**Udemy Course 2nd Video**

**(DevOps CI CD with Jenkins Maven Git and Pipeline in Windows)**

* **What is devops**
* Devops is a term used to refer to a set of practices that emphasize the collaboration and communication of both software developers and information technology professionals while automating the process of software delivery and infrastructure changes.
* **Different phases in devops:**
* Design: architecture prepare the overall solution and hand it over to the development team.
* Develop: the development team understand the solution and implements it. And from time to time, we check in the code to a central repo. The code is then merged and build periodically.
* Test: the QA team or test team perform test functionality and approves final delivery of code to the production.
* Package
* Deliver
* Monitor:
* **Three important points in Devops:**
* Continuous integration
* Continuous deployment
* Continuous delivery.

**Continuous integration:**

Continuous integration is a devops software development practice where developers frequently merge their code changes into a central repository, after which automated builds and tests run.

**Continuous deployment:**

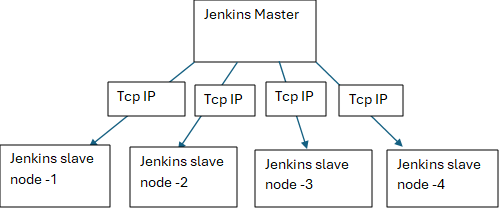
The code built in the CI process is automatically deployed to a higher environment, usually a staging/test environment.

**Continuous delivery:**

Code that passes functional and system tests is deployed to production.

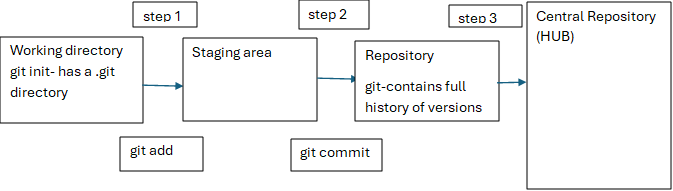
Typically, a test lead or the release manager triggers this manually

* **Popular Devops CI/CD tools**:
* Jenkins
* Circle CI
* Bamboo
* Travis and many more.
* **What is Jenkins?**
* A extremely powerful open-source automation tool that supports continuous build and deployments cycles in a software project.
* Has hundreds of plugins that make automation set up easy.
* Has support for nearly every scripting language like maven, ant, Gradle and many more.
* **Jenkins deployment architecture:**



