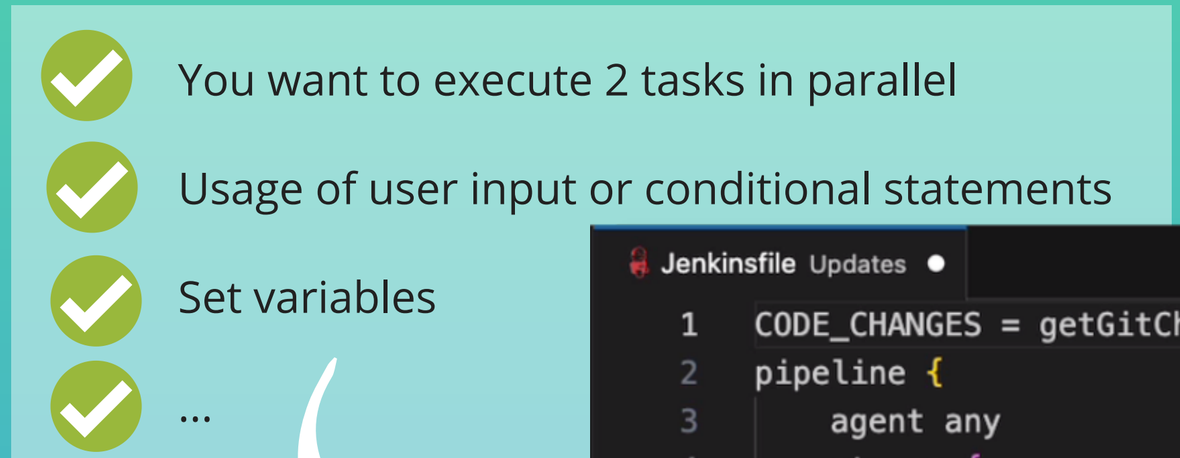
- To orchestrate **simple jobs** for a project
- Configuration: Via UI
- Very limited, limited to the input fields of plugin

Pipeline job

- For more complex workflows, therefore **suitable for CI/CD pipelines**
- Ability to divide the tasks into different stages
- Configuration: UI and Scripting (Pipeline as Code)
- Much more powerful

Jenkins Job Types - 2

- **More advantages or use cases** for a Pipeline job:



Jenkinsfile Updates

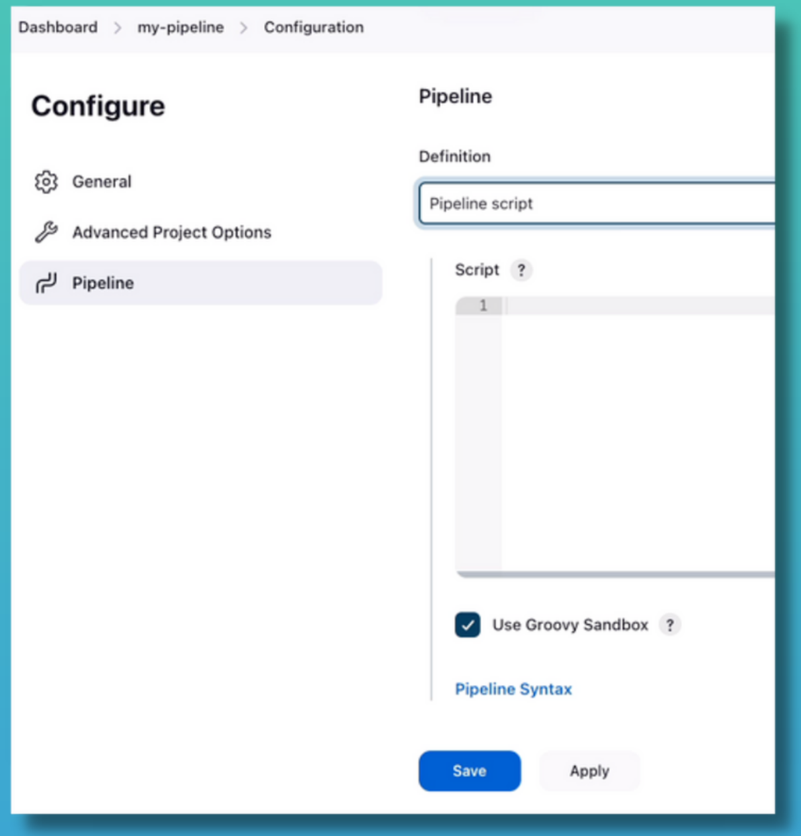
```
1   CODE_CHANGES = getGitChanges()
2   pipeline {
3       agent any
4       stages {
5           stage("build") {
6               when {
7                   expression {
8                       BRANCH_NAME == 'dev' && CODE_CHANGES == true
9                   }
10              }
11              steps {
12                  echo 'building the application...'
13              }
14          }
```

"Pipeline as Code"

- With a Pipeline job you have the ability to **script your pipeline** instead of using UI configuration
- "Pipeline as Code", which is current best practice in industry: "**Everything as Code**"

  (Infrastructure as Code, Configuration as Code, Policy as Code, Security as Code, ...)

1) Write code directly in UI:



2) Write code in a **Jenkinsfile:**

TECHWORLD WITH NANA

Jenkinsfile

- Jenkinsfile is a **text file that contains the definition of a Jenkins Pipeline**
  - Is checked into a project's git repository
- **2 ways to write a Jenkinsfile**:

Scripted

- First syntax
- Groovy engine
- Advanced scripting capabilities, high flexibility
- Difficult to start

Declarative Pipeline

- Easier to get started, but not that powerful
- **Pre-defined structure**

dev / Jenkinsfile

```
1  pipeline {
2
3      agent any
4
5      stages {
6
7          stage("build") {
8
9              steps {
10
11             }
12         }
13     }
14 }
15
```

TECHWORLD WITH NANA

Jenkinsfile Syntax - 1

dev / Jenkinsfile

```
1  pipeline {
2
3      agent any
4
5      stages {
6
7          stage("build") {
8
9              steps {
10
11             }
12         }
13     }
14 }
15
16 node {
17     // groovy script
18 }
```

Required fields of Jenkinsfile

- "**pipeline**" - must be top-level
- "**agent**" - where to execute
- "**stages**" - where the "work" happens
    - "stage" and "steps"

Jenkinsfile Syntax - 2

post

- Execute some logic **AFTER** all stages executed
- Available **conditions**: Execute "always", only on build "success" or build "failure"

```
19          stage("deploy") {
20
21              steps {
22                  echo 'deploying the application...'
23              }
24          }
25      }
26      post {
27          always {
28              //
29          }
30          failure {
31
32          }
33      }
34  }
35
```

