# jenkins class -04 realtime

---

## jenkins class -04 realtime

🔹 BIG PICTURE (What this PDF is about)

This PDF explains **how Jenkins CI/CD works for a Java application**
—from **developer code push** → **build & test** → **WAR file** → **deploy on Tomcat**.

---

## 🔹 REAL-TIME FLOW (Very Simple)

### 👨‍💻 Step 1: Developer side

- Developers write code (Java / Frontend)
- They **push code to GitHub / Bitbucket**

`Developer → GitHub`

👉 This is where **your work starts as DevOps**

---

### 🧱 Step 2: Jenkins (CI Server)

- Jenkins **pulls code from GitHub**
- Jenkins **builds + tests the code**

For Java:

`mvn clean package`

What happens:

- ✅ Code compiles
- ✅ Tests run (Surefire)
- ✅ WAR file is created

`GitHub → Jenkins (Build + Test → WAR)`

**📌 Important reality**

- If **WAR is generated** → build SUCCESS
- If **tests fail** → build FAILS (no deploy)

---

### 📦 Step 3: WAR file (Artifact)

- WAR file is created inside:

`target/*.war`

👉 WAR = deployable Java application
👉 This is what moves between environments (UAT → PROD)

---

## 🔹 Frontend vs Backend (Very Important in Real Projects)

### 🌐 Frontend (HTML, CSS, JS, React)

Deployed on:

- **NGINX**
- **Apache**

Why?

- Only static files
- Very fast
- No business logic

`Frontend → NGINX`

---

### ⚙️ Backend (Java, Spring, Python, .NET)

Deployed on:

- **Tomcat**
- **JBoss**
- **GlassFish**

Why?

- Executes business logic
- Connects DB
- Handles APIs

`Backend → Tomcat`

---

## 🔹 Web Server vs Application Server (Simple Reality)

| Web Server (NGINX) | App Server (Tomcat) |
|---|---|
| Serves UI | Runs backend code |
| No logic | Business logic |
| Very fast | Slightly slower |
| Static content | Dynamic processing |

👉 **NGINX never runs Java code**
👉 **Tomcat never serves only static UI**

---

## 🔹 Jenkins Job – What you ACTUALLY do

### 1️⃣ Install Jenkins

- Using script
- Make script executable:

`chmod +x install_jenkins.sh`

---

### 2️⃣ Create Jenkins Job

- Select **Git repository**
- Branch: `main` / `master`
- Jenkins clones code to:

`/var/lib/jenkins/workspace/<job-name>`

---

### 3️⃣ Build using Maven

Install:

- Java (JDK 8 / 11)
- Maven

Build step:

`clean package`

(No need to type `mvn` inside Jenkins)

---

### 4️⃣ Result after build

Inside workspace:

`pom.xmlsrc/target/app.war ✅`

This means:

- Build + Test done successfully

---

## 🔹 Deploy to Tomcat (Real Deployment)

### Tomcat setup

- Install Java first
- Download Tomcat
- Start Tomcat on port `8080`

Access:

`http://server-ip:8080`

---

### Tomcat Manager Access (Common Real Issue)

Problem:

`403 Access Denied`

Solution:

1. Edit `context.xml` → remove restriction
2. Add user in `tomcat-users.xml`

```
<user username="tomcat" password="admin@123" roles="manager-gui,manager-script"/>
```

3. Restart Tomcat

---

## 🔹 Jenkins → Tomcat Auto Deploy (CI/CD)

Install Jenkins plugin:

`Deploy to container`

Post-build action:

- WAR path:

`target/*.war`

- Context path:

`swiggy`

- Tomcat URL:

`http://ip:8080`

Result:

`Build + Test + Deploy ✅`

---

## 🔹 Final REAL-TIME CI/CD FLOW (Easy to Remember)

`Developer ↓GitHub ↓Jenkins (Build + Test → WAR) ↓Tomcat (Application Running)`

---

## 🧠 One-Line Interview / Exam Answer

> Jenkins pulls code from GitHub, builds and tests it using Maven, generates a WAR file, and deploys it automatically to Tomcat using a CI/CD pipeline.