* **Install and setup java environment:(In windows)**
* Go to browser
* Oracle java installation
* Click on java install
* Click on download
* Click on accept license agreement
* Click on windows 64
* Once downloaded click on installer
* Click on next
* Close the installer once complete the installation.
* Before start using Jenkins/java first we need set up environment variables
* Enter environment variables in the search bar
* Open
* Click on environment variables
* Double click on path
* Click on ‘NEW’
* Paste path here
* Click on ok.
* You want to verify the installation
* Open command prompt
* Enter java -version
* If java installed it shows version of JAVA.
* **Install Jenkins:**
* Go to google
* Enter Jenkins installer
* Click on Jenkins installation and setup
* Go to windows under long term support
* Click on download
* Go to download page and unzip it
* Click on next
* Click on install
* Click finish
* It automatically open and asked administrator password
* It provides above one link copy that one and paste it in cmd will get the password
* Copy and paste the password
* Click on select plugins to install
* Click None
* Click install
* Create first admin user password
* Click on start using Jenkins
* Jenkins page will be opened.
* **How to start a Jenkins in standalone java program**
* Go to file explorer and create a Jenkins home folder
* Go to environment variables and set the path of Jenkins home
* Click ok
* To see the Jenkins path applied or not
* Go to command prompt
* Enter (echo %JENKINS\_HOME%) it shows path of the Jenkins home
* Enter java -jar full path of the location where you downloaded WAR file --httpport=8090
* Enter
* Wait few seconds Jenkins start
* Go to browser
* Enter (<http://localhost:8080>)
* Enter
* Copy and paste administrator password
* Click continue
* Click on select plugins to install
* Click None
* Click install
* Create first admin user password
* Click on save and finish
* Click on start using Jenkins
* Jenkins page will be opened.
* **Install Tomcat:**
* Go to browser
* Enter tomcat installer windows
* Click on Apache tomcat@ download
* Click on 64-bit windows download
* we will get zip file of tomcat
* Unzip that file
* Open that file enter username and password
* Click next
* Click install
* Select run apache tomcat
* Click finish
* To see tomcat installed or not
* Open browser enter localhost:8080
* Enter
* Tomcat page will be displayed.
* Click on manager app
* Enter username and password
* Ok
* Some apps displayed you can start and stop.
* **Jenkins setup on a Tomcat:**
* Set environment variables (Jenkins path)
* Click on Jenkins .war file copy
* Paste it in tomcat folder
* Wait few seconds Jenkins folder will be created (this means app deployment successful)
* To versify this open browser enter <http://localhost:8080/jenkins>
* Copy the administrator password and paste it.
* Click continue
* Click on select plugins to install
* Click None
* Click install
* Create first admin user password
* Click on save and finish
* Click on start using Jenkins
* Jenkins page will be opened.
* **Jenkins UI walkthrough:**
* Click on new item (it is used create new jobs)
* Click on people (to see the list of users who can access Jenkins)
* Click on build history (it shows total builds history)
* Click on manage Jenkins (lot of options to manage Jenkins instance)
* **Create first Jenkins job hello world:**
* Go to browser
* Enter <http://localhost:8080>
* Enter username and password
* Jenkins dashboard will be displayed
* Click on new item
* Enter name for job
* Click freestyle project
* Click ok
* Click on build
* Click on add build step
* Select execute windows bash command
* Enter echo hello-world!!
* Click on save
* Click on build now
* You can see the output in console output.
* You Want to make any changes in code click on configure you can edit the code.
* **Git meaning:**
* It is free and open source distributed version control system.
* Used to manage source code, but it can be manage versions of any set of files.
* Very easy to learn and has a very strong community support and learning resources.
* **Meaning of git:**
* Git is a distributed version control system that enables multiple developers to collaborate on software projects efficiently.
* It is free and open source.
* It was created by Linux Torvalds in 2005.
* Git helps you keep track of code changes.
* It is version control system means it is also known as source control is the practice of tracking and managing changes to software code.

**Git states-three**



* Git hub account creation(completed)
* **Git installation:**
* Go to browser
* Enter git install
* Click on download
* Click on windows and download
* To check the git installed or not (git --version) in command prompt.
* After completion of installation
* We can set username and password.
* Git config --global user.name “Bhavya”
* Git config --global user.mail [bhavyasri@puropalecreations.com](mailto:bhavyasri@puropalecreations.com)
* This tells git who is making changes.
* **How to create repository and how to push file from local machine to remote:**
* Login into github account
* Click on new
* Enter repository name
* Click on add README file
* Put it in public or private
* Click on new
* Repository will be created.
* **Push files from local to remote location**:
* Git init
* Git add --all
* Git commit -m “message”
* Git branch -M main
* Git remote add origin <repository URL>
* Git push -u origin main.
* **Apache Maven:**
* Managed by Apache software foundation
* It supports java-based applications
* It supports small to large, based projects.
* **Why learn maven:**
* Totally automates dependency and library management for developers.
* Documentation generated with just a command.
* Devops tools like Jenkins have native support for Maven, so if you know maven, it would be an easier transition to devops CI/CD.
* Project infrastructure is built in an instant so that developers can focus more on development and not project structure.
* **Maven installation:**
* Open browser
* Enter maven-apache.download.org
* Click on download binary file link
* Extracted the file.
* We can see {M2\_HOME}
* bin
* boot
* conf-settings.XML
* Lib
* open environment variables
* Set the path of the maven.
* Path is set or not maven installed or not. By using below command we can see.
* Open command prompt
* Enter mvn -version.
* **Maven core concepts:**
* **Pom.xml file:**
* Pom.xml stands for project object model and .xml means Xtensible markup language
* Must be present in the projects root directory
* Root element <project>
* Contains everything needed to build a project using maven.
* **Maven coordinates:**
* The first few elements of pom.xml file makes maven coordinates.
* The first few elements groupid, artifactid, version from the coordinates.
* It marks a specific place in a repository and hence called coordinates.
* **Maven goals:**
* A goal is an action that maven performs in a phase.
* A plugin is a collection of goals
* **Maven plugins:**
* Need to information more about single plugin mvn help:describe -Dplugin=compiler
* **Maven lifecycle:**
* Mvn clean
* Mvn validate
* Mvn compile
* Mvn test
* Mvn package
* Mvn verify
* Mvn install
* Mvn deploy
* **maven repositories:**
* There are three repositories:
* Remote
* Central
* Local
* **Dependency management:**
* This section contains main libraries the project depends on
* **Convention over configuration:**
* Every artifact of a project has its own place.
* Source files
* Test files
* Packaged jar/war files
* Pom.xml
* Installed files
* Repositories.
* Maven hello world:
* **What is archetype?**
* Archetype is a maven project templating toolkit.
* In maven, an archetype is a template used for creating new projects.
* It provides a structured way to set up a project with a predefined configuration, directory layout, and basic files.
* Mvn archetype:generate.
* This command in maven is used to create a new project based on a specified archetype. This command sets up the project structure and generates the necessary files according to the selected archetype template.
* **Convention over configuration:**
* Every artifact of a project has its own place.
* Source files-src/main/java-source code
* Test files-src/test/java-test code
* Packaged files-target directory-war/jar/ear
* Pom.xml-stored in root directory
* **Eclipse IDE for maven installation:**

