**Complete Jenkins Learning Summary (Steps + Actions)**

**Step 1: Jenkins Installation & Initial Setup**

**Actions you performed:**

1. Ran Jenkins Docker container on Windows:
2. docker run -p 8080:8080 -p 50000:50000 jenkins/jenkins:lts
3. Verified container is running:
4. docker ps
5. Opened Jenkins in browser: http://localhost:8080
6. Retrieved initial admin password from container:
7. docker exec -it <jenkins-container-id> cat /var/jenkins\_home/secrets/initialAdminPassword
8. Entered password in browser to unlock Jenkins.
9. Installed **Suggested Plugins**.
10. Created an **admin user** with username and password.
11. Verified Jenkins Dashboard is accessible.

**Key Takeaways:**

* Jenkins runs as a local server accessible via browser.
* Dashboard is where jobs are created and managed.
* Plugins enable integrations like Git, Docker, etc.

**Step 2: Jenkins Jobs & Freestyle Project**

**Actions you performed:**

1. On Jenkins Dashboard → Clicked **New Item**.
2. Entered job name: First-Freestyle-Job.
3. Selected **Freestyle Project** → Click **OK**.
4. Configured job:
   * General → Optional description: "My first Jenkins job".
   * Build → **Add Build Step** → **Execute Shell**.
   * Entered command:
   * echo "Hello from Jenkins!"
5. Clicked **Save**.
6. Clicked **Build Now** to run the job.
7. Checked **Console Output** → saw "Hello from Jenkins!".

**Key Takeaways:**

* Freestyle jobs are simple GUI-based jobs.
* You can execute shell commands in Jenkins.
* Console Output logs the job’s execution.

**Step 3: Integrating Jenkins with GitHub**

**Actions you performed:**

1. Ensured Git is installed in Jenkins container:
2. docker exec -it <jenkins-container-id> git --version

(Installed if missing using apt-get install -y git)

1. Created GitHub repo: Jenkins-practice-repo.
2. Added simple README.md file:
3. Hello Jenkins from GitHub!
4. Created new Jenkins Freestyle Job: Github-Freestyle-Job.
5. Configured **Source Code Management**:
   * Selected **Git**
   * Repo URL: https://github.com/mishra0010/Jenkins-practice-repo.git
   * Updated branch to \*/main (fixed master/main issue).
6. Build Step → Execute Shell:
7. echo "Code pulled from GitHub repo:"
8. ls -l
9. cat README.md
10. Saved job and clicked **Build Now**.
11. Checked Console Output → verified repo files and README content.

**Key Takeaways:**

* Jenkins can fetch source code from GitHub automatically.
* Always match branch names (main vs master).
* Console Output helps debug SCM integration issues.

**Step 4: Automating Builds with Webhooks (ngrok)**

**Actions you performed:**

1. Installed **ngrok**.
2. Ran ngrok to expose local Jenkins:
3. ngrok http 8080
4. Obtained public forwarding URL: https://<ngrok-id>.ngrok.io
5. Created GitHub Webhook:
   * Settings → Webhooks → Add webhook
   * Payload URL: https://<ngrok-id>.ngrok.io/github-webhook/
   * Content type: application/json
   * Trigger event: **Push only**
6. Configured Jenkins job:
   * Build Triggers → GitHub hook trigger for GITScm polling
7. Tested webhook:
8. echo "Webhook test $(date)" >> webhook.txt
9. git add .
10. git commit -m "Testing Jenkins webhook trigger"
11. git push origin main
12. Verified Jenkins job triggered automatically and executed build.

**Key Takeaways:**

* Webhooks notify Jenkins of GitHub pushes automatically.
* ngrok exposes local Jenkins to the public temporarily.
* Console Output shows automated build triggered by webhook.
* Closing ngrok stops webhook-based builds.

**Step 4.5: ngrok Notes**

* ngrok session is temporary; URL changes every time you run it.
* Manual Jenkins builds still work when ngrok is closed.
* For permanent auto-builds, use a public server or **Poll SCM**.

**✅ Current Status**

* Jenkins installed and running locally.
* Can create Freestyle Jobs and execute builds manually.
* Can pull code from GitHub successfully.
* Can auto-build via webhook (ngrok).
* Understand the **full flow**: GitHub push → Webhook → Jenkins Job → Auto Build

**Step 5: Poll SCM (Automated Builds without ngrok)**

* **What you did:**
  + Configured the **Poll SCM** trigger in Github-Freestyle-Job.
  + Used a **cron expression** (H/2 \* \* \* \*) to check GitHub every 2 minutes.
  + Pushed a commit → Jenkins detected the change → auto-build triggered.
* **What happened behind the scenes:**
  + Jenkins cloned your repo at intervals and checked for new commits.
  + On detecting changes → build was triggered with message *“Started by an SCM change”*.
* **Key Takeaways:**  
  ✔️ Poll SCM works without ngrok or webhook.  
  ✔️ Uses cron syntax to schedule checks.  
  ✔️ Less “instant” than webhooks, but reliable for local setups.

**Step 6: Jenkins Pipeline (as Code)**

* **What you did:**
  + Created a Jenkinsfile in your repo with 3 stages: **Build → Test → Deploy**.
  + Created a **Pipeline Job** in Jenkins → “Pipeline script from SCM”.
  + Ran the job → saw stages executing in **Console Output** & **Blue Ocean UI**.
* **What happened behind the scenes:**
  + Jenkins pulled your repo → located the Jenkinsfile.
  + Parsed the file using **Declarative Pipeline DSL**.
  + Created a **workspace** and executed each stage step-by-step (echo in this case).
* **Key Takeaways:**  
  ✔️ Pipelines are **code-driven** (stored in repo, version-controlled).  
  ✔️ Declarative Pipeline uses structured syntax.  
  ✔️ Pipeline stages = workflow steps (build, test, deploy, etc.).  
  ✔️ This is the foundation of real CI/CD in Jenkins.

**Step 7: Declarative vs Scripted Pipelines**

* **What you did:**
  + Compared two pipeline syntaxes:
    1. **Declarative Pipeline** (structured, easy, what you used first).
    2. **Scripted Pipeline** (Groovy-based, flexible, more code-like).
  + Converted your Jenkinsfile to **Scripted Pipeline** and ran it again.
* **What happened behind the scenes:**
  + Declarative pipeline → Jenkins enforces structure (pipeline { agent any ... }).
  + Scripted pipeline → Jenkins directly runs your Groovy script (node { stage('...') }).
  + Both produced same stages → but handled differently under the hood.
* **Key Takeaways:**  
  ✔️ Declarative Pipeline → beginner-friendly, strict format.  
  ✔️ Scripted Pipeline → advanced, flexible, like writing code.  
  ✔️ Both run in Jenkins’ pipeline engine.  
  ✔️ Most modern teams prefer **Declarative Pipelines** for clarity + maintainability.

✅ **Your Current Status**

* You can now:
  + Run **Freestyle jobs**.
  + Trigger builds manually, via **webhooks**, and via **Poll SCM**.
  + Write **Pipelines** using Jenkinsfile.
  + Understand the difference between **Declarative & Scripted pipelines**.