---

If you want next:

- 🔹 Same explanation for **Node.js + React + PM2**
- 🔹 Jenkins **pipeline (Jenkinsfile) version**
- 🔹 **Exam-ready short notes**
- 🔹 **Real production mistakes & fixes**

Just tell me 👍

yes yes please

Great 👍
I'll give you **both**, in the **same simple + real-time way**:

**1️⃣ Node.js + React + PM2 CI/CD (like your real projects)**
**2️⃣ Jenkins Pipeline (Jenkinsfile) – exam + production ready**

No confusion, no theory overload.

---

# 🔹 1️⃣ Node.js + React + PM2 – Real-Time CI/CD Flow

## 🧠 Key difference from Java

- ❌ No WAR file

- ❌ No Tomcat
- ✅ Build artifacts = `build/` (React) or app files (Node)
- ✅ PM2 runs the app

---

## ✅ REAL PROJECT FLOW (Simple)

```
Developer ↓GitHub ↓Jenkins (Build + Test) ↓Server (PM2 + NGINX)
```

---

### 🔹 React Application Flow

#### Jenkins does:

```
npm installnpm run build
```

Output:

```
build/ ├── index.html ├── static/
```

#### Deployment:

- Copy `build/` to:

```
/var/www/react-app/
```

#### NGINX serves UI:

```
http://domain.com
```

👉 **NGINX = frontend hosting**

---

### 🔹 Node.js Backend Flow

#### Jenkins does:

```
npm installnpm test (optional)
```

#### Deployment:

- Code copied to server
- Start with PM2:

```
pm2 start index.js --name backend-apipm2 save
```

👉 **PM2 keeps app alive (restart, crash recovery)**

---

### 🔹 NGINX + Node (Real Production Setup)

`User Request ↓NGINX (80/443) ↓Node.js (PM2, port 3000)`

NGINX is **reverse proxy**, not backend runner.

---

## 🔹 2️⃣ Jenkins Pipeline (Jenkinsfile) – SIMPLE & REAL

### 🧩 Java Project (WAR → Tomcat)

```
pipeline {
  agent any

  stages {
    stage('Checkout') {
      steps {
        git 'https://github.com/org/java-app.git'
      }
    }

    stage('Build & Test') {
      steps {
        sh 'mvn clean package'
      }
    }

    stage('Deploy') {
      steps {
        sh 'cp target/*.war /opt/tomcat9/webapps/'
      }
    }
  }
}
```

✔ Build
✔ Test
✔ Deploy

---

### 🧩 React Project Jenkinsfile

```
pipeline {
  agent any

  stages {
    stage('Checkout') {
      steps {
        git 'https://github.com/org/react-app.git'
      }
    }

    stage('Build') {
      steps {
        sh '''
          npm install
          npm run build
        '''
      }
    }

    stage('Deploy') {
      steps {
        sh 'cp -r build/* /var/www/react-app/'
      }
    }
  }
}
```

---

## 🧩 Node.js + PM2 Jenkinsfile

```
pipeline {
  agent any

  stages {
    stage('Checkout') {
      steps {
        git 'https://github.com/org/node-app.git'
      }
    }

    stage('Install') {
      steps {
        sh 'npm install'
      }
    }

    stage('Restart App') {
      steps {
        sh '''
          pm2 restart backend-api || pm2 start index.js --name backend-api
          pm2 save
        '''
      }
    }
  }
}
```

---

## 🔹 Java vs Node vs React (Exam-Ready Table)

| Tech | Build Tool | Artifact | Server |
|---|---|---|---|
| Java | Maven | WAR | Tomcat |
| React | npm | build/ | NGINX |
| Node.js | npm | JS files | PM2 |
| Python | pip | Source | Gunicorn |

---

## 🧠 One-Line Interview Answers

### Jenkins

> Jenkins is a CI/CD tool that automates build, test, and deployment.

### PM2

> PM2 is a process manager that keeps Node.js applications running in production.