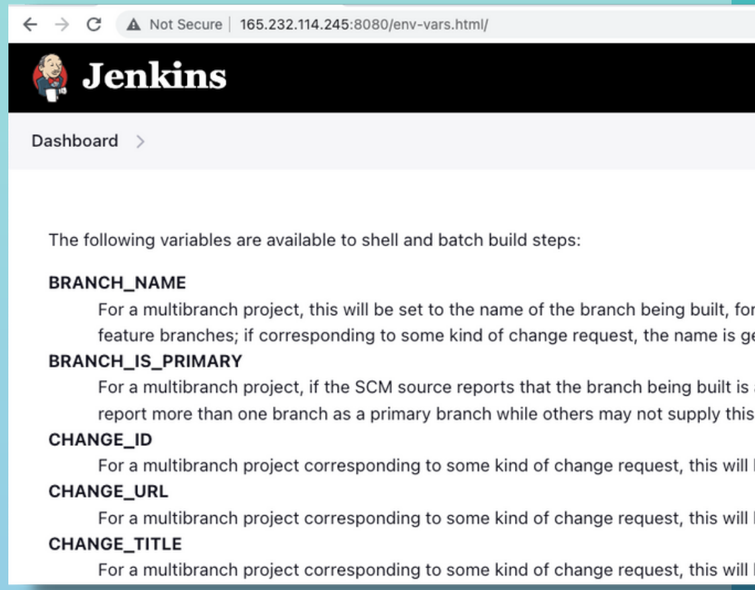
Conditions

- **Logical control flow** in stages of what to execute under which condition

```
pipeline {
    agent any
    stages {
        stage("test") {
            steps {
                script{
                    echo "Testing the application...
                }
            }
        }
        stage("build") {
            when {
                expression {
                    BRANCH_NAME == "master"
                }
            }
```

Environment Variables

- Jenkins offers information on build - some out of the box through **ENV VARS**
- See available env vars at */env-vars.html*



Jenkinsfile Syntax - 3

tools

- To access build tools for your projects, like npm, gradle, maven

```
Jenkinsfile Updates
1  pipeline {
2      agent any
3      tools {
4
5      }
6      stages {
7          stage("build") {
8              steps {
9                  echo 'building the appl
```

parameters

- You can parameterize your build to make it more re-usable
- **Types of parameter**:
  - string, choice, booleanParam

```
Jenkinsfile Updates
1  pipeline {
2      agent any
3      parameters {
4          choice(name: 'VERSION', choices: ['1.1.0', '1.2.0', '1.3.0'], description:'')
5          booleanParam(name: 'executeTests', defaultValue: true, description:'')
6      }
7      stages {
8          stage("build") {
```

Jenkinsfile Syntax - 4

Using external Groovy script

- For complete flexibility you can write any groovy script and include it in the Jenkinsfile

Jenkinsfile Updates ● script.groovy Updates ●

```
1  def gv
2
3  pipeline {
4      agent any
5      parameters {
6          choice(name: 'VERSION', choices: ['1.1.0'
7          booleanParam(name: 'executeTests', def
8      }
9      stages {
10         stage("init") {
11             steps {
12                 script {
13                     gv = load "script.groovy"
14                 }
15             }
16         }
17         stage("build") {
18             steps {
19                 script{
```

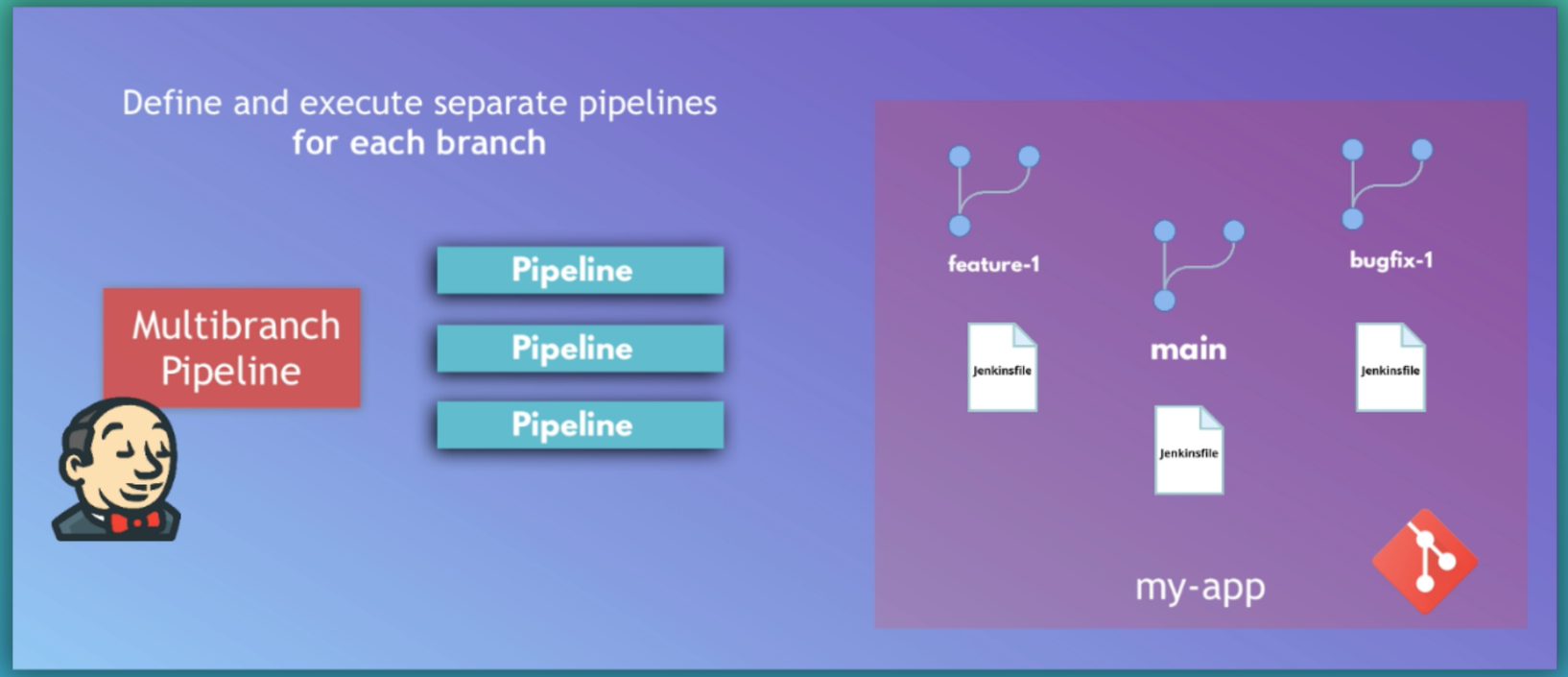
Jenkinsfile script.groovy ●

```
1   def buildApp() {
2       echo 'building the application...'
3   }
4
5   def testApp() {
6       echo 'testing the application...'
7   }
8
9   def deployApp() {
10      echo 'deplying the application...'
11      echo "deploying version ${params.VERSION}"
12  }
13
14  return this
```

Multibranch Pipeline

- Enables you to implement **different Jenkinsfiles for different branches** of the same project

- Jenkins automatically discovers, manages and executes Pipelines for branches, which **contain a Jenkinsfile** in a Git repository (no need to manually create the Pipeline)



