JENKINS

→What is Jenkins?

- JENKINS IS A DEVOPS TOOL THAT AUTOMATES THE PROCESS OF BUILDING, TESTING, AND DEPLOYING SOFTWARE
- HELPING TEAMS DELIVER HIGH-QUALITY APPLICATIONS FAST.

→Why jenkins?

- IT IS AN OPEN SOURCE AND JENKINS SIMPLIFIES THE CI CD PROCESS AND IT HAS VARIOUS SUPPORTED PLUGINS WHICH HELPS IN MAKING TASKS FASTER WITHOUT ERRORS
- CI CD IS CONTINUOUS INTEGRATION AND CONTINUOUS DELIVERY/CONTINUOUS DEPLOYMENT
- ➢ CI- Continuous Integration:
- WHEN A COMMIT OCCURS ON GITHUB REPO THEN WITH THE HELP OF WEBHOOKS AND BASED UPON EVENTS JENKINS JOB IS AUTOMATICALLY TRIGGERRED WHICH WILL EXECUTE AND BUILD WAR PACKAGE AND PUSH THOSE TO ARTIFACTORY LOCATION.
- ➢ CD-Continuous Delivery:
- THIS MAKES SURES THAT THE ENVIRONMENT IS READY FOR DEPLOYMENT BUT IT WON'T DEPLOY THE PACKAGE NEEDS MANUAL INTERVENTION (APPROVAL)
- CD-Continuous Deployment:
- THIS MEANS WHENEVER THE PACKAGE IS CREATED AND READY IT WILL AUTOMATICALLY DEPLOY IT TO END ENVIRONMENT.

→Pre-requisite for jenkins is JAVA

→Jenkins runs on Port no : 8080

→Jenkins file?

- A TEXT FILE WHICH CONSISTS OF DIFFERENT STAGES
- EACH STAGE WILL HAVE SET OF TASKS TO BE EXECUTED

→Important Jenkins points to Remember

- Default port: 8080
- Jenkins home directory: /var/lib/jenkins
- Jenkins port config file: /etc/sysconfig/jenkins
- Jenkins secrets key: /var/lib/jenkins/secrets/initialAdminPassword
- Jenkins Task logs: /var/lib/jenkins/logs
- Jenkins app logs: /var/log/jenkins/jenkins.log
- Jenkins jobs: /var/lib/jenkins/jobs
- Jenkins workspace: /var/lib/jenkins/workspace

→Plugin in Jenkins?

- PLUGINS ARE EXTENSIONS WHICH HELPS IN MAKING JOB EASIER BY JUST DOWNLOADING THE PLUGINS AS PER OUR REQUIREMENTS.
- A **PLUGIN** IS A SOFTWARE COMPONENT THAT ADDS SPECIFIC FEATURES OR FUNCTIONALITIES TO AN EXISTING COMPUTER PROGRAM. IT'S LIKE A MINI EXTENSION THAT ENHANCES WHAT THE MAIN SOFTWARE CAN DO.

→SOME JENKINS PLUGIN NAMES:

- ➢ EX. Docker , Role Based Access , Git JAVA Maven , Slack , SonarQube , Sonar Scanner

→Jenkins Pipeline?

- JENKINS PIPELINE WILL BE EXECUTED BASED ON JENKINS FILE , WHERE DIFFERENT STAGES WILL GET EXECUTED
- THERE ARE TWO TYPES OF PIPELINE

