TASK 2-DevOps

Question: 1. What are things you need to set, if you want download dependency from private repository ?

Solution: To download dependencies from a private repository, you need to set up authentication and configure the repository properly.

Steps to Access a Private Repository:

1. Use Authentication (Username/Password, Personal Access Token, or SSH key)
2. Configure Repository URL in the package manager (Git, Maven, or others)
3. Store Credentials Securely (Environment variables, .netrc, or settings file)

Examples:

For Git (Using HTTPS with Token)

git clone https://TOKEN@github.com/user/private-repo.git

For Git (Using SSH Key)

git clone git@github.com:user/private-repo.git

For Maven (Private Repository Configuration)

1. Add repository in pom.xml:

```
<repositories>
  <repository>
    <id>private-repo</id>
    <url>https://repo.example.com/maven-private/</url>
  </repository>
</repositories>
```

2. Set authentication in settings.xml:

```
<servers>
  <server>
    <id>private-repo</id>
    <username>your-username</username>
    <password>your-password</password>
  </server>
</servers>
```

Question: 2. What are the common issues faced while working on Maven projects?

1. Java Path/OpenJDK Not Detected in Jenkins

- Issue: Jenkins does not detect the Java path or OpenJDK.
- Solution:

  ```
  export JAVA_HOME=/usr/lib/jvm/java-11-openjdk
  export PATH=$JAVA_HOME/bin:$PATH
  ```

  Add these to ~/.bash_profile or ~/.bashrc, then run source ~/.bash_profile.

2. Confusion While Setting Environment Variables in .bash_profile

- Issue: Incorrectly setting environment variables, leading to build failures.
- Solution: Use echo $VARIABLE_NAME to verify values after adding them to .bash_profile.

  ```
  echo 'export MAVEN_HOME=/opt/maven' >> ~/.bash_profile
  echo 'export PATH=$MAVEN_HOME/bin:$PATH' >> ~/.bash_profile
  source ~/.bash_profile
  ```

3. Tomcat Website Not Working Sometimes

- Issue: The Tomcat server does not start or the deployed application does not respond.
- Solution:
  - Ensure Tomcat is running with ps aux | grep tomcat
  - Check logs in $CATALINA_HOME/logs/catalina.out
  - Restart Tomcat: systemctl restart tomcat

4. Problems Downloading Dependencies Due to Version Issues

- Issue: Maven fails to download dependencies because of version mismatches.
- Solution:
  - Use mvn dependency:tree to check dependencies.
  - Force update dependencies using:

    ```
    mvn clean install-U
    ```

  - Manually specify compatible versions in pom.xml.

5. Dockerfile Did Not Trigger for Tomcat Web Deployment

- Issue: The Dockerfile does not execute properly during deployment.
- Solution:
    - Check Docker logs with docker logs <container_id>.
    - Ensure the correct working directory and port mapping in the Dockerfile:

      ```
      FROM tomcat:latest
      COPY target/*.war /usr/local/tomcat/webapps/
      EXPOSE 8080
      CMD ["catalina.sh", "run"]
      ```

Question: 3. Command to skip the test cases in maven

The command to skip test cases in Maven is:

mvn clean install-DskipTests

or

mvn clean install-Dmaven.test.skip=true

Explanation:

- -DskipTests > Skips the test execution but still compiles the test code.
- -Dmaven.test.skip=true > Completely skips test execution and compilation.

.

Question: 4 How to set Jenkins build to fail based specific word in console output ?

Freestyle Job:

1. Install **"Text Finder Plugin"** from **Manage Jenkins → Plugin Manager**.
2. Go to **Job Configuration > Post-build Actions**.
3. Add **"Text Finder"**.
4. Enter the word (e.g., ERROR or FAILURE) in the **"Regular expression"** field.
5. Check **"Also search the console output"**.
6. Select **"Unstable if found"** or **"Fail the build if found"**.
7. Save and run the job. If the word appears, the build will fail.

Pipeline Job (Declarative Pipeline):

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                script {
                    def log = sh(script: 'your_build_command_here', returnStdout: true)
                    if (log.contains('ERROR')) {
                        error("Build failed due to ERROR in console output!")
                    }
                }
            }
        }
    }
}
```

NOTE : Question: 5 Question: 4 Both are same so i skipped the 5th qn

Question: 6 What are Active and Reactive Parameters (Dynamic Parameterization) in Jenkins

Active and Reactive Parameters are part of **Dynamic Parameterization** in Jenkins, primarily used with the **Active Choices Plugin** to create dynamic input fields in Jenkins jobs.

1. Active Parameter:

- These parameters dynamically populate values based on **predefined logic or scripts**.
- The values are fetched from external sources like databases, APIs, or files.

Example: Dynamically fetching branch names from Git:

return ['main', 'develop', 'feature-branch']

When the job runs, we can see a dropdown with these dynamically fetched branch names.

2. Reactive Parameter:

- These parameters change based on the selection of another parameter.
- They are useful when one parameter depends on another dynamically.

Example: Selecting a **project name** (Active Parameter) dynamically updates the **environment list** (Reactive Parameter).

If the user selects **"Project A"**, the environment dropdown will show devA, testA, prodA. If they select **"Project B"**, it will show devB, testB, prodB.

How to Use in Jenkins?

1. Install **Active Choices Plugin**.
2. Go to **Job Configuration >Add Parameter > Active Choices Parameter**.
3. Use Groovy/Script to dynamically generate values.
4. Add **Reactive Choices Parameter** and link it to another parameter.

This helps in **dynamic form generation** for better automation in Jenkins

Question: 7 How to Customize the Build Number Display in Jenkins?

By default, Jenkins displays the **build number (e.g., #1, #2, #3)** on the job page. You can customize it to show meaningful names like **version numbers, timestamps, or commit IDs** using the **"Build Name Setter"** plugin.

Method 1: Using "Build Name Setter" Plugin

1. Install **Build Name Setter Plugin** from **Manage Jenkins > Plugin Manager**.
2. Go to **Job Configuration >Build Environment**.
3. Enable **"Set Build Name"**.
4. Use variables like:
    - **${BUILD_NUMBER}** >Default build number
    - **${GIT_BRANCH}-${BUILD_NUMBER}** > Show branch & build number
    - **${GIT_COMMIT}** > Show commit ID
    - **Version-${BUILD_NUMBER}** > Custom naming

eg: Release-${BUILD_NUMBER}

Method 2: Using Pipeline Script

```
pipeline {
   agent any
   stages {
      stage('Set Build Name') {
         steps {
            script {
               currentBuild.displayName = "Release-${BUILD_NUMBER}"
            }
         }
      }
   }
}
```

Question: 8. What is a Multibranch Pipeline in Jenkins?

A **Multibranch Pipeline** in Jenkins automatically creates and manages pipelines for each branch in a repository.

Key Features:

- Detects new branches automatically.
- Runs separate pipelines for each branch.
- Useful for CI/CD workflows with feature branches.
- Eliminates manual job creation for new branches.

Example:

- A repository has main, develop, and feature-branch branches.
- A **Multibranch Pipeline** automatically creates jobs for each branch based on its Jenkinsfile.

Steps to Create a Multibranch Pipeline:

1. Go to **Jenkins Dashboard >New Item >Multibranch Pipeline**.
2. Configure the **Git/Bitbucket/GitHub repository URL**.
3. Define the **Jenkinsfile path**.
4. Jenkins will automatically detect and create pipelines for each branch.

Question: 9. What is a Shared Library in Jenkins?

A **Shared Library** in Jenkins allows you to reuse common pipeline code across multiple Jenkinsfiles, making pipeline management more efficient.

Why Use a Shared Library?

- Avoid code duplication across multiple Jenkins pipelines.
- Store reusable functions for CI/CD.
- Maintain centralized pipeline logic.

How to Use a Shared Library?

1. **Create a Git Repository** to store the shared library (e.g., jenkins-shared-library).
2. Inside the repo, create a vars/ folder and add Groovy scripts:

   ```
   // vars/deploy.groovy
   def call() {
      echo "Deploying application..."
   }
   ```

3. Configure Jenkins to use the shared library (**Manage Jenkins > Configure System > Global Pipeline Libraries**).
4. Use it in a Jenkinsfile:

   ```
   @Library('jenkins-shared-library') _
   deploy()
   ```