Summary of all 3 Job Types

Freestyle

- Executing a single task

Pipeline

- Better solution for CI/CD
- 1 branch

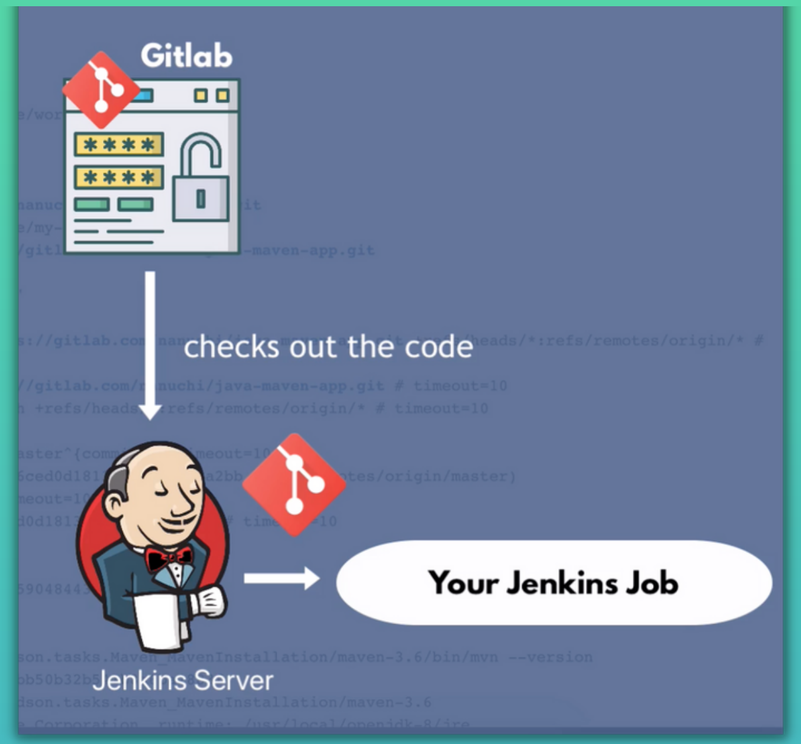
Multibranch Pipeline

- Parent of Pipeline Jobs

Jenkins - Important Terms

- **Job** = A deprecated term, synonymous with Project
- **Project** = A user-configured description of work which Jenkins should perform, such as building a piece of software, etc.
- **Build** = Result of a single execution of a Project
- **Pipeline** = A user-defined model of a continuous delivery pipeline
- **Plugin** = An extension to Jenkins functionality provided separately from Jenkins Core

Credentials - 1



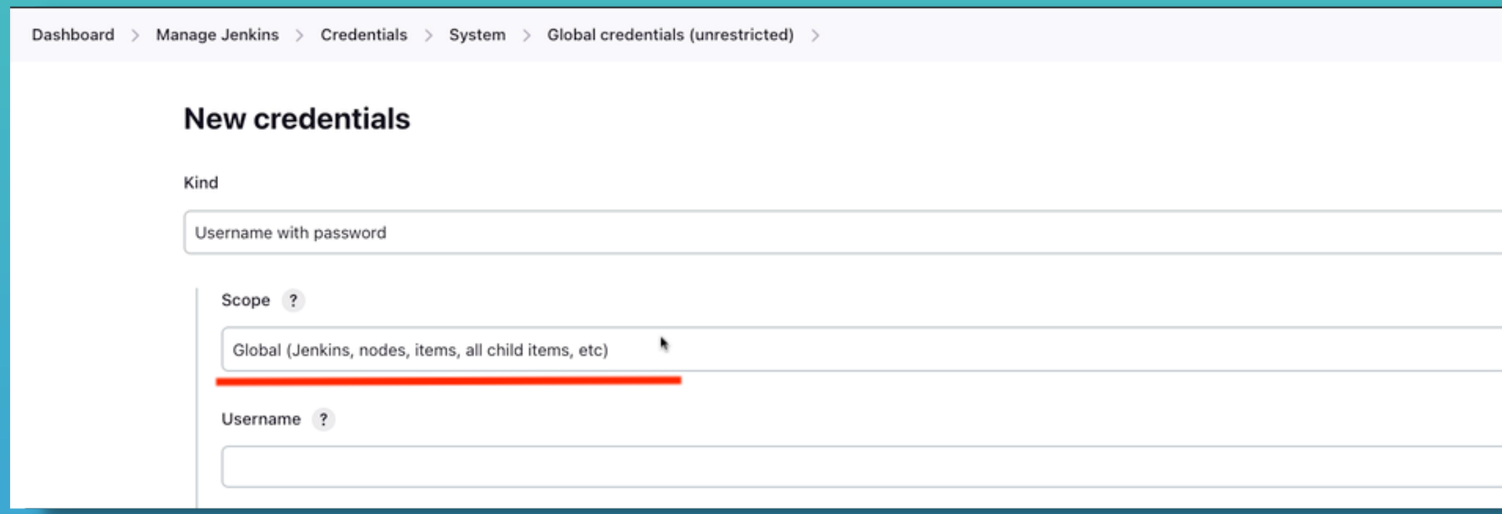
Jenkins **needs credentials for all the tasks** in the pipeline, such as:

- fetch code of git repository,
- login to docker registry to push images,
- SSH to remote server to deploy

Credentials Scopes

Credentials in Jenkins have a **scope**:

- **System**: Only available on Jenkins server, NOT for Jenkins jobs
- **Global**: Everywhere accessible



