	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.

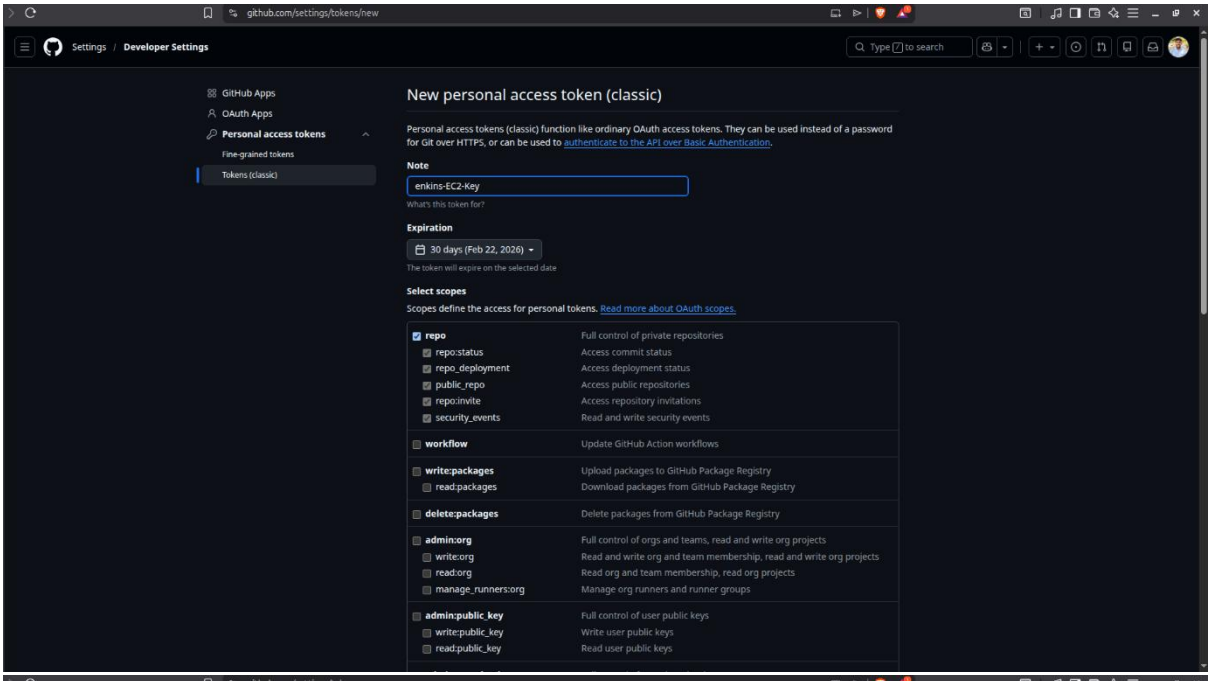
2. Jenkins Declarative Pipeline

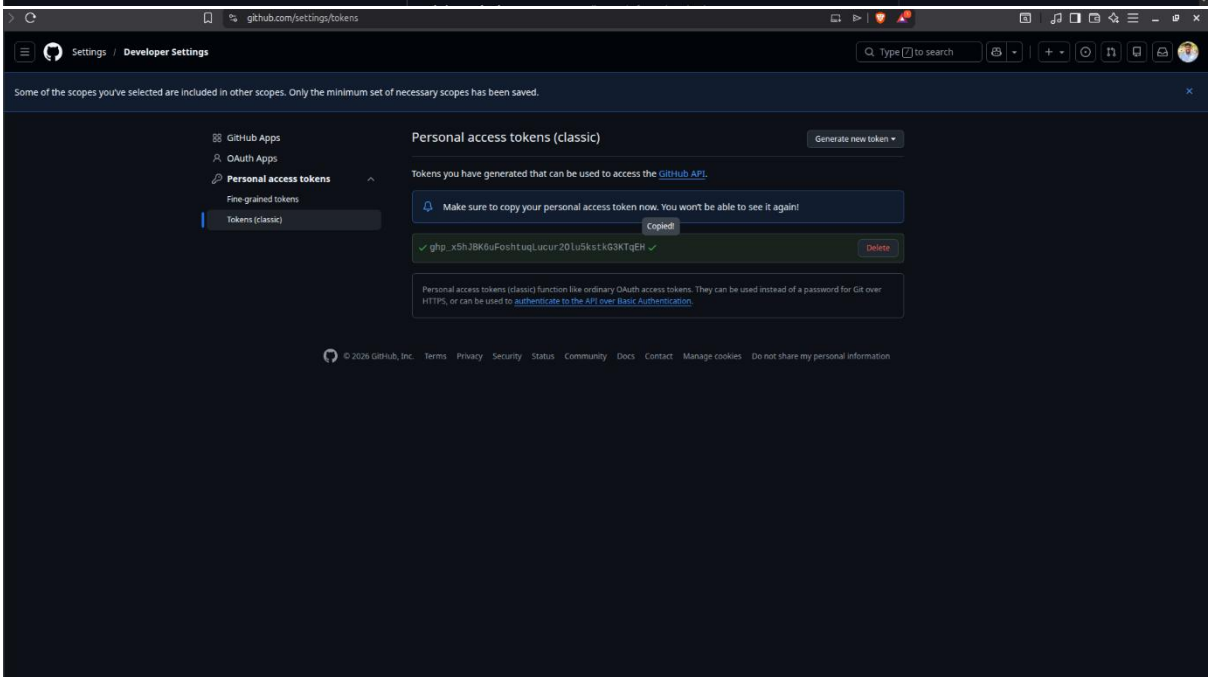
Unlike Freestyle jobs, a Pipeline allows the entire build process to be defined as code. Using the pipeline {} block, we can specify which agent (node) should perform the work and define sequential stages like "Checkout" and "Verify."

