

# Introduction to Jenkins

## Module 6: Source Control and Multibranch Pipelines



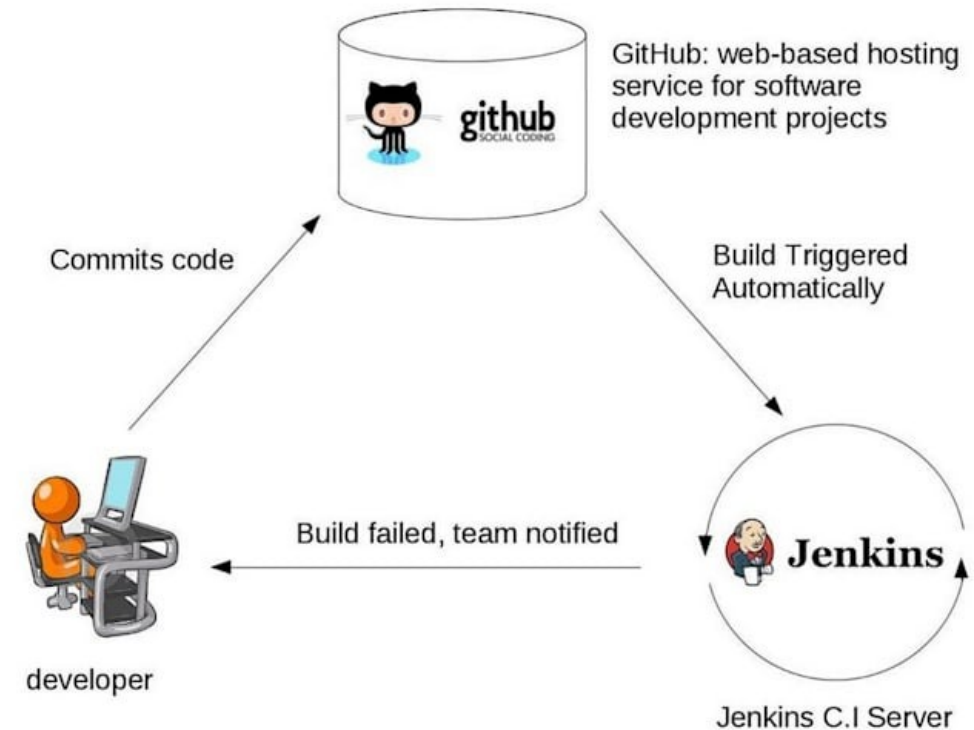
# Topics

- GitHub and GitLab integration
- Webhooks and event-based builds
- Multi-branch Pipeline configuration
- PR validation workflows