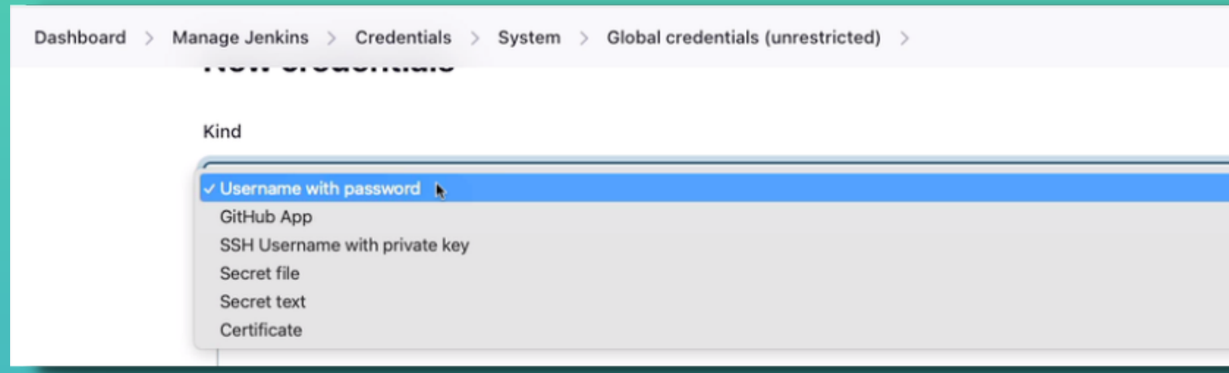
Credentials - 2

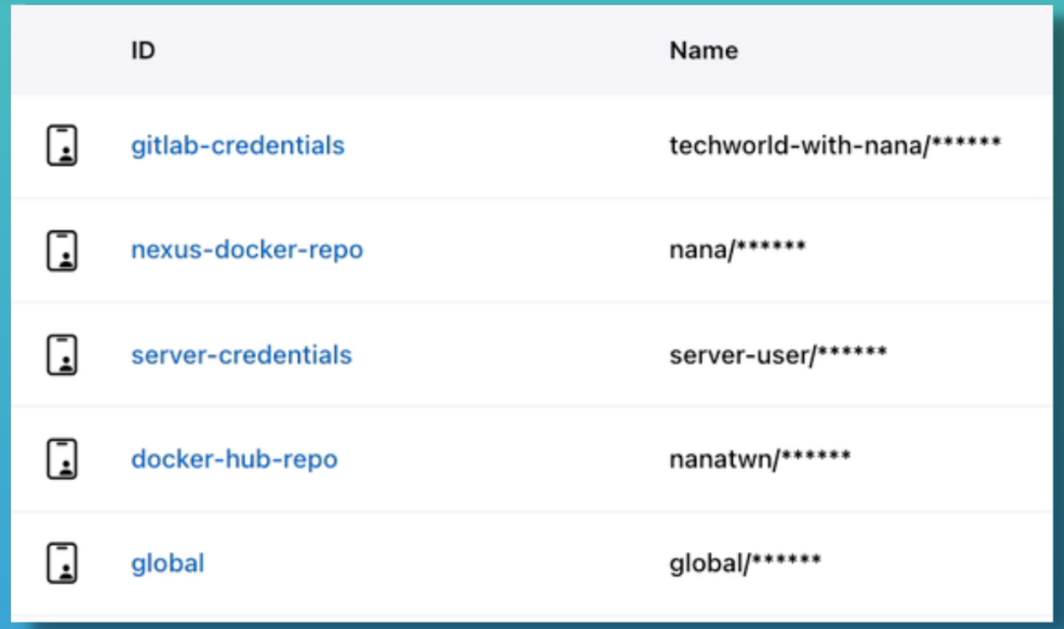
Credential Types

Jenkins can store the following types of credentials

- **Secret text** - a token such as an API token
- **Username and password** - which could be handled as separate components or as a colon separated string in the format username:password
- **Secret file** - secret content in a file
- **SSH Username with private key** - an SSH public/private key pair
- **Certificate** - a PKCS#12 certificate file and optional password
- **Docker Host Certificate Authentication** credentials.

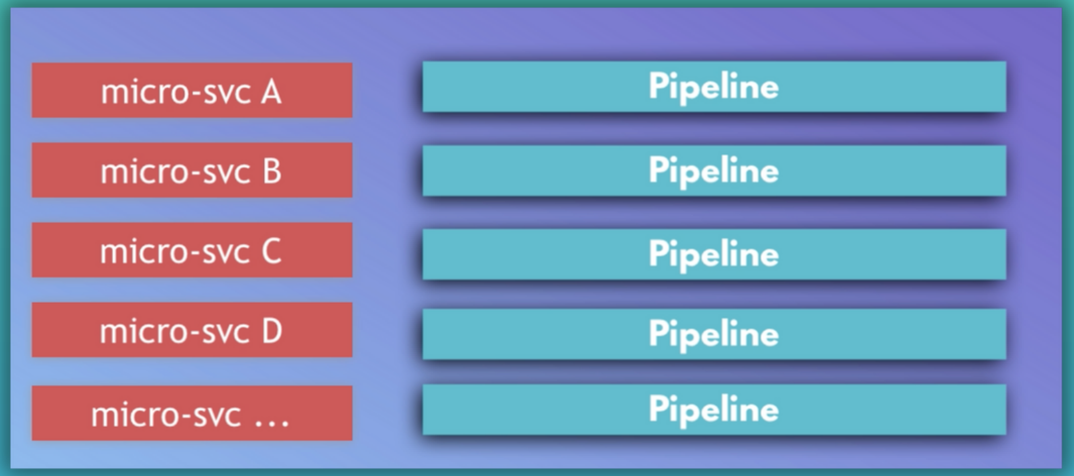
New types based on plugins



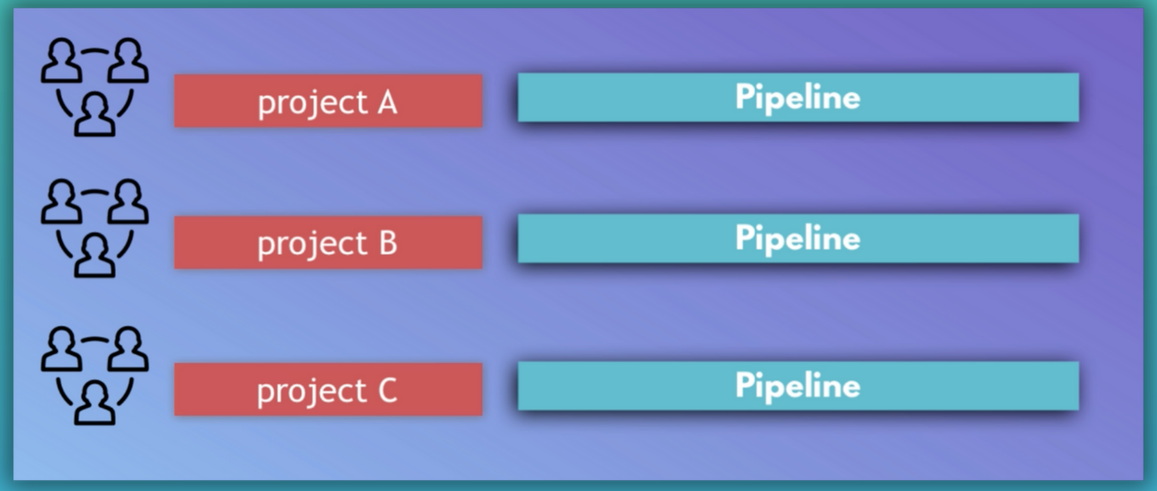


Jenkins Shared Library - 1

- **Used to share parts of Pipelines** between various projects to reduce duplication
- "Shared Libraries" can be defined in external git repositories and **loaded into existing Pipelines**



Use Case 1: Same logic in different microservices



Use Case 2: Many different projects in a company with some common tasks and logic