Practical Procedure / Steps:

Step 1: Generate GitHub PAT

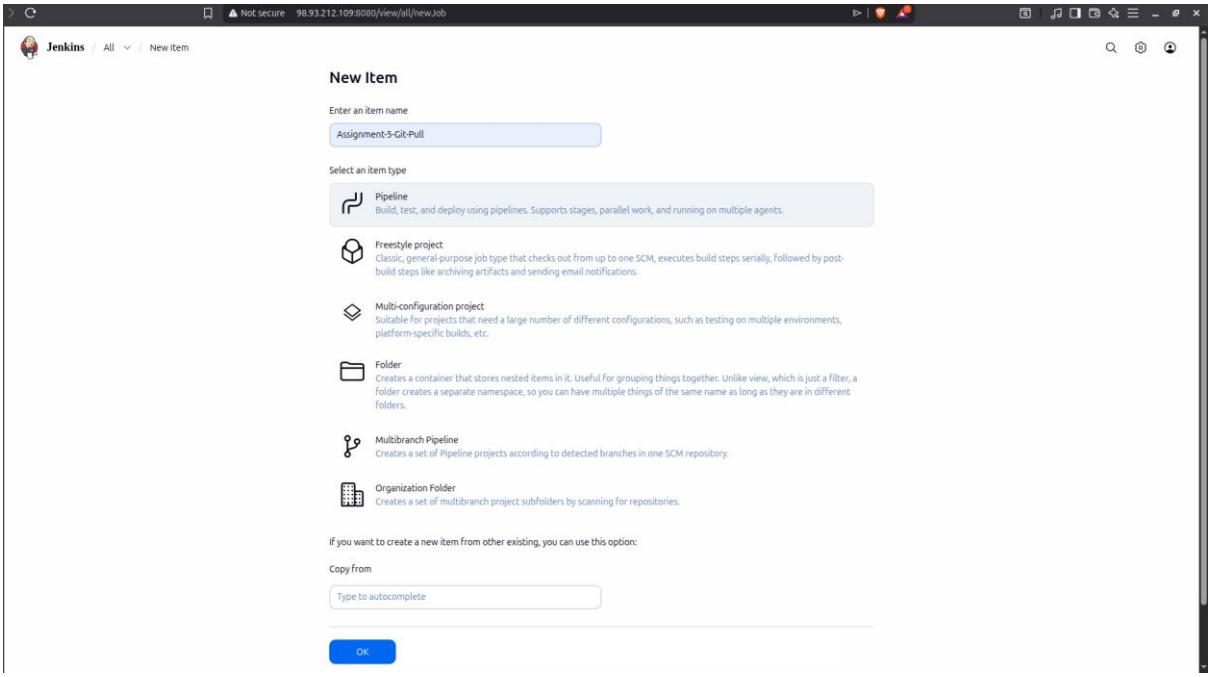
1. Navigate to **GitHub Settings > Developer Settings > Personal access tokens (classic)**.
2. Click **Generate new token**.
3. **Note:** Jenkins-EC2-Key.
4. **Scopes:** Select repo (Full control of private repositories).
5. Click **Generate token** and copy the resulting key (e.g., ghp_...).





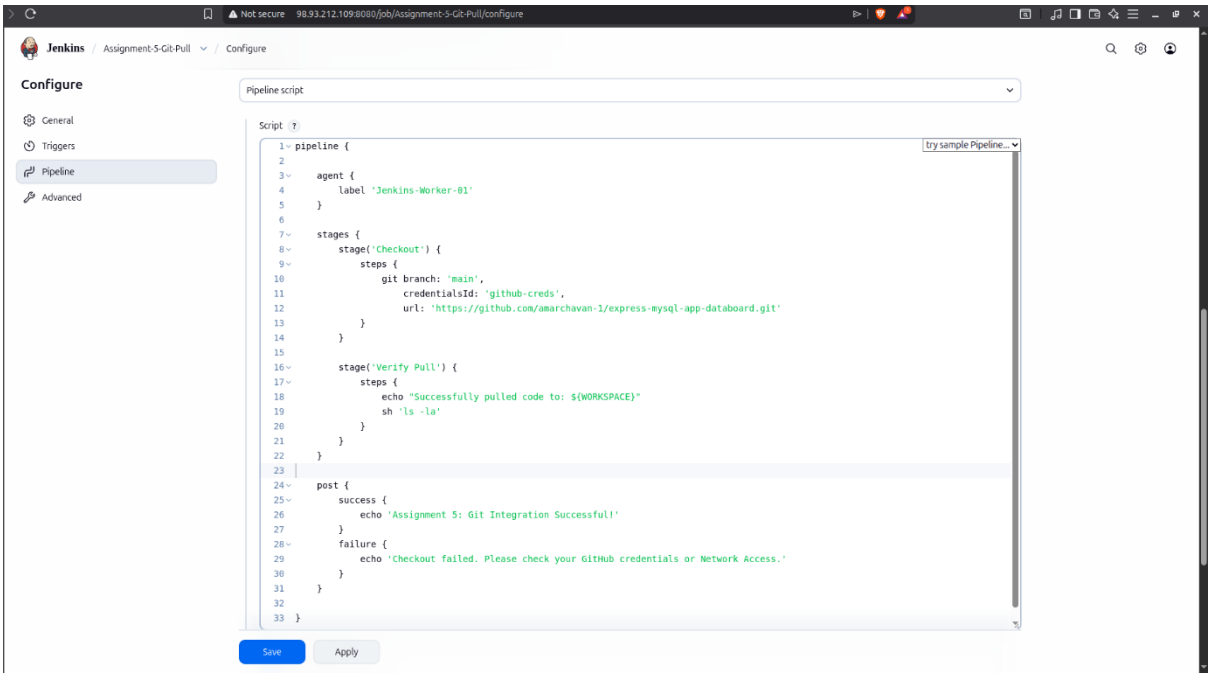
Step 2: Create Jenkins Pipeline Job

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



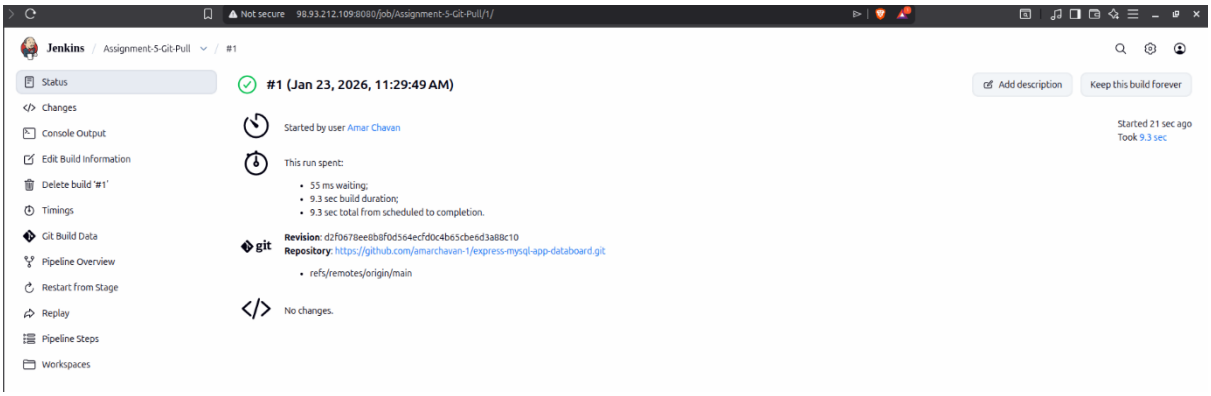
Step 3: Configure Pipeline Script

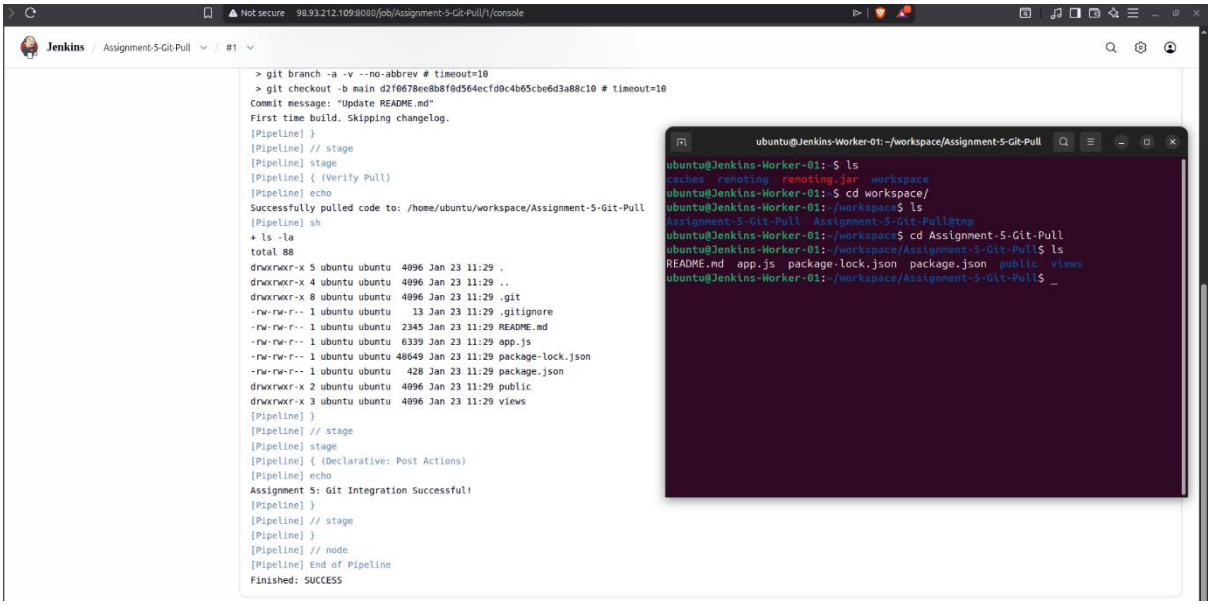
Navigate to the **Pipeline** section and enter the following script to target your worker node:



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.).





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.