To set up a CI/CD pipeline for a Java application using Jenkins, I followed a structured approach. First, I created three separate Jenkins jobs: one for compiling the source code (mvn compile), another for testing the code (mvn test), and a third for packaging the code (mvn package). These jobs were configured to execute their respective Maven commands. Next, I set up a CI/CD pipeline that orchestrates the execution of these jobs in sequence. This pipeline ensures that the code is compiled, tested, and packaged automatically upon changes in the source repository. Additionally, I configured a master-slave node setup within Jenkins to distribute the workload across multiple nodes, optimizing build times and resource usage. The master node orchestrates the tasks, while the slave nodes perform the actual build steps.

Commands for Job Configuration:
1. Compiling Job:

shell
Copy code
mvn compile
Jenkins Job Configuration:

Name: Compile Job
Build Step: Execute shell
Command: mvn compile
2. Testing Job:

shell
Copy code
mvn test
Jenkins Job Configuration:

Name: Test Job
Build Step: Execute shell
Command: mvn test
3. Packaging Job:

Shell
Copy ABC Technologies
mvn package
Jenkins Job Configuration:

Name: Package Job
Build Step: Execute shell
Command: mvn package
CI/CD Pipeline Setup:
Jenkinsfile:

groovy
Copy code

```
pipeline {
    agent any
    stages {
        stage('Compile') {
            steps {
```

```
            build job: 'Compile Job'
        }
    }
    stage('Test') {
        steps {
            build job: 'Test Job'
        }
    }
    stage('Package') {
        steps {
            build job: 'Package Job'
        }
    }
  }
}
```

Master-Slave Node Configuration:

Navigate to Manage Jenkins -> Manage Nodes and Clouds.
Add a new node (slave) and configure its properties (e.g., remote root directory, labels, etc.).
Ensure the master node delegates tasks to the slave nodes based on the labels and job configurations.