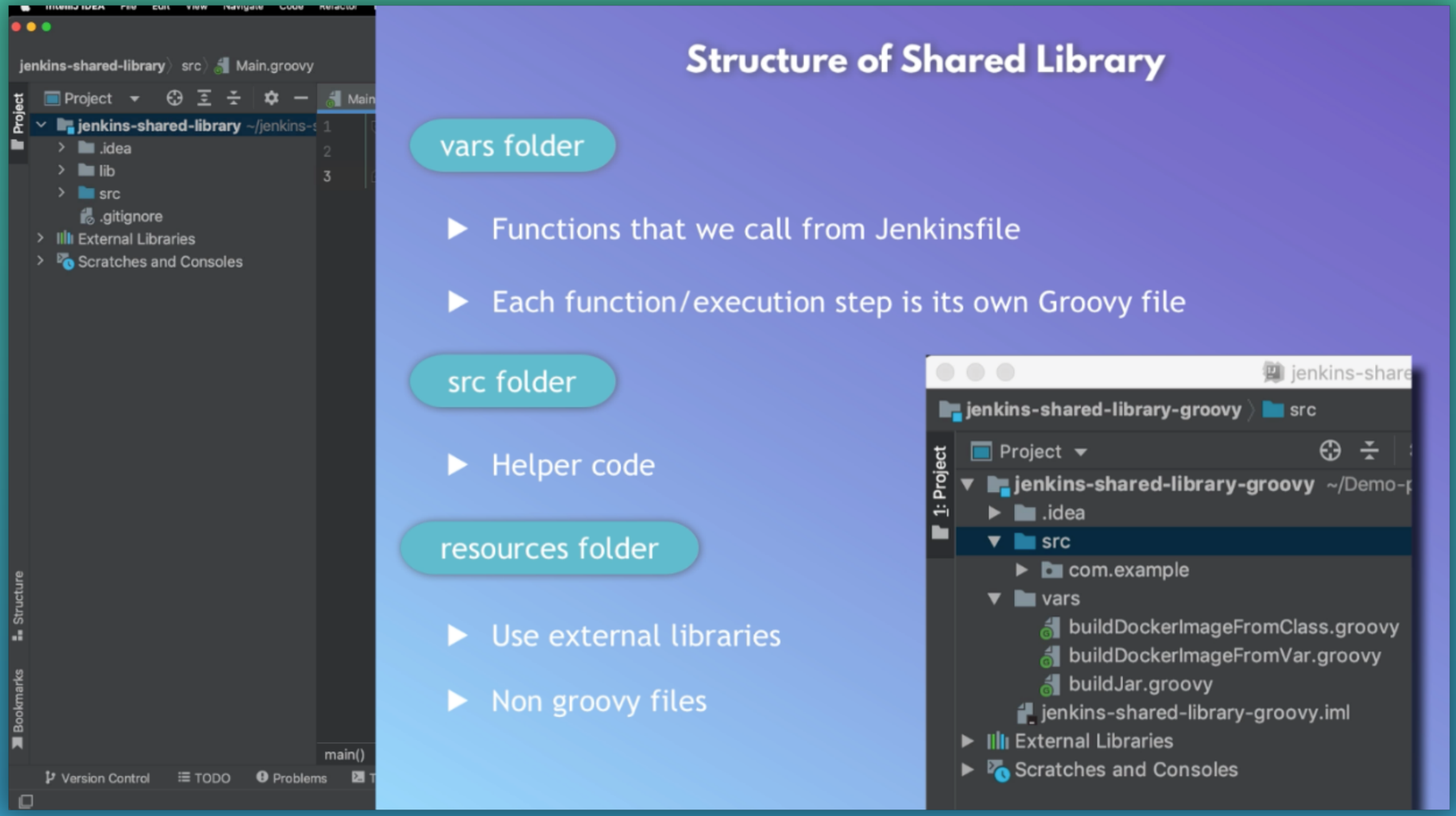
Jenkins Shared Library - 2

Steps to use Shared Library

1. **Create the Shared Library (SL)**
    a. Create Repository
    b. Write the Groovy Code
2. **Make the SL available in Jenkins globally or for project**
3. **Use the SL in Jenkinsfile to extend the Pipeline**