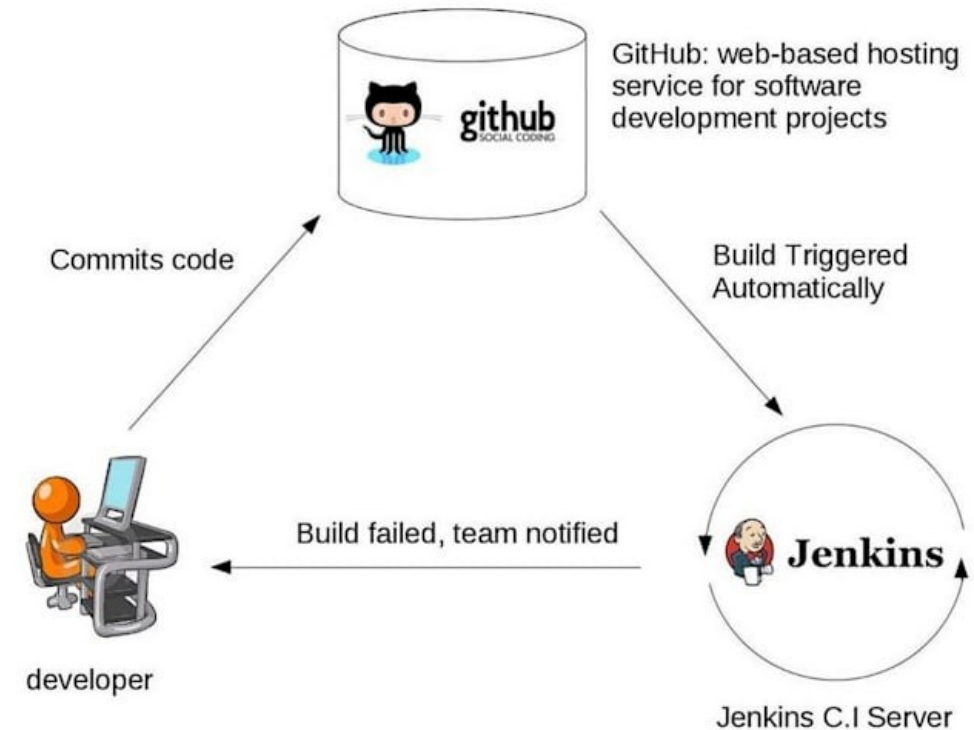
# SCM Integration

- Modern Jenkins is SCM-driven
  - Meaning it responds to events that occur in a code repository
  - Removes the need to manually start builds or Jenkins actions
- Jenkins responds to events like
  - Code changes
  - Branch creation
  - Pull requests / merge requests
  - Tag events



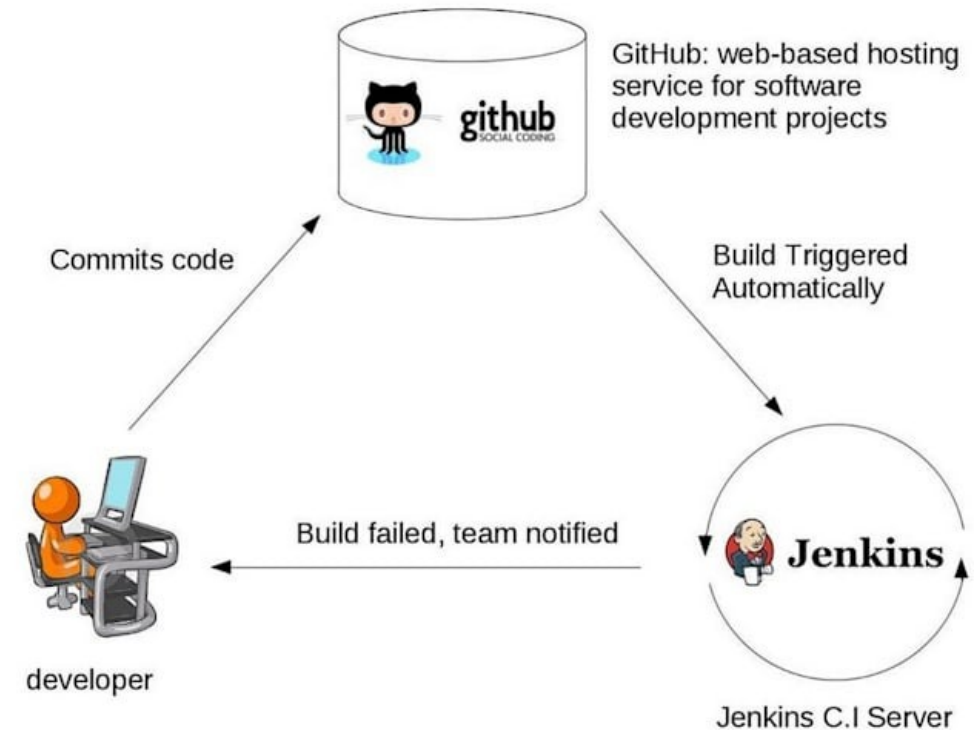
# GitHub and GitLab Integration

- Jenkins integrates with Git hosting platforms to
  - Fetch repositories
  - Discover branches and PRs
  - Trigger builds
  - Report build status back to the repository
- Supported platforms include:
  - GitHub
  - GitLab
  - Bitbucket



