1. When you push to GitHub → Jenkins auto-triggers
2. Jenkins pulls the repo
3. Builds your Spring Boot app into a JAR
4. (Later) can include unit tests, version documentation, and local auto-deploy

---

1 Prerequisites

- Jenkins installed (local or server)
- Maven installed on Jenkins server (Jenkins → Manage Jenkins → Global Tool Configuration → Maven installations)
- Java 17+ installed on Jenkins server
- Git installed on Jenkins server
- Your GitHub repo is public or Jenkins has credentials for it
- Webhook configured in GitHub so commits trigger Jenkins

---

2 Set up GitHub → Jenkins Auto Trigger

Alright — here's the step-by-step ngrok method for Windows (both CMD & PowerShell) so GitHub can talk to your locally running Jenkins.

---

1 Download and Install ngrok

1. Go to: https://ngrok.com/download
2. Download the Windows zip.
3. Extract `ngrok.exe` to a folder (e.g., `C:\ngrok`).
4. (Optional but convenient) Add `C:\ngrok` to your PATH so you can run `ngrok` from anywhere.

---

2 Connect ngrok to Your Account

1. Sign up for a free ngrok account (needed for stable tunnels).

2. From the ngrok dashboard, copy your Auth Token.

3. In CMD or PowerShell:

   ```
   ngrok config add-authtoken <YOUR_AUTH_TOKEN>
   ```

---

3 Start Jenkins Locally

- Make sure Jenkins is running:

  ```
  http://localhost:8080
  ```

---

4 Start ngrok Tunnel

CMD:

```
cd C:\ngrok
ngrok http 8080
```

PowerShell:

```
Set-Location C:\ngrok
.\ngrok.exe http 8080
```

---

5 Get the Public URL

- ngrok will display something like:

  ```
  Forwarding     https://abc123.ngrok-free.app -> http://localhost:8080
  ```

- Copy the https URL — this is now your public Jenkins address.

---

6 Set the GitHub Webhook

1. Go to your GitHub repo → Settings → Webhooks → Add webhook.
2. Payload URL:

   ```
   https://abc123.ngrok-free.app/github-webhook/
   ```

3. Content type: `application/json`
4. Select: "Just the push event"
5. Save.

---

7 Configure Jenkins Job

- In your pipeline job:
  - Build Triggers → ✓ `GitHub hook trigger for GITScm polling`

- Save job.

---

8 Test

1. Commit & push to your repo.
2. In GitHub → Webhooks, you should see green ticks for deliveries.
3. Jenkins should start building instantly.

💡 Tip: ngrok URLs change each time you restart ngrok unless you have a paid plan (static domains). For free tier, you'll need to update the GitHub webhook URL when you restart ngrok.

————

Got it — since ngrok URLs change every time you restart it (free plan), the trick is to not hardcode the Jenkins URL in your Jenkinsfile at all. Instead, we let GitHub trigger Jenkins via the webhook, and the Jenkinsfile will just define the

Why this works

- The Jenkinsfile does not care what the incoming webhook URL is.
- Jenkins listens on whatever ngrok forwards, and GitHub sends events there.
- When the ngrok URL changes, you only update the webhook in GitHub, not the Jenkinsfile.

Minimal URL-free Jenkinsfile

```
pipeline {
    agent any

    triggers {
        // Jenkins will run whenever GitHub webhook hits /github-webhook/
        githubPush()
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/YOUR_USERNAME/YOUR_REPO.git'
            }
        }
        stage('Build Jar') {
            steps {
                sh './mvnw clean package -DskipTests'
            }
        }
        stage('Unit Tests') {
            steps {
                sh './mvnw test'
            }
        }
        stage('Generate Version Document') {
            steps {
                script {
                    def version = sh(returnStdout: true, script: "./mvnw help:evaluate -Dexp
                    writeFile file: 'version.txt', text: "Build version: ${version}"
                }
            }
        }
        stage('Local Deploy') {
            steps {
                sh 'java -jar target/*.jar &'
            }
        }
    }
}
```

How to make ngrok changes painless

Right now, if ngrok URL changes, you only need to:

1. Start ngrok:

   ```
   ngrok http 8080
   ```

2. Copy the new `https://abc123.ngrok-free.app`
3. Update GitHub → Repo → Settings → Webhooks → Replace old URL → Save.

💡 Pro Tip: If you want to skip even that manual webhook update step, you can:

- Use localtunnel or Cloudflare Tunnel (free, with static domain)
- Or, use a paid ngrok plan with a fixed subdomain.

---

3 Create Jenkins Pipeline Job

1. New Item → Name: `spring-boot-ci-pipeline`
2. Select Pipeline
3. In "Build Triggers", check: ✅ GitHub hook trigger for GITScm polling
4. In "Pipeline", select Pipeline script from SCM
   - SCM: Git
   - Repo URL: your GitHub repo
   - Branch: `*/main`
   - Script Path: `Jenkinsfile` (we'll create this next)

---

4 Create `Jenkinsfile` in Your Repo

Here's a starter Jenkinsfile that builds your JAR and is ready to extend with tests, docs, and deploy steps later.

```
pipeline {
    agent any

    tools {
        maven 'Maven_3.9.11'   // Name from Jenkins "Global Tool Configuration"
        jdk 'Java_17'          // Name from Jenkins "Global Tool Configuration"
    }

    triggers {
        githubPush()
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', url: 'https://github.com/YOUR_USERNAME/YOUR_REPO.git'
            }
        }

        stage('Build JAR') {
            steps {
                sh 'mvn clean package -DskipTests'
            }
        }

        stage('Unit Tests') {
            steps {
                sh 'mvn test'
            }
        }

        stage('Version Document') {
            steps {
                sh '''
                echo "Build Version: $(mvn help:evaluate -Dexpression=project.version -q -Df
                echo "Build Time: $(date)" >> version.txt
                '''
                archiveArtifacts artifacts: 'version.txt', followSymlinks: false
            }
        }

        stage('Local Deploy') {
            steps {
                sh 'java -jar target/*.jar --server.port=9090 &'
            }
        }
    }

    post {
        success {
            echo "Build and Deploy completed successfully!"
        }
        failure {
            echo "Build failed!"
        }
    }
}
```

---

5 How This Works

- Trigger → Webhook calls Jenkins on each Git commit
- Checkout → Pulls latest code
- Build JAR → Creates `target/*.jar`
- Unit Tests → Runs your test suite
- Version Document → Generates a `version.txt` file with app version & timestamp
- Local Deploy → Runs the JAR locally on `:9090` (change as needed)
- Artifacts → Keeps `version.txt` in Jenkins build history

---

6 Next Steps

- Add `mvn verify` or `mvn test` for more robust testing
- Add code coverage reporting (JaCoCo)
- Configure separate jobs for staging/prod deployment
- Secure Jenkins with credentials for production deployments

If you want, I can now give you an extended Jenkinsfile that:

- Creates the JAR
- Runs unit tests with coverage report
- Generates HTML version documentation
- Deploys locally and stops previous instance before starting the new one (so you avoid port conflicts)

That would make this a complete auto-deploy loop for your Spring Boot app.