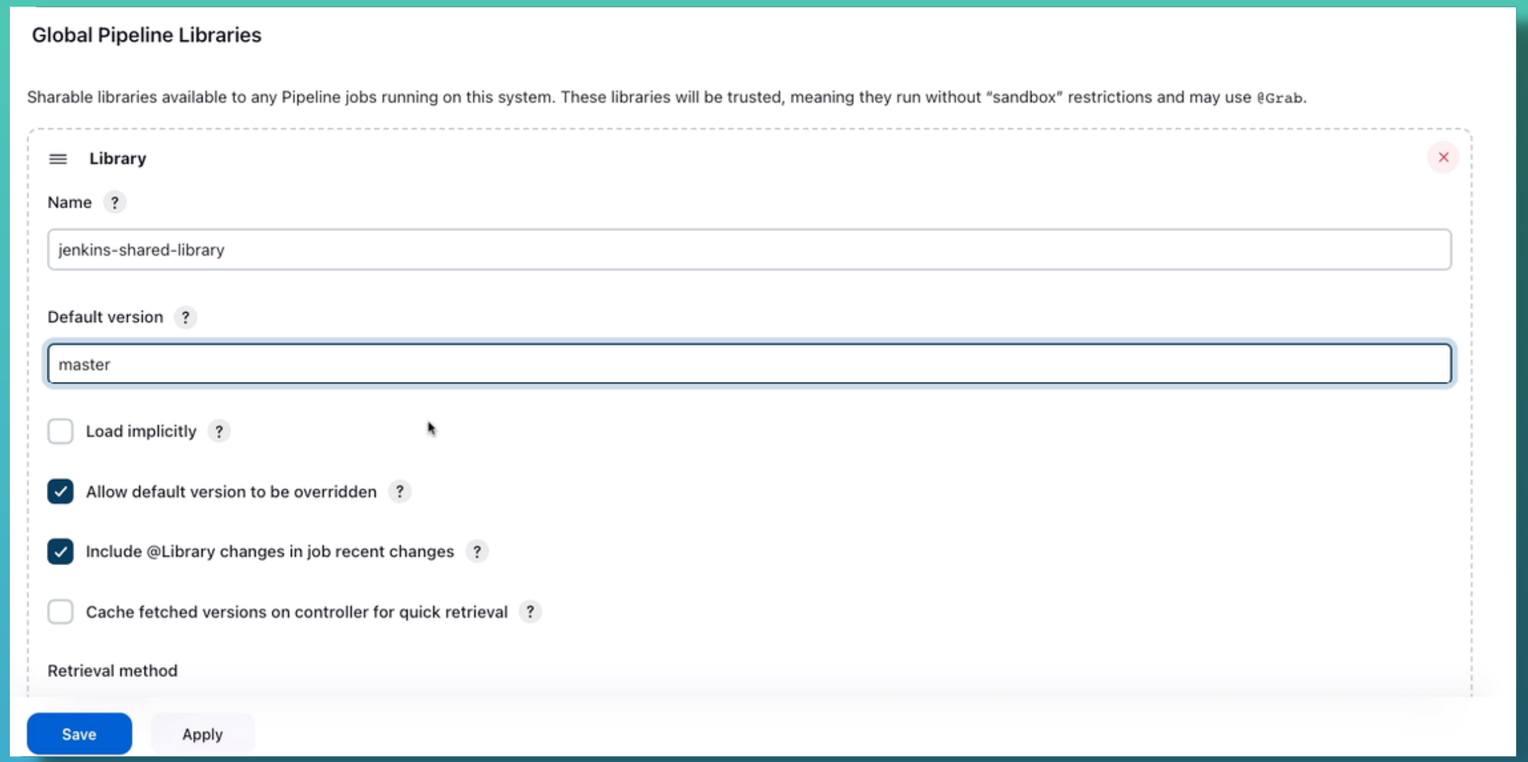
Jenkins Shared Library - 3

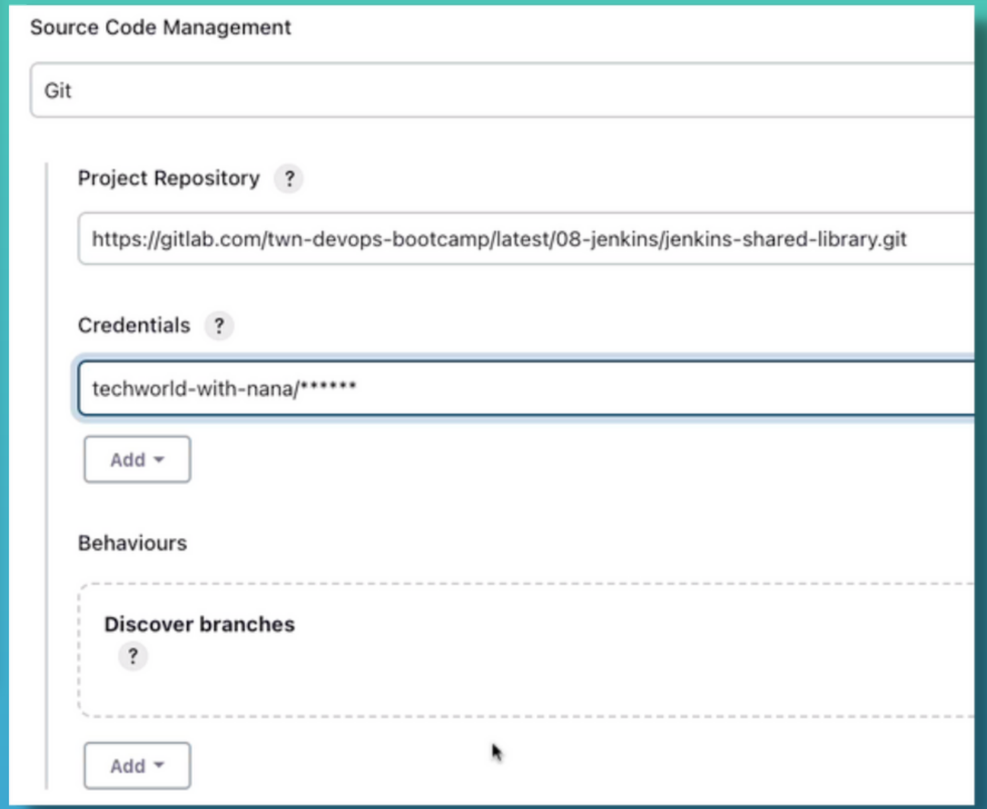
1) Create the Shared Library (SL)



Jenkins Shared Library - 4

2) Configure globally in Jenkins





Jenkins Shared Library - 5

3) Use the configured JSL in any of your pipeline scripts

Jenkinsfile | script.groovy

```
#!/user/bin/env groovy

library identifier: 'jenkins-shared-library@master', retriever: modernSCM()

def gv

pipeline {
    agent any
    tools {
        maven 'Maven'
    }
    stages {
        stage("init") {
            steps {
                script {
                    gv = load "script.groovy"
                }
            }
        }
```

TECHWORLD WITH NANA

Trigger Jenkins Job

3 ways to **trigger Jenkins jobs**:

Manually

- Use case may be **for production pipelines**

Automatically

- Trigger **automatically when changes happen in the git repository (Webhook)**
- Jenkins and Gitlab needs to be configured for that

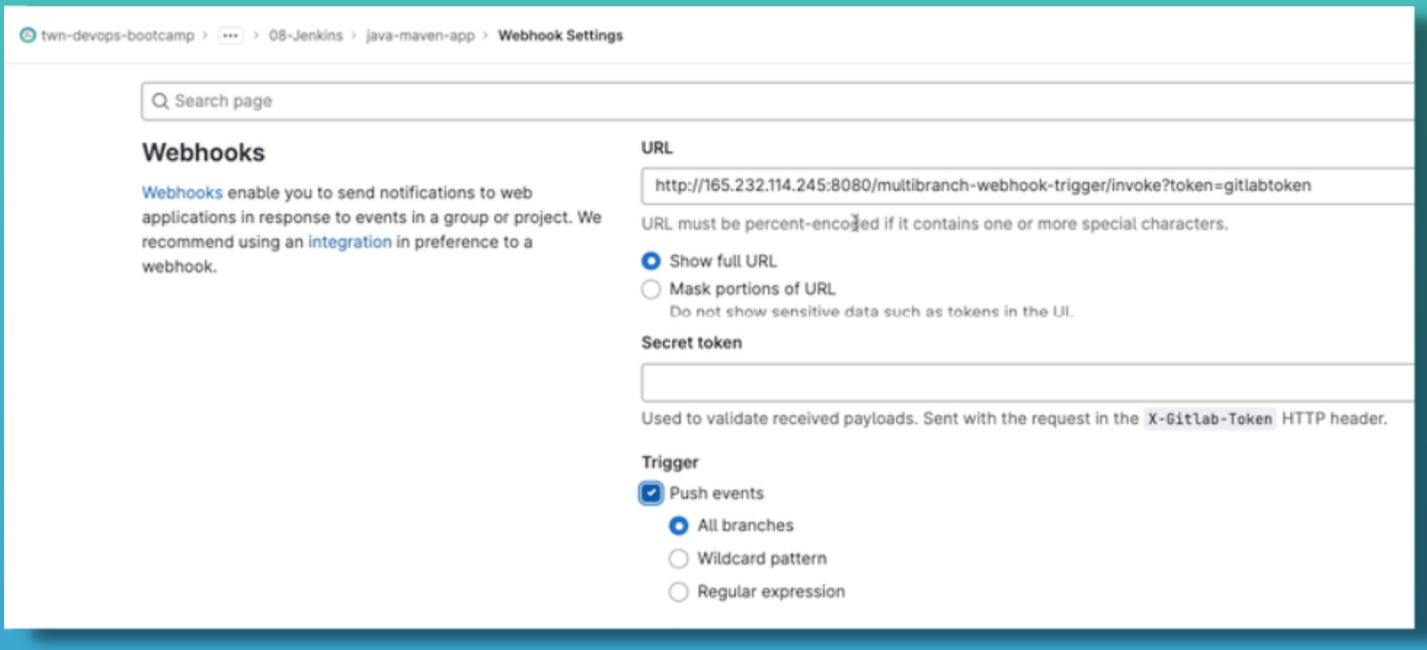
Scheduling

- Trigger job **on scheduled times**
- For example:
  - For long running tests
  - Tasks that need to run only weekly/monthly