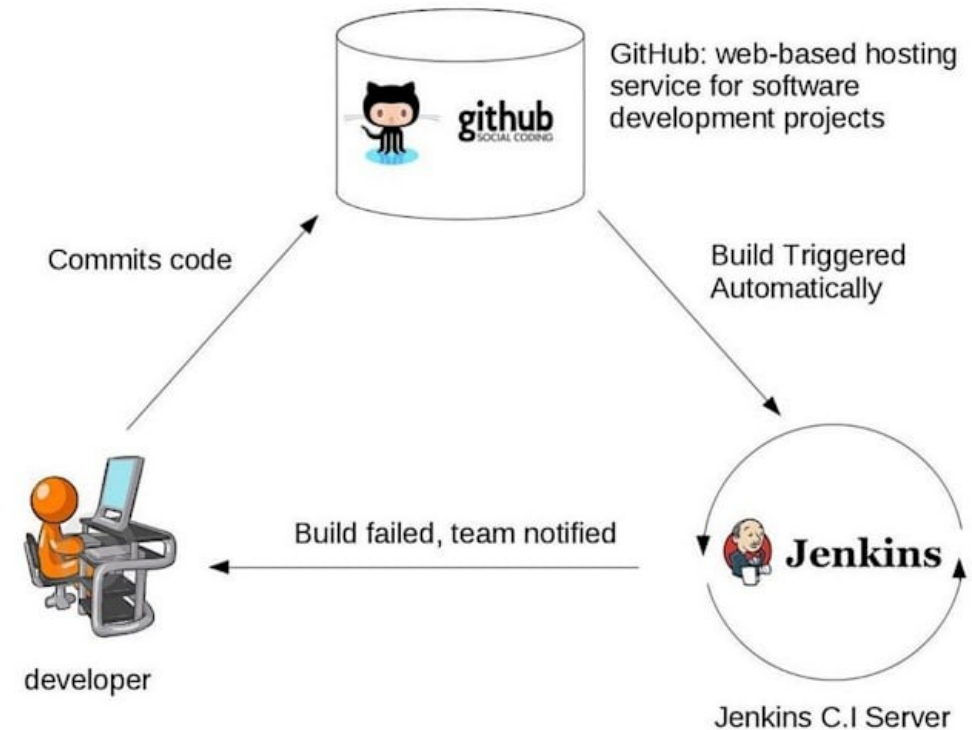
# Authentication

- Access to Git platforms uses standard authentication methods
  - Personal access tokens
  - App-based authentication
  - SSH keys
- Credentials are stored in Jenkins and injected securely
- With proper integration, Jenkins can
  - Discover repositories automatically
  - Detect new branches
  - Build pull requests
  - Update commit status (pass/fail)
  - Comment on PRs



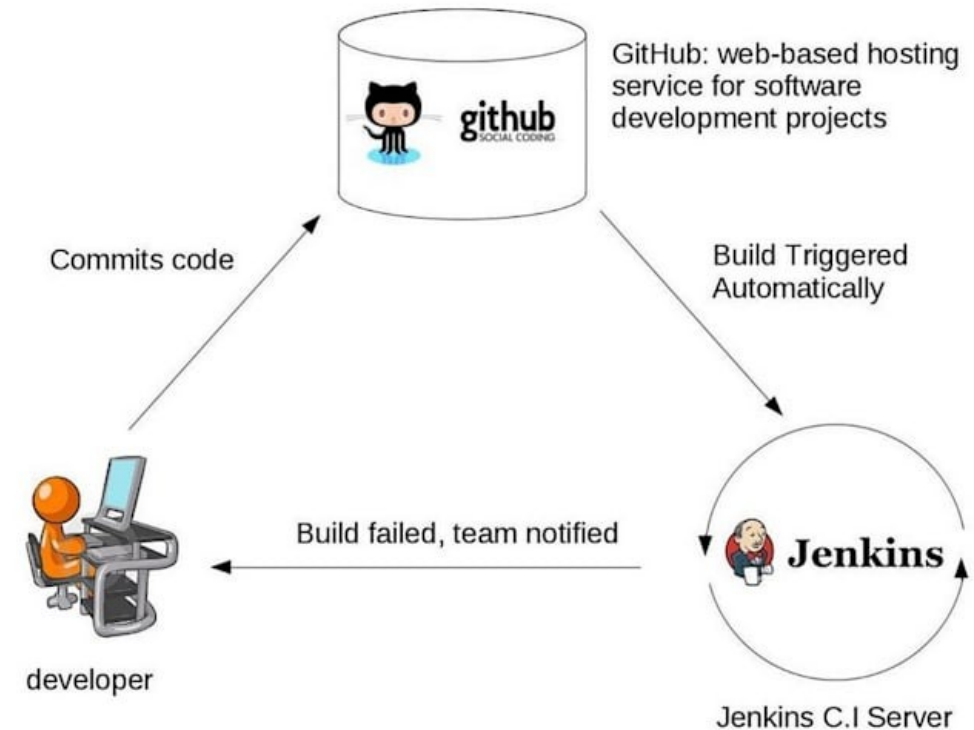
# Discover Repositories

- Jenkins can
  - Connect to a GitHub or GitLab organization or group
  - List available repositories
  - Allow admins to select repos without manually entering URLs
- This is often done using
  - Organization folders
  - Access tokens or app integration
- Sees the source control system as a catalog, not just a file server



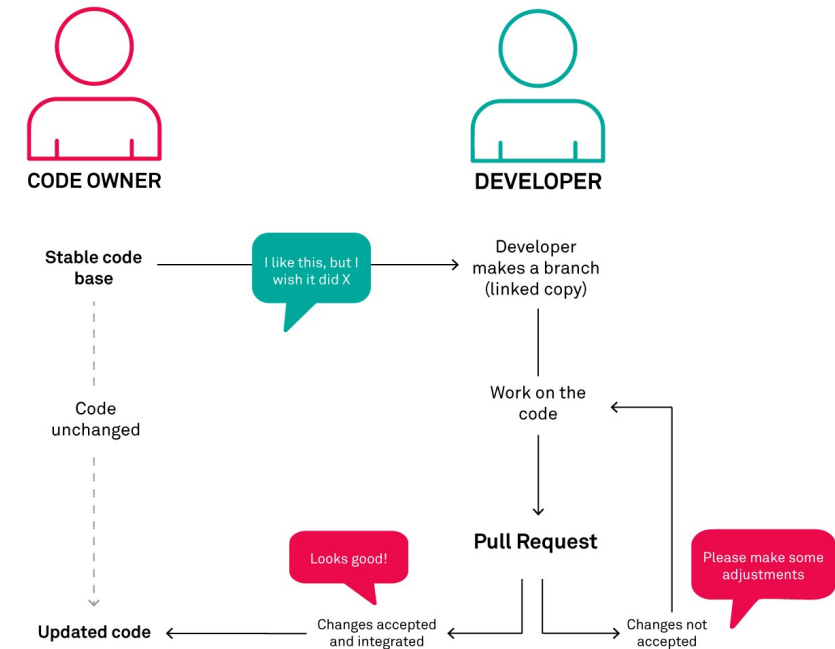
# Detect New Branches

- Jenkins can
  - Detect when a new branch is created
  - Automatically create a pipeline for that branch
  - Use the Jenkinsfile from that branch
  - No manual job creation is required
- Benefits
  - Feature branches are built automatically
  - Teams don't need to ask for Jenkins jobs
  - CI keeps up with modern Git workflows



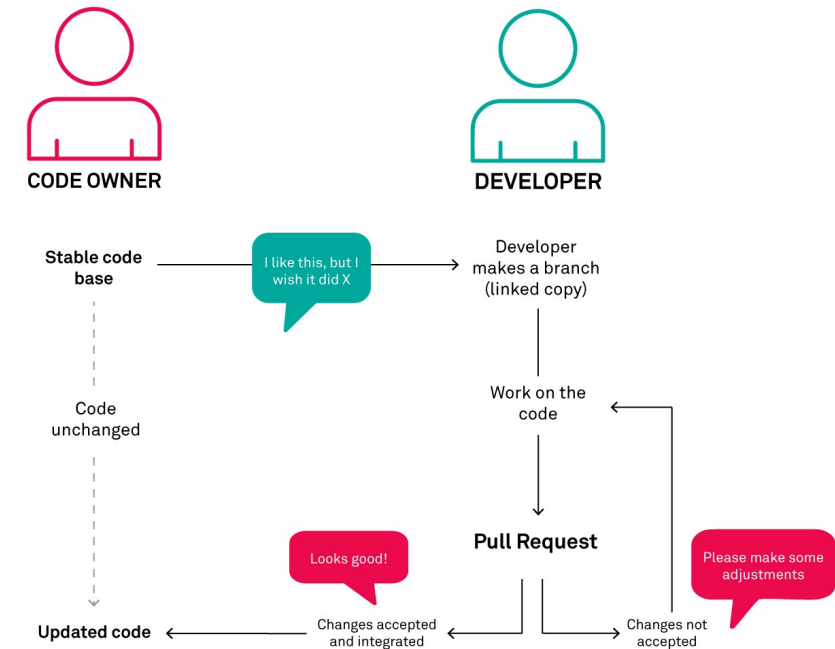
# Pull Request (PR) Validation Workflows

- PR validation ensures that before code is merged
  - It builds successfully
  - All the relevant tests pass
  - All quality gates and standards are enforced
- Typical PR validation flow
  - Developer opens PR
  - Jenkins detects PR event
  - Jenkins runs pipeline
  - Results are reported back to the PR
  - Merge is allowed or blocked depending on the outcome of the pipeline



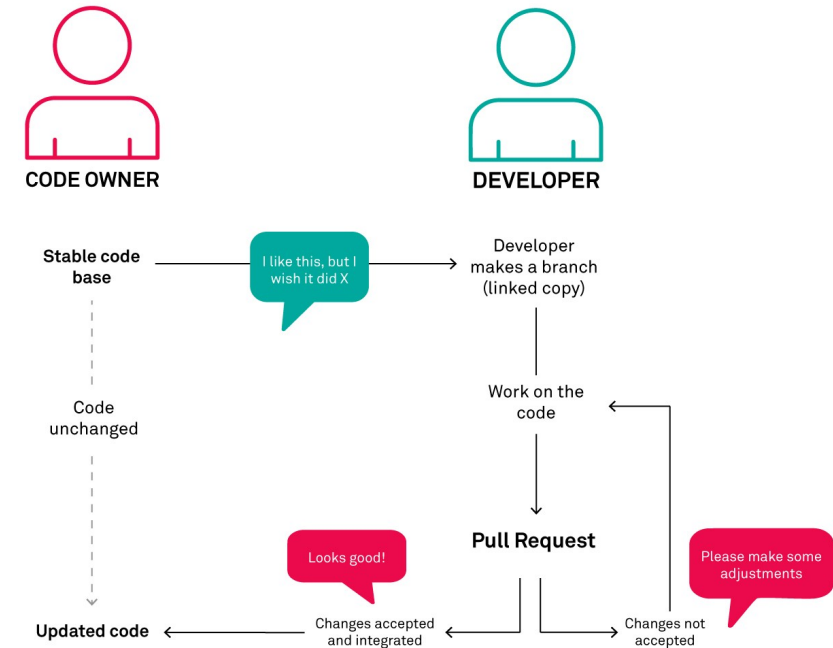
# Pull Request (PR) Validation Workflows

- PR Pipelines usually include
  - Build and compile
  - Unit tests
  - Static analysis
  - Security scans
  - Linting
- Deployments are typically excluded
  - These pipelines are part of the CI process, not the deployment process



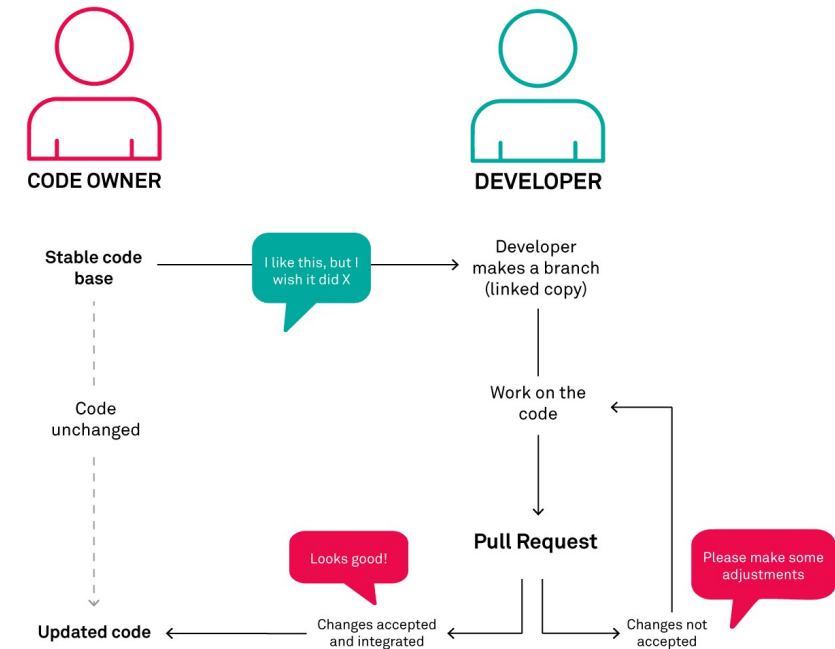
# Build Pull Requests (PRs)

- When a PR or merge request is opened
  - It triggers a pipeline specifically for the PR
  - Tests the proposed changes before merge
  - PR builds typically
    - *Combine the feature branch with the target branch*
    - *Simulates the post-merge state*
  - Benefits
    - *Catches integration issues early*
    - *Prevents broken code from being merged*
    - *Shifts quality checks left*
  - PR builds answer the question
    - *“Will this code work after it’s merged?”*



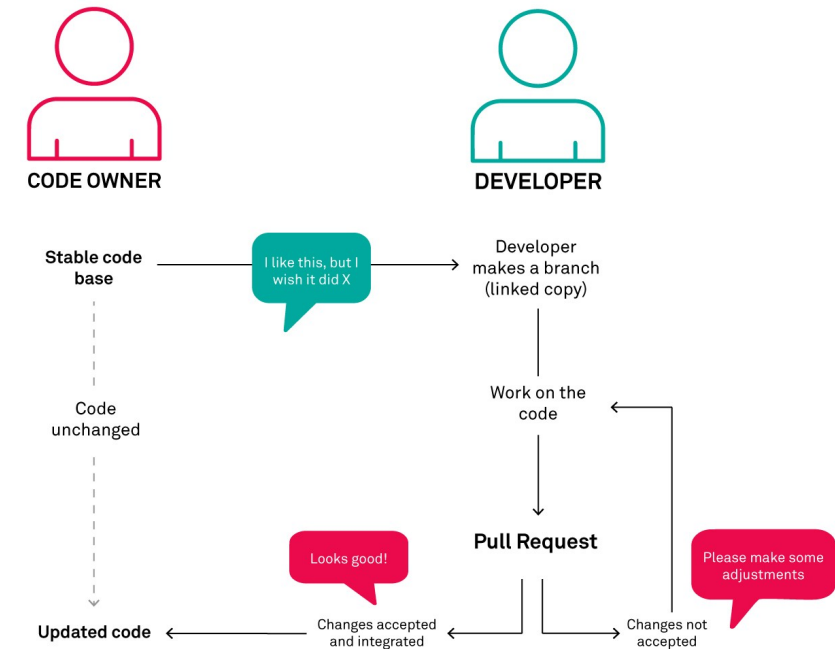
# Update Commit Status (Pass / Fail)

- Jenkins can report build results directly back to GitHub or GitLab by
  - Updating commit status checks
  - Marking builds as:
    - *Pending*
    - *Successful*
    - *Failed*
  - These statuses appear
    - *On commits*
    - *On PRs*
    - *In merge checks*
  - Developers get feedback without leaving GitHub/GitLab
    - *Reviewers see CI results instantly*
    - *Branch protection rules can block merges on failure*
  - Jenkins becomes part of the code review process.



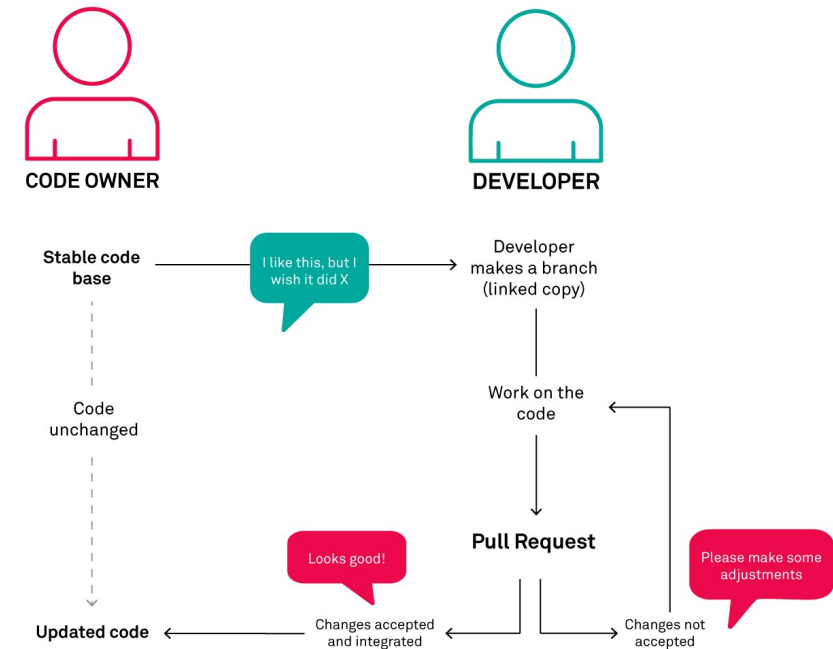
# Comment on Pull Requests

- Jenkins can:
  - Post comments on PRs
  - Include build summaries
  - Link to logs and reports
  - Provide actionable feedback
- Examples:
  - “Build failed: unit tests”
  - “Quality gate failed”
  - “Security scan detected vulnerabilities”
- Benefits
  - Feedback is contextual and visible
  - Developers don’t need to check Jenkins manually
  - Improves collaboration and speed
  - CI feedback should live where developers work