* Go to browser
* Enter Eclipse download
* Eclipse IDE for java EE developers
* Click download 64bits in windows
* Unzip it
* Open it
* Go to file
* Click on import
* Click on existing maven projects
* Next
* Click on project location
* Finish.
* After completion of import project name will be displayed on the left side.
* **Creating a web application using maven:**
* Open java IDE
* File-new-maven project-browse path
* select maven archetype webapp
* Click next
* Enter maven coordinates (groupid,artifactid,version,package)
* Click finish
* New web app created
* After giving mvn clean package war file created.
* **Assignment solution:**
* Click on com.learnmavendemo
* New
* Class
* Enter project name(calculator)
* Finish
* Add four functions in calculator(add,mul,division,sub)
* Go to app.java
* Go to browser download maven log 4g
* Download latest version
* Copy the dependency path and come to java IDE -open pom.xml file paste the path in the dependency.
* Go to app.java (import org.apache.log4j.Logger;
* **Multi module project:**
* **Compiler plugin:**
* Mvn help:describe -Dplugin=compiler (it gives all the details of the compiler plugin and all the goals description etc)
* If you want details of particular goal in the plugin you can run below command

mvn help:describe -Dcmd-compiler:compile -Ddetail=true

* How can we modify the plugin behaviour
* Go to browser
* Enter maven compiler plugin
* click on first link
* take the code and paste in the dependency under create build tag
* Mvn compiler:compile.
* **Mvn checkstyle plugin:**
* Mvn checkstyle:checkstyle (download all the dependencies and libraries)
* **Surfire plugin.**
* **Add maven project to git version control:**
* Login to github account
* Create on new repository
* Open command prompt window
* Navigate to maven workspace
* Git init
* Git add --all
* Git commit -m “project”
* Git remote add origin “repo url”
* Git push origin main
* **Jenkins continuous integration poll git repository**
* Open jenkins dashboard
* Login into username and password
* Click manage jenkins
* Click plugins
* Click available plugins
* Click git plugin
* Click install plugin without restart
* Once installed go back to dashboard
* Click new item
* Enter name
* Click freestyle
* Ok
* Go to source code management
* Select git
* Copy and paste GitHub repo url
* In build section execute windows batch command
* Enter code (echo downloaded code)
* Apply and save
* Click on build now
* Click on console output to see the output.
* Go to configure page click advanced
* Click custom workspace
* Give a directory ($JENKINS\_HOME)/workspace/mavenproject
* Save
* Click build now
* **You can edit build run automatically:**
* Go to configure
* Go to build triggers
* Select pollscm
* select every minute.
* **Jenkins maven integration:**
* Click manage jenkins
* Click global tool configuration
* Click add maven
* Copy and paste maven directory (echo %M2\_HOME%)
* Apply and save
* **Static code analysis on Jenkins:**
* We need to install check style project
* Click manage Jenkins
* Click manage plugins
* Available plugin
* Select check style plugin and click without restart install
* Click on new item
* Enter project name
* Select freestyle
* Ok
* Go to configure page click advanced
* Click custom workspace
* Give a directory ($JENKINS\_HOME)/workspace/mavenproject/mavendemo
* In build section select invoke top level maven targets
* Give version name
* Under goal enter clean checkstyle:checkstyle
* In post build actions select publish checkstyle analysis results
* Give result checkstyle name as \*\*/checkstyle-result.xml
* Apply and save
* Go to workspace oprn app.java file
* Change capital LOGGER to logr -save
* Package -info.java (covert to package- infojava)
* Click build now
* Open pom.xml file
* Add reporting line
* Save and build now (got two errors)
* Go back and undo those errors (logr-logger and package -infojava into package -info.java)
* Build now.
* **Jenkins continuous testing Junit:**
* Go to manage jenkins