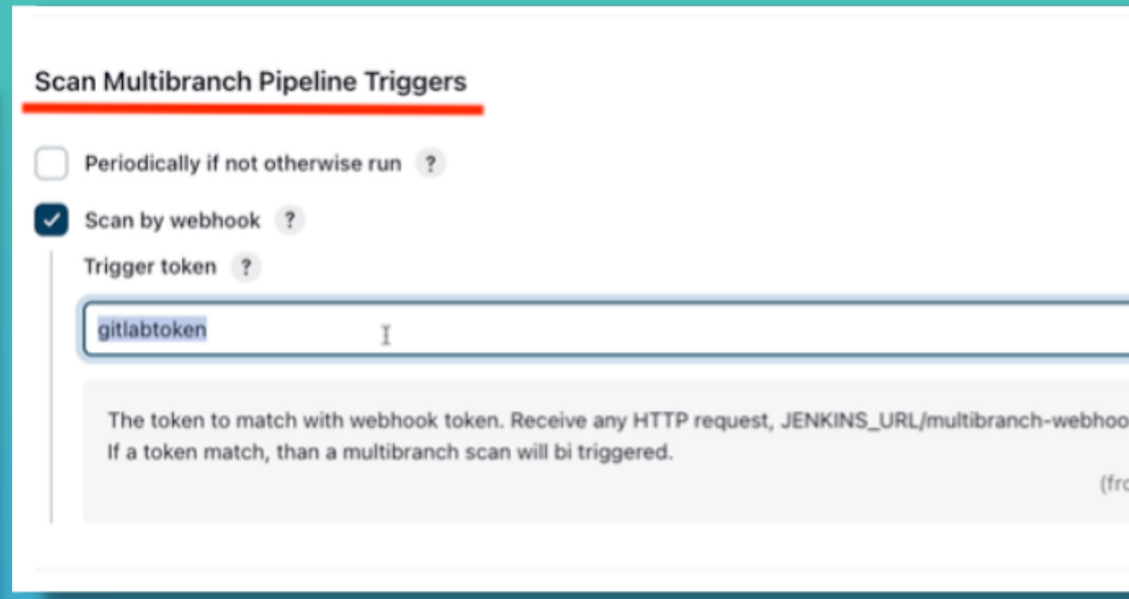
Webhook

- Mechanism to automatically trigger the build of a Jenkins project upon a commit pushed in a Git repository

Configure in Git repository:



Configure in Jenkins Job:



Software Versioning

Software Versioning

- Each **build tool** or package manager tool **keeps the project's version**

build.gradle

```
defaultTasks 'bootRun'

group = 'com.myapp'
version = '1.0.0-SNAPSHOT'

description = ''
```

package.json

```
{
  "name": "test-js-app",
  "version": "0.0.1",
  "description": "Beta",
  "private": true,
  "license": "UNLICENSED",
  "cacheDirectories": [
    "node_modules"
  ],
  "dependencies": {
    "@babel/polyfill": "7.8
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?
<project xmlns="http://maven.apache.o
         xmlns:xsi="http://www.w3.org
         xsi:schemaLocation="http://m
    <modelVersion>4.0.0</modelVersion

    <groupId>maven</groupId>
    <artifactId>test-maven</artifactI
    <version>1.0-SNAPSHOT</version>


</project>
```

- You and the team decide how you version your application

1.1.0-SNAPSHOT

3 Parts of a version

Major Version

- When you increase this part, it means the **new software version contains big changes**, breaking changes, NOT backward-compatible

Minor Version

- New, but backward-compatible changes, new API features

Patch Version

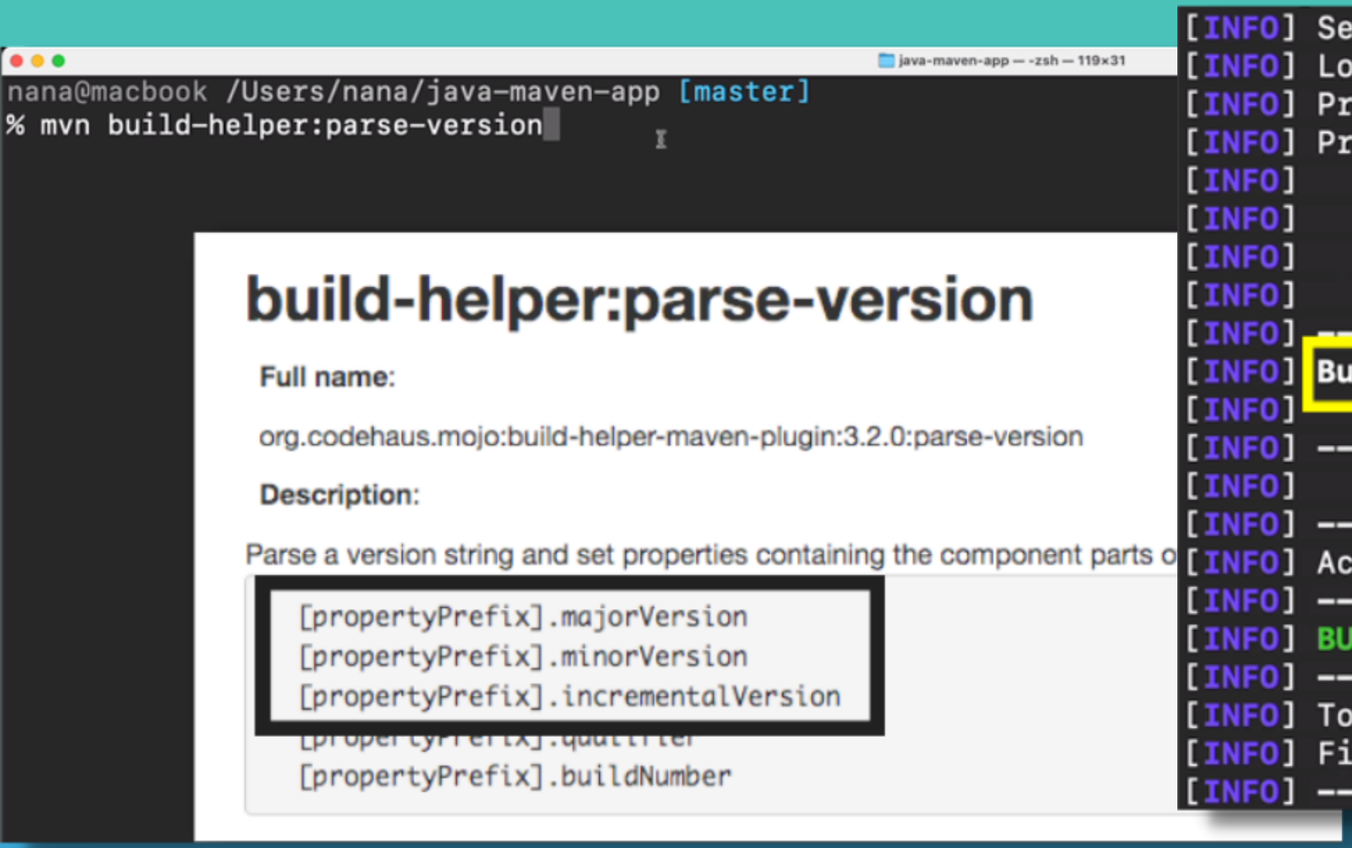
- Minor changes and **bug fixes**, doesn't change API