➢ Declarative Pipeline: DECLARATIVE PIPELINES ARE EASIER TO READ AND MAINTAIN, IDEAL FOR SIMPLE USE CASES

➢ Scripted Pipeline: SCRIPTED PIPELINES OFFER GREATER FLEXIBILITY AND CONTROL FOR MORE COMPLEX SCENARIOS

SAMPLE SKELETON FOR DECLARATIVE	SAMPLE SKELETON FOR SCRIPTED

pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        echo 'Building...'
      }
    }
    stage('Test') {
      steps {
        echo 'Testing...'
      }
    }
    stage('Deploy') {
      steps {
        echo 'Deploying...'
      }
    }
  }
  post {
    always {
      echo 'This will always run'
    }
    success {
      echo 'This will run only if successful'
    }
    failure {
      echo 'This will run only if failed'
    }
  }
}
	node {
	  stage('Build') {
	    echo 'Building...'
	  }
	  stage('Test') {
	    echo 'Testing...'
	  }
	  stage('Deploy') {
	    echo 'Deploying...'
	  }
	  // Post actions
	  always {
	    echo 'This will always run'
	  }
	  success {
	    echo 'This will run only if successful'
	  }
	  failure {
	    echo 'This will run only if failed'
	  }
	}

• DECLARATIVE PIPELINE IS CONFIGURED WITHIN JENKINS FILE AND STORED IN CENTRAL REPOSITORY (GITHUB) • IT STARTS WITH PIPELINE AND IF THERE IS ANY ISSUE IN PIPELINE THEN THE MOMENT PIPELINE IS TRIGGERED IT WILL THROUGH ERROR	• SCRIPTED PIPELINES ARE CONFIGURED INSIDE THE JOB • IT WILL START WITH NODES AND IF THERE IS AN ERROR IN LINE 20 THEN IT WILL EXECUTE ALL 19 LINES AND THEN THROW ERROR ON 20TH LINE

→Types of Jenkins jobs/projects

- **Freestyle projects**: THE DEFAULT AND MOST FLEXIBLE PROJECT TYPE.
- **Maven projects**: USED TO BUILD MAVEN PROJECTS.
- **Pipeline projects**: DEFINED USING CODE IN A JENKINSFILE.
- **Multi-branch pipeline projects**: AN EXTENSION OF A PIPELINE JOB.
- **Organization folders**: USED TO ORGANIZE OTHER PROJECTS.

→Backup of jenkins server

- **TWO WAYS WE CAN TAKE BACKUP**

➢ THIN BACKUP PLUGIN

➢ USING BASH SCRIPT (CREATING A BACKUP MANUALLY AND SENDING IT TO OTHER LOCATION)

→Jenkins master and slave

- THIS IS BASICALLY FOR HIGH AVAILABILITY OF JENKINS SERVER
- WE CAN OFF LOAD THE WORK FROM MASTER TO SLAVE AND MASTER CAN ONLY BE USED TO PASS ORDER ON SLAVE NODES
- SLAVE NODES ARE ALSO RESPONSIBLE FOR SPECIFIC JOBS

  <For example, if we have two applications java and python then we can configure two slaves one for java and one for python>

→labels in slave:

- LABELS ARE NAME TO DEFINE OUR SLAVE MACHINE
- LABELS PLAY A VERY IMPORTANT ROLE WHEN DEDICATING A JOB TO PARTICULAR SLAVE

→Executers in Jenkins:

- IF WE HAVE 3 EXECUTERS THEN IT MEANS WE CAN RUN 3 JOBS AT SAME TIME

→Parameterized job:

- IT HELPS US TO PASS THE PARAMETERS IN JENKINS JOB
- FOR EXAMPLE IF WE WANT TO DEPLOY SAME JOB IN DIFFERENT ENVIRONMENTS THEN INSTEAD OF CREATING MULTIPLE JOBS WE CAN USE PARAMETRIZED JOB

*SIMPLE DEFINITION : By using a parameterized job in Jenkins, you can run the same job on different repositories or branches by simply changing the input parameters (like branch name or repository URL) each time you run the job. This way, you don't need to create separate jobs for each branch or repository. *

→WEBHOOKS:

- ALLOW JENKINS TO BE NOTIFIED OF CHANGES IN THE REPOSITORY AND AUTOMATICALLY TRIGGER BUILDS

→POLL SCM:

- PERIODICALLY CHECKS THE REPOSITORY FOR CHANGES AND TRIGGERS BUILDS IF NEW COMMITS ARE DETECTED.
-

<The main difference is that webhooks push notifications to Jenkins, whereas Poll SCM pulls updates by checking the repository at regular intervals.>

→Global jenkins Variable:

- **BUILD_NUMBER**: The current build number, unique to each build.
- **BUILD_ID**: A unique identifier for the build.
- **JOB_NAME**: The name of the job being executed.
- **BUILD_TAG**: A string of the form `jenkins-${JOB_NAME}-${BUILD_NUMBER}`, useful for tagging builds.
- **NODE_NAME**: The name of the node (slave) on which the job is running.
- **JOB_URL**: The URL for the job in Jenkins.
- **AWS_REGION**: The AWS region in use.
- **JAVA_HOME**: The path to the Java runtime environment.
- **JENKINS_HOME**: The Jenkins home directory.
- **WORKSPACE**: The directory where Jenkins executes the job.
- **BUILD_URL**: The URL for the build in Jenkins.

COMPLETE CI CD PROCESS {DECLARATIVE PIPELINE}

REQUIREMENTS:

- 1 jenkins server
- 1 sonarqube server
- 1 nexus server
- 1 tomcat server

→setting up jenkins server

LAUNCH ONE EC2 AND INSTALL JENKINS

Jenkins pre requisite is java so make sure java is installed first with git for cloning the source code

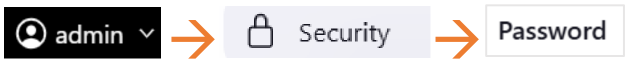
```
yum install git
yum install java-17*
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
yum install fontconfig java-17-openjdk
yum install Jenkins
systemctl start Jenkins
systemctl status Jenkins
```

→**COPY THE INITIAL ADMIN PASSWORD**

```
cat /var/lib/Jenkins/secrets/initialadminpassword
```

→**OPEN BROWSER AND ACCESS THE JENKINS SERVER (JENKINS WORKS ON PORT 8080)**

→LOGIN AND SECURE YOUR JENKINS ACCOUNT BY CREATING ADMIN USER PASSWORD



→setting up sonarqube server

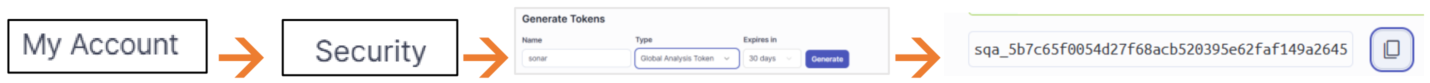
LAUNCH ONE EC2 AND INSTALL SONARQUBE

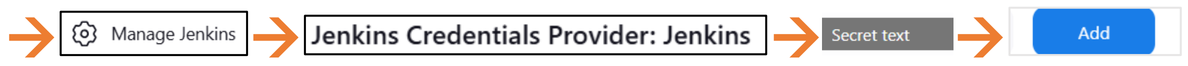
Prerequisites

1. EC2 instance with Java installed (t2.large with at least 20GB storage).
2. PostgreSQL database server.

```
yum install -y docker
cd
system start docker
systemctl start docker
docker pull sonarqube
docker images
docker run -itd -p 9003:9000 sonarqube
```

→ACCESS FROM BROWSER http://<EC2_PUBLIC_IP>:9003 <change password>





→setting up nexus server

LAUNCH ONE EC2 AND INSTALL SONARQUBE

Prerequisites

- EC2 instance with Java installed

```
yum install java-1.8*
cd /opt
wget https://sonatype-download.global.ssl.fastly.net/nexus/3/nexus-3.0.2-02-unix.tar.gz
tar -zxvf  nexus-3.0.2-02-unix.tar.gz
mv /opt/nexus-3.0.2-02 /opt/nexus
sudo adduser nexus
visudo \\ nexus   ALL=(ALL)       NOPASSWD: ALL
sudo chown -R nexus:nexus /opt/nexus
vi /opt/nexus/bin/nexus.rc
sudo ln -s /opt/nexus/bin/nexus /etc/init.d/nexus
su – nexus
service nexus start
```

