



Pimpri Chinchwad Education Trust's
Pimpri Chinchwad College of Engineering
(PCCoE)
(An Autonomous Institute)
Affiliated to Savitribai Phule Pune
University(SPPU) ISO 21001:2018 Certified by
TUV



Course: DevOps Laboratory	Code: BIT26VS01
Name: Amar Vaijinath Chavan	PRN: 124B2F001
Assignment 5: Integrate Jenkins with a Git repository .Configure Jenkins to pull code from GitHub.	

Aim: To configure a Jenkins Pipeline that securely pulls source code from a GitHub repository onto a dedicated Worker Node using a Personal Access Token for authentication.

Objectives:

1. To generate and utilize **GitHub Personal Access Tokens (classic)** for secure SCM authentication.
2. To implement a **Declarative Pipeline** to automate the "Checkout" process.
3. To verify successful code synchronization on a remote **Jenkins Worker Node**.

Prerequisites:

- Jenkins Master-Worker architecture.
- GitHub Repository: <express-mysql-app-databoard>(any codebase).
- GitHub Personal Access Token with repo scopes.

Theory:

1. GitHub Personal Access Tokens (PAT)

A PAT serves as a secure alternative to using a standard password for Git authentication. In professional environments, PATs are preferred because they can be scoped to specific permissions (like "repo" for full control of private repositories) and have expiration dates for better security management.

2. Jenkins Declarative Pipeline

Unlike Freestyle jobs, a Pipeline allows the entire build process to be defined as code. Using the pipeline {} block, we can specify which agent (node) should perform the work and define sequential stages like "Checkout" and "Verify."

Practical Procedure / Steps:

Step 1: Generate GitHub PAT

1. Navigate to **GitHub Settings > Developer Settings > Personal access tokens (classic)**.
2. Click **Generate new token**.
3. **Note:** Jenkins-EC2-Key.
4. **Scopes:** Select repo (Full control of private repositories).
5. Click **Generate token** and copy the resulting key (e.g., ghp_...).

The screenshot shows two consecutive screenshots of the GitHub Developer Settings page for generating a new personal access token.

Screenshot 1: New personal access token (classic) creation screen

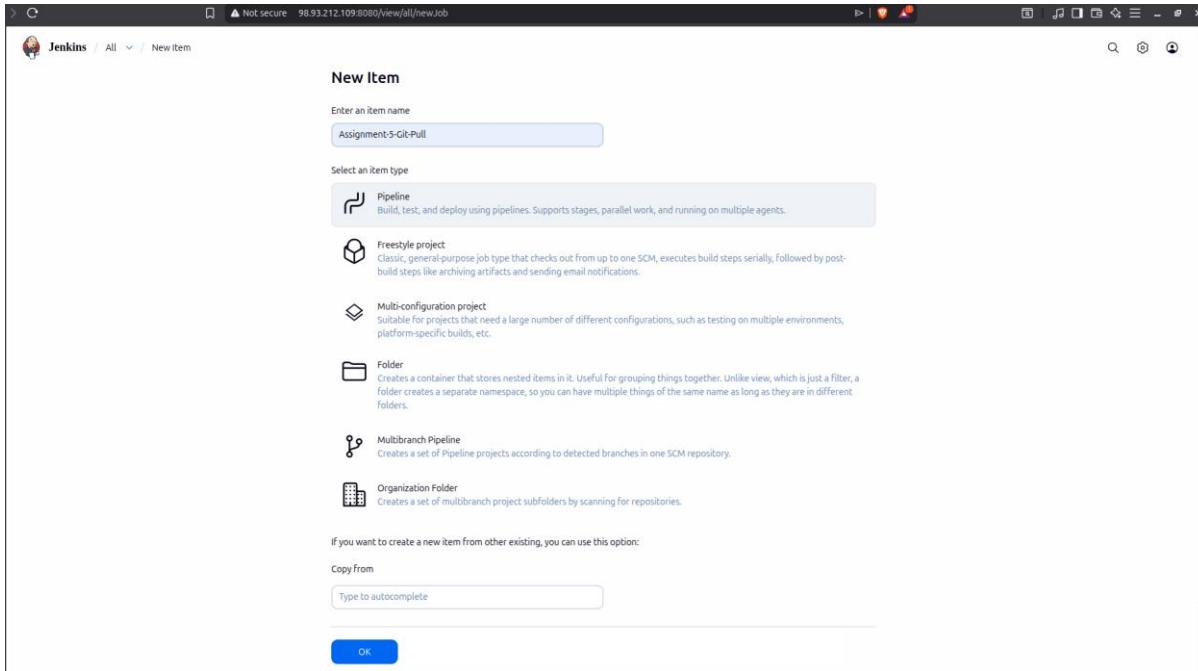
- The left sidebar shows "Personal access tokens" selected under "Tokens (classic)".
- The main area is titled "New personal access token (classic)".
- A note says: "Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication."
- A "Note" field contains "jenkins-EC2-Key".
- An "Expiration" dropdown is set to "30 days (Feb 22, 2026)".
- A "Select scopes" section lists various GitHub permissions:
 - repo**: repo_status, repo_deployment, public_repo, repo_invite, security_events
 - workflow**: Update GitHub Action workflows
 - write:packages**: Upload packages to GitHub Package Registry, Download packages from GitHub Package Registry
 - delete:packages**: Delete packages from GitHub Package Registry
 - admin:org**: Full control of orgs and teams, read and write org projects
 - read:org**: Read org and team membership, read org projects
 - manage:org**: Read org and team membership, read org projects
 - admin:public_key**: Full control of user public keys
 - write:public_key**: Write user public keys
 - read:public_key**: Read user public keys