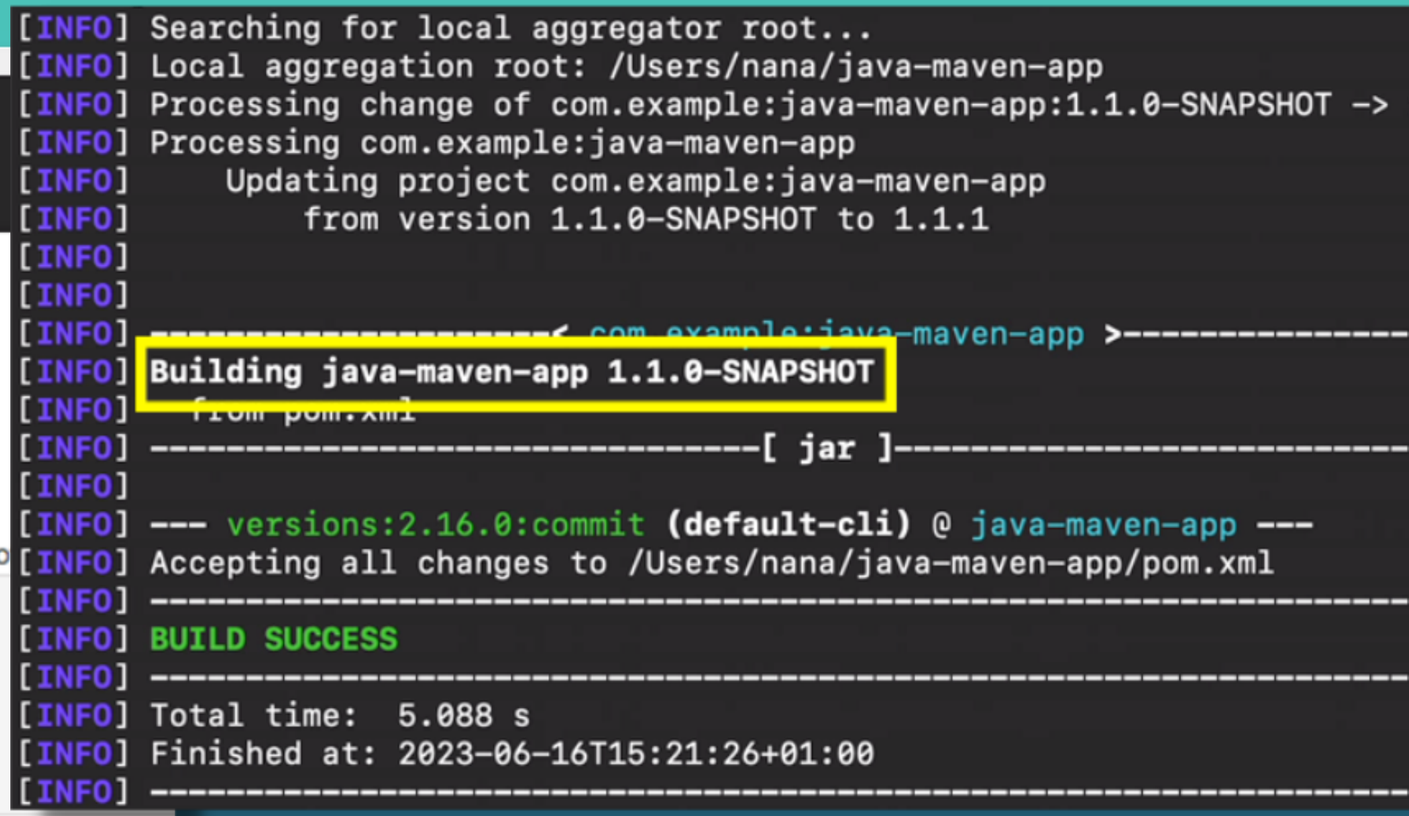
Suffix

- Gives **more information**
- For example, "SNAPSHOT" term means it's a "pre-release" version indicating that the version is **unstable** and still under development. When code is ready and it's time to release it, you will remove the "SNAPSHOT" suffix
- Another example "alpha"

Dynamically increment application version - 1

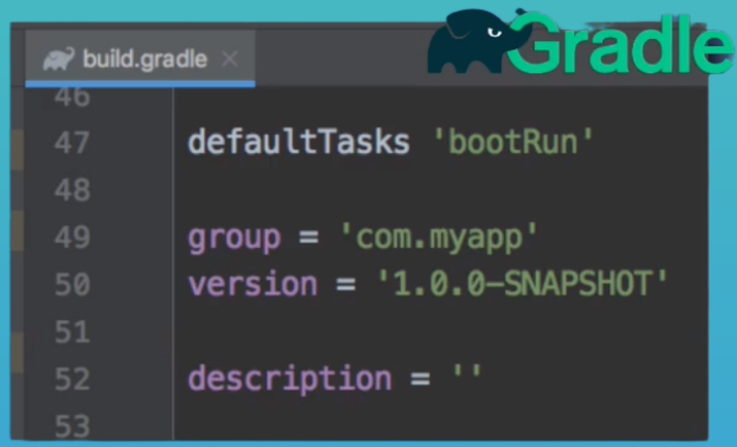
- Instead of updating the version manually, you will configure the build automation tool to **increment the version automatically**
- Build Tools have commands to increment the version





Dynamically increment application version - 2

- Different package manager/build tools have the **same concept**
- For npm, you can read the version from package.json file, increment and save back



TECHWORLD WITH NANA

Best Practices - 1

Pipeline Script/Jenkinsfile related best practices:

- Use Pipeline as Code: Store your Jenkinsfile in Git Repository (instead of inline script in Jenkins). Good for history, working in a team etc.
- Call your Pipeline Script the default name: Jenkinsfile
- Non-Setup work should occur within a stage block. Makes your builds easier to visualize, debug etc.
- Make sure to put #!/usr/bin/env groovy at the top of the file so that IDEs, GitHub diffs, etc properly detect the language and do syntax highlighting for you.
- Input Parameter: input shouldn't be done within a node block. It's recommended to use timeout for the input step in order to avoid waiting for an infinite amount of time, and also control structures (try/catch/finally).

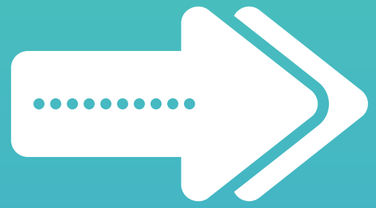
Best Practices - 2

Other best practices:

- Use Automatic versioning
- Store common pipeline code in a Shared Library, so that it can be used by other projects/teams

- More best practices and how to put it to practice: https://www.lambdatest.com/blog/jenkins-best-practices/

CI Part of CI/CD



CD Part of CI/CD in separate Module