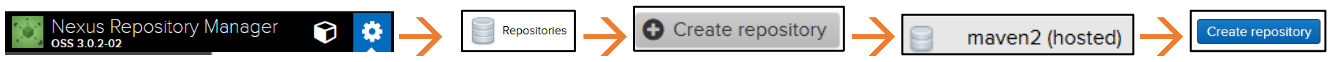
<default username is admin and password is admin123>

 Access Nexus: Open a web browser and navigate to http://<your_instance_ip>:8081. Use the default credentials to log in:

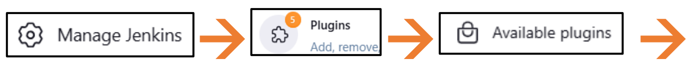
- Username: admin
- Password: Found in the admin.password file located in /opt/sonatype-work/nexus3/admin.password.

 Change Admin Password: Once logged in, change the admin password to something more secure.

→CREATE NEXUS REPOSITORY FOR MAVEN war PACKAGE TO STORE



→INSTALL PLUGINS IN JENKINS GUI



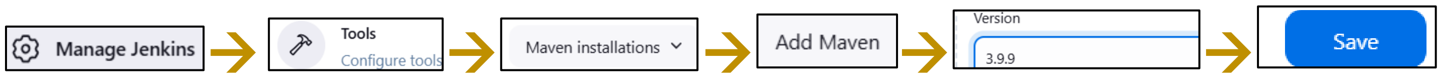
Required Plugins

→Git →SonarQube Scanner →Slack Notification Plugin
→Maven Integration Plugin →Nexus Artifact Uploader
→ssh plugin

CI CD PROCESS

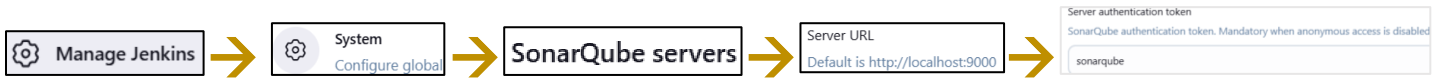
1. Git Clone: Clones the repository from the feature-1.1 branch.
2. Sonarqube Integration: Runs the SonarQube scanner to analyze the code quality.
3. Maven Compilation: Compiles the project using Maven.
4. Nexus Artifactory: Deploys the built artifacts to Nexus.
5. Slack Notification: Sends a notification to a Slack channel.
6. Deploy on Tomcat: Copies the .war file to the Tomcat server.

1.Configure Maven Installation in Jenkins



Version
3.9.9

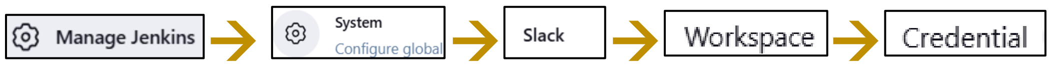
2.Configure SonarQube Server in Jenkins



Server URL
Default is http://localhost:9000

Server authentication token
SonarQube authentication token. Mandatory when anonymous access is disabled.
sonarqube

3.Set Up Slack Integration in Jenkins



4.Configure Nexus Repository in Jenkins

→COPY THE NEXUS URL

5. Set Up Jenkins Declarative Pipeline Job

- Go to Manage Jenkins > Configure System.
- Scroll down to the Publish over SSH section and click Add.
- Configure your SSH server details:

Name: TomcatServer (or any name you prefer).

Hostname: The IP address or hostname of your Tomcat server.

Username: The SSH user.

Remote Directory: (Optional) Base directory on the remote machine.

Private Key: Choose the method of authentication. You can either use an existing private key from Jenkins, enter the key directly, or reference a known host file.

Add Jenkinsfile to Your Repository:

- Create a file named Jenkinsfile in your repository.
- Add the declarative pipeline in it

```
pipeline {
  agent any
  environment {
    REPO_URL = 'https://github.com/betawins/hiring-app.git'
    BRANCH = 'feature-1.1'
    SONARQUBE_SERVER = 'SonarQubeServer'
    MAVEN_TOOL = 'Maven'
    NEXUS_URL = 'http://nexus.example.com/repository/maven-releases/'
    SLACK_CHANNEL = '#ci-cd-notifications'
    TOMCAT_SERVER = 'TomcatServer'
  }
  stages {
    stage('Clone') {
      steps {
        git branch: BRANCH, url: REPO_URL
      }
    }
    stage('SonarQube') {
      steps {
        withSonarQubeEnv(SONARQUBE_SERVER) {
          sh "sonar-scanner -Dsonar.projectKey=hiring-app -Dsonar.sources=."
        }
      }
    }
    stage('Build') {
      steps {
        script {
          def mvnHome = tool MAVEN_TOOL
          sh "${mvnHome}/bin/mvn clean install"
        }
      }
    }
    stage('Nexus') {
      steps {
        script {
          def mvnHome = tool MAVEN_TOOL
          sh """
          ${mvnHome}/bin/mvn deploy:deploy-file -Durl=${NEXUS_URL} -DrepositoryId=nexus-releases -Dfile=target/hiring-
app.war -DgroupId=com.example -DartifactId=hiring-app -Dversion=1.0.0 -Dpackaging=war
          """
        }
      }
    }
```

```
        stage('Notify') {
            steps {
                slackSend channel: SLACK_CHANNEL, message: "Build and Deploy for ${BRANCH} branch is successful."
            }
        }
        stage('Deploy') {
            steps {
                sshPublisher(
                    publishers: [sshPublisherDesc(
                        configName: TOMCAT_SERVER,
                        transfers: [sshTransfer(
                            sourceFiles: 'target/hiring-app.war',
                            remoteDirectory: '/path/to/tomcat/webapps',
                            removePrefix: 'target',
                            execCommand: 'sudo systemctl restart tomcat'
                        )],
                        verbose: true
                    )]
                )
            }
        }
    }
    post {
        always {
            cleanWs()
        }
        success {
            slackSend channel: SLACK_CHANNEL, message: "Pipeline completed successfully!"
        }
        failure {
            slackSend channel: SLACK_CHANNEL, message: "Pipeline failed."
        }
    }
}
```

COMMON ERRORS FACED IN JENKINS

1. OutOfMemoryError

Error: Jenkins displays OutOfMemoryError in logs. **Cause**: Jenkins is running out of heap memory. **Solution**:

- Increase the heap size by modifying the JENKINS_JAVA_OPTIONS in your Jenkins configuration file (jenkins.xml on Windows or /etc/default/jenkins on Linux):

```
sh
JENKINS_JAVA_OPTIONS="-Xmx2g"
```

- Restart Jenkins after updating the configuration.

2. Jenkins Not Starting After Upgrade

Error: Jenkins fails to start after an upgrade. **Cause**: Incompatible plugins or corrupted installation. **Solution**:

- Review jenkins.log for specific error messages.
- Start Jenkins in safe mode (http://your_jenkins_url/safeRestart) to troubleshoot plugin issues.
- Roll back to a previous Jenkins version if necessary.

3. Failed to Connect to Git Repository

Error: Jenkins cannot connect to the Git repository. **Cause**: Incorrect credentials, network issues, or repository settings. **Solution**:

- Verify Git repository URL and credentials.
- Check network connectivity.
- Update Jenkins credentials for the repository.

4. No Space Left on Device

Error: Jenkins shows No space left on device error. **Cause**: Insufficient disk space. **Solution**:

- Check disk space on the server where Jenkins is installed using commands like df -h.
- Clean up old builds and unnecessary files to free up space.

5. SSL Certificate Issues

Error: SSL certificate validation errors. **Cause**: Invalid or expired SSL certificate. **Solution**:

- Update or replace the SSL certificate.
- Configure Jenkins to trust the certificate authority (CA).