Screenshot 2: Personal access tokens (classic) list screen

- The left sidebar shows "Personal access tokens" selected under "Tokens (classic)".
- The main area lists generated tokens:
 - A message: "Some of the scopes you've selected are included in other scopes. Only the minimum set of necessary scopes has been saved."
 - A "Personal access tokens (classic)" heading with a "Generate new token" button.
 - A message: "Tokens you have generated that can be used to access the GitHub API." followed by a "Copied" button.
 - A list of tokens:
 - ghp_x5hJBK8uFoshtuqLuccr20lu5ksttG3KTqEH ✓
 - A note at the bottom: "Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication."

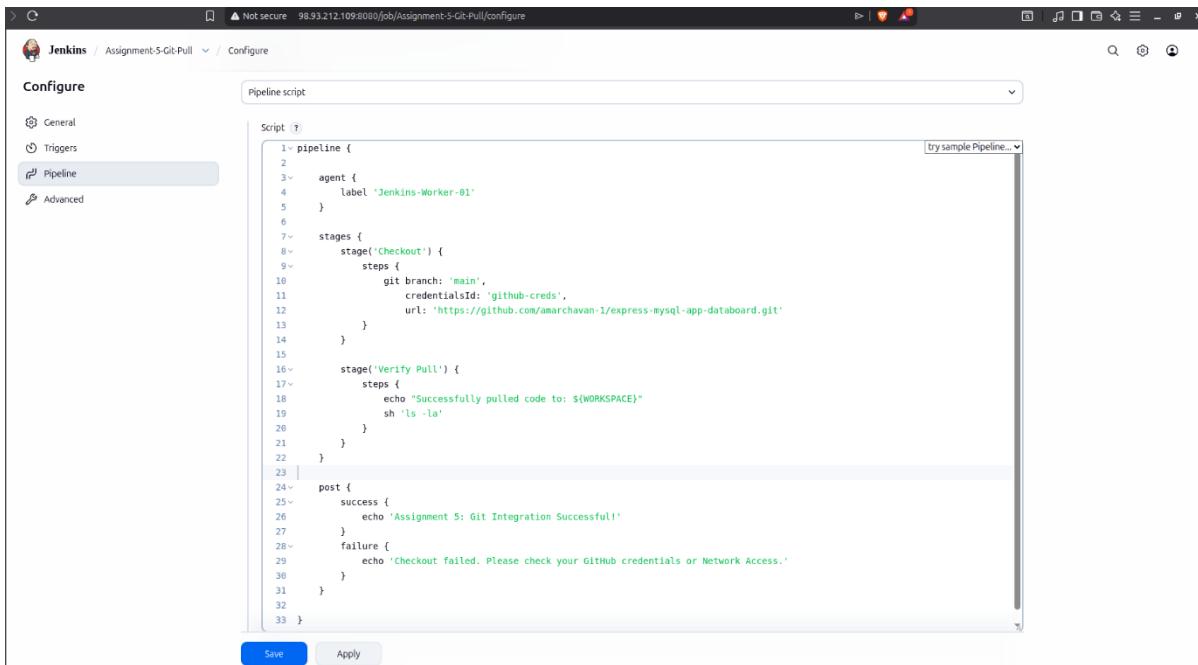
Step 2: Create Jenkins Pipeline Job

1. Open Jenkins Dashboard and select **New Item**.
2. **Item Name:** Assignment-5-Git-Pull.
3. Select **Pipeline** and click **OK**.



Step 3: Configure Pipeline Script

Navigate to the **Pipeline** section and enter the following script to target your worker node:



Step 4: Execute and Verify

1. Click **Build Now**.
2. Open **Console Output** to verify the Git clone status.
3. Access the **Worker Node Terminal** and navigate to `~/workspace/Assignment-5-Git-Pull` to see the physically cloned files (`app.js`, `package.json`, etc.).

The screenshot shows the Jenkins job details for Assignment-5-Git-Pull. It indicates a successful build (#1) from Jan 23, 2026, at 11:29:49 AM. The build was started by user Amar Chavhan. The build duration was 9.3 seconds. The repository URL is https://github.com/amarchavhan-1/express-mysql-app-database.git. The commit hash is d2fb67eeebab0f0d564ecfd0c4b65cbe6d3a88c10. The build log shows no changes.

The screenshot displays two Jenkins windows. The top window shows the Jenkins console output for job #1, which includes a command to pull code from GitHub and lists the files in the workspace. The bottom window shows a terminal session on a worker node (Ubuntu) where the user has navigated to the workspace directory and listed the contents of the Assignment-5-Git-Pull folder, confirming the successful cloning of the repository.

```
> git branch -a -v --no-abbrev # timeout=10
> git checkout -b main d2fb67eeebab0f0d564ecfd0c4b65cbe6d3a88c10 # timeout=10
Commit message: "Update README.md"
First time build. Skipping changelog.
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Verify Pull)
[Pipeline] echo
Successfully pulled code to: /home/ubuntu/workspace/Assignment-5-Git-Pull
[Pipeline] sh
+ ls -la
total 88
drwxr-x 5 ubuntu ubuntu 4096 Jan 23 11:29 .
drwxr-x 4 ubuntu ubuntu 4096 Jan 23 11:29 ..
drwxr-x 8 ubuntu ubuntu 4096 Jan 23 11:29 .git
-rw-rw-r-- 1 ubuntu ubuntu 13 Jan 23 11:29 .gitignore
-rw-rw-r-- 1 ubuntu ubuntu 2345 Jan 23 11:29 README.md
-rw-rw-r-- 1 ubuntu ubuntu 6339 Jan 23 11:29 app.js
-rw-rw-r-- 1 ubuntu ubuntu 46049 Jan 23 11:29 package-lock.json
-rw-rw-r-- 1 ubuntu ubuntu 428 Jan 23 11:29 package.json
drwxr-x 2 ubuntu ubuntu 4096 Jan 23 11:29 public
drwxr-x 3 ubuntu ubuntu 4096 Jan 23 11:29 views
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
Assignment 5: Git Integration Successful!
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
Assignment 5: Git Integration Successful!
[Pipeline]
[Pipeline] // stage
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

```
ubuntu@Jenkins-Worker-01:~/workspace/Assignment-5-Git-Pull$ ls
caches  remoting  remotings.jar  workspace
ubuntu@Jenkins-Worker-01:~/workspace/Assignment-5-Git-Pull$ cd workspace/
ubuntu@Jenkins-Worker-01:~/workspace$ ls
Assignment-5-Git-Pull  Assignment-5-Git-Pull@tmp
ubuntu@Jenkins-Worker-01:~/workspace$ cd Assignment-5-Git-Pull
ubuntu@Jenkins-Worker-01:~/workspace/Assignment-5-Git-Pull$ ls
README.md  app.js  package-lock.json  package.json  public  views
ubuntu@Jenkins-Worker-01:~/workspace/Assignment-5-Git-Pull$ _
```

Conclusion

The completion of this assignment demonstrates the critical integration between Jenkins and GitHub, forming the foundation of a secure Continuous Integration workflow. By utilizing a Declarative Pipeline, we successfully automated the source code retrieval process specifically for a remote Worker Node, ensuring the Master remains optimized for orchestration. The use of GitHub Personal Access Tokens (PAT) provided a secure, industry-standard authentication method for accessing the repository. Verifying the cloned files, such as `app.js` and `package.json`, directly within the EC2 Worker terminal confirmed the seamless synchronization between the cloud-based version control and our local build environment. This setup effectively simulates a production environment where code is automatically staged for subsequent stages of the DevOps lifecycle.