* Click on manage plugins
* Go to available plugins
* Search JUnit Realtime test reporter plugin
* Install without restart
* Save
* Click new item
* Junit
* Select freestyle
* Copy from (enter code review)
* Ok
* Go to configure page click advanced
* Click custom workspace
* Give a directory ($JENKINS\_HOME)/workspace/mavenproject
* Give version name
* Under goal enter test
* In post build actions
* Give test report xmls
* Target/surefire-reports/\*.xml
* Save
* Click build now
* Open pom.xml file
* Set skiptests<false>
* Save file
* Go to jenkins click on build now.
* Go to workspace
* Open app.javatest file
* In subtract tag give value 10
* Save file
* Go to jenkins build now.
* **Commit changes to GitHub repository:**
* Git commit -am “committing changes”
* Git remote -v (it gives remote repository names)
* Git push origin main (this pushes all the changes to the GitHub)
* **Jenkins-Deploy to stage environment:**
* Under conf directory
* Open server.xml file (see port number)
* Open tomcat-users.xml (see admin credentials)
* go to jenkins
* Click manage jenkins
* Click manage plugins
* Go to available
* Search deploy to container plugin
* Install without restart
* Once installed
* Go to dashboard
* Click on new item
* Enter name (stage deploy)
* Select freestyle project
* Copy from code review
* Click on ok
* Go to configure page click advanced
* Click custom workspace
* Give a directory ($JENKINS\_HOME)/workspace/mavenproject/mavendemo/webapp
* Under build section
* Under goal enter package
* Under post build actions select Deploy war/ear to a container.
* Give war file name(\*\*/\*.war)
* Enter context path (/mavenwebapp-demo)
* Click add container (tomcat 7.x)
* Add jenkins
* enter username and password(admin and admin)
* Add
* Go to bin directory
* Select tomcat 7 exe
* Go to browser(localhost:8090)
* Now back in jenkins give tomcat URL as (<http://localhost:8090>)
* Save
* Click build now.
* **Jenkins build pipeline: deploy to stage environment:**
* Build pipeline is a way of time jobs together so that they run in sequence one after the other. It offers a pipeline view of linked jobs
* First we need to install the build pipeline plugin
* Go to manage Jenkins
* Click on manage plugins
* Go to available
* Enter build pipeline plugin in search bar
* Install without restart
* Save
* Open gitcheckout job
* Configure
* click on post build actions
* Select build other projects
* In the text box select code review
* Save
* It means once the checkout job triggered after that code review job triggered automatically
* Click on code review job
* Click on configure
* click on post build actions
* Select build other projects
* Add unit test job
* Save
* Open unit test job
* Click on configure
* click on post build actions
* Select build other projects
* Add stage deploy
* Save
* Once all jobs saved create view (maven\_pipeline)
* Select build pipeline view
* Ok
* In pipeline flow select initial job(gitcheckout)
* Ok
* **Deploy to production:**
* In tomcat installation modify port number 9090
* Verify tomcat running or not (<https://localhost:9090>)
* Go to jenkins
* Click new item
* Enter name deploy to prod
* Select freestyle
* copy from stage deploy
* Ok
* Remove build step
* Give war file name(\*\*/\*.war)
* Enter context path (/mavenwebapp-demo)
* Click add container (tomcat 7.x)
* Add Jenkins
* enter username and password(admin and admin)
* Add
* Go to bin directory
* Select tomcat 7 exe
* Go to browser(localhost:8090)
* Now back in Jenkins give tomcat URL as (http://localhost:9090)
* Save
* Click build now
* **Email notifications:**
* **How to integrate email notifications into your jobs**
* Go to manage Jenkins
* Click on configure system
* Go to smtp server
* Enter smtp.gmail.com
* Click use SSL checkbox
* Give smtp port 465
* Click on use smtp authentication
* Username and password
* Click on test configuration by sending test email by sending test email recipient
* click on test configuration
* **Before you send the email, we need one plugin:**

Go to manage Jenkins

Click manage plugins

Available

Email extension plugin

Install without restart

Ok

Click on prod deploy job

Configure

Under post build section

Editable email notification

under project recipient list keep you email address

Project reply to [list-noreply@example.com](mailto:list-noreply@example.com)

content type-html

default subject as production deployment of maven webapp

Default content-deployment log attached

Select attach build log

Save

Build now.

* **Master slave architecture in windows:**
* Manage jenkins
* Manage nodes
* Click new node
* Enter name (slave -1)
* click permanent agent
* Ok
* give remote root directory of jenkins (c:/jenkins)
* Launch method-lauch agent via execution of command on the master
* Launch method- Java-jar path of jar file
* click on slave
* Run the gitcheckout job build now.
* **Java network launch protocol:**
* Click on configure global security
* Select random
* Save
* Create another node -slave\_jnlp
* Permanent agent
* Ok
* Give root directory c:/jenkins
* Launch method-launch agent via java web start
* Click save.
* **Master slave setup for Linux:**
* Need two servers
* One is Jenkins master
* Another is Jenkins agent
* To add slave node
* Connect jenkins master server(public ip :8080)
* Go to manage jenkins
* Click on nodes
* Click new node
* Enter node name (Build\_node\_1)
* Permanent agent
* Click create
* Give description
* Number of executors(2)
* Remote rot directory(/var/jenkins)
* By using labels we can identify nodes
* We can use same label to multiple nodes
* Go to build server and connecting with gitbash
* Install java (same version in master that version only we need to install)
* Go to jenkins after clicking node one window will be opened.on that file install agent.jar file and copy the link and paste it into the build node server
* Agent.jar file we need to copy
* Execute that big command in node to gitbash
* Connected
* Agent node connected with master node
* We can run jobs in the agent node
* Go to dashboard
* Click on new item
* Enter job name(remote\_job)
* Freestyle project
* Ok
* Select agent purpose-restrict where this project can be run
* Label name enter(java\_node)starting we gave this name.
* Under build
* Select execute shell (uptime-to identify the build node uptime)
* Apply
* Save
* **Jenkins pipeline as code:**
* **What is pipeline in jenkins:**
* In Jenkins, a pipeline is a suite of plugins that supports implementing and integrating continuous delivery pipelines into Jenkins.
* Pipelines automate the delivery process, reducing manual effort and improving efficiency.
* We need one pipeline plugin
* Go to jenkins
* Click manage jenkins
* Click manage plugins
* Go to available
* Search pipeline plugin
* Click install without restart
* Save
* Go to new item
* Enter job name
* Click on pipeline
* Ok
* Under pipeline
* Give pipeline script
* Enter pipeline script
* Save
* Click build now
* **Deploy pipeline into production:(In windows)**
* Go to item
* Enter name(CI/CD pipeline)
* Select pipeline
* Ok
* Under pipeline section
* Write pipeline script
* Pipeline{
* Agent any
* Stages{
* Stage(‘GitCheckout’){
* Steps{
* Click pipeline syntax
* Select checkout:checkout from version control
* Select GIT
* Copy and paste project build repository URL.
* Click on generate pipeline script
* Copy and paste in the after checkout
* }
* }
* Stage(‘code review’){
* Tools{
* Maven\_3.2.5
* }
* Steps{
* Dir(‘java\_project){
* Sh ‘mvn checkstyle:checkstyle’
* Create new job
* Go to new item
* Give job name(publicheckstyleresult)
* Select freestyle project
* Ok
* build job: 'PublishCheckStyleResult'
* }
* }
* }
* stage('UnitTest'){
* tools{
* maven 'Maven\_3.5.3'
* }
* steps{
* dir('mavendemo'){
* sh 'mvn test'
* junit 'target/surefire-reports/\*.xml'
* }
* }
* }
* stage('Deploy'){
* steps{
* timeout(time: 1, unit: 'DAYS'){
* input 'Deployment Approved?'
* }
* echo 'Deployed to Production successfully'
* }
* }
* }
* }