### NGINX

> NGINX is a web server and reverse proxy used to serve frontend and route traffic.

### WAR

> WAR is a deployable Java artifact used in application servers like Tomcat.

---

## 🔥 REAL DEVOPS TIP (Important)

In **modern companies**:

- ❌ WAR + Tomcat → slowly decreasing
- ✅ Node + React + Docker + Kubernetes → increasing
- PM2 still widely used in **VM-based setups**

# ✅ Jenkins Class PDF – Real-Time Explanation (Simple)

This PDF explains **how Java applications are built and deployed in real companies using Jenkins**.

---

## 1️⃣ Developer → GitHub (REAL LIFE)

### What happens in real projects

- Developers write **Java code**
- They push code to **GitHub / Bitbucket**

📌 Example:

`Swiggy backend team pushes new order-service code to GitHub`

👉 DevOps **does NOT write code**, DevOps **automates what happens next**

---

## 2️⃣ Jenkins (CI Server) – What Jenkins ACTUALLY DOES

### Real-time scenario

- Jenkins job is already created
- Jenkins **clones code from GitHub**

📍 Jenkins workspace:

`/var/lib/jenkins/workspace/order-service/`

Inside workspace:

`pom.xmlsrc/`

👉 This confirms **code is successfully pulled**

---

## 3️⃣ Build + Test using Maven (MOST IMPORTANT)

### Command used in real companies

`mvn clean package`

### What happens internally

1. **Compile code**
2. **Run unit tests (Surefire)**
3. **Generate WAR file**

📍 Output:

`target/order-service.war`

✅ WAR file = build success
❌ No WAR = build failed

📌 Real example:

> If tests fail, Jenkins STOPS → deployment never happens

---

## 4️⃣ WAR File – Why it is Important (REAL USE)

### What is WAR?

- WAR = Java application package
- This is what moves to **UAT / PROD**

📌 Real life:

`Same WAR file tested in UAT is deployed to PROD`

👉 No rebuilding in PROD (best practice)

---

## 5️⃣ Web Server vs Application Server (REAL DIFFERENCE)

### Frontend (UI)

- HTML, CSS, JS
- Deployed on **NGINX**

📌 Example:

`Swiggy UI → NGINX`

---

### Backend (Java code)

- Business logic
- DB calls
- APIs

Runs on **Tomcat**

📌 Example:

`Order creation → Java API → Tomcat`

👉 **NGINX cannot run Java**

👉 **Tomcat executes Java logic**

---

## 6️⃣ Jenkins → Tomcat Deployment (REAL FLOW)

### Production setup

- Jenkins server (CI)
- Tomcat server (PROD)

### Jenkins uses plugin:

`Deploy to container`

Post-build action:

`WAR: target/*.warContext path: swiggy`

📌 Real effect:

`http://prod-ip:8080/swiggy`

Application is LIVE 🚀

---

## 7️⃣ Tomcat Installation – Why These Steps Matter

### Why Java first?

Tomcat depends on Java.

📌 Real issue:

> If Java not installed → Tomcat will NOT start

---

### Why context.xml edit?

Tomcat blocks manager access by default (security)

📌 Real fix:

- Remove restriction lines
- Add user in `tomcat-users.xml`

Without this:

`403 Access Denied`

---

## 8️⃣ Jenkins + Tomcat = FULL CI/CD (REAL COMPANY FLOW)

### What happens after every code push

`Developer pushes code↓Jenkins auto build + test↓WAR generated↓WAR deployed to Tomcat↓Application updated`

👉 This is **continuous deployment**

---

## 9️⃣ Console Logs & Workspace (REAL DEBUGGING)

### Where DevOps checks issues

- Jenkins console output
- Jenkins workspace
- Tomcat logs

📌 Example:

`/opt/tomcat9/logs/catalina.out`

This is how **production issues are debugged**

---

## 🔑 ONE-LINE REAL-TIME SUMMARY (VERY IMPORTANT)

> In real projects, Jenkins pulls Java code from GitHub, builds and tests it using Maven, generates a WAR file, and deploys it automatically to a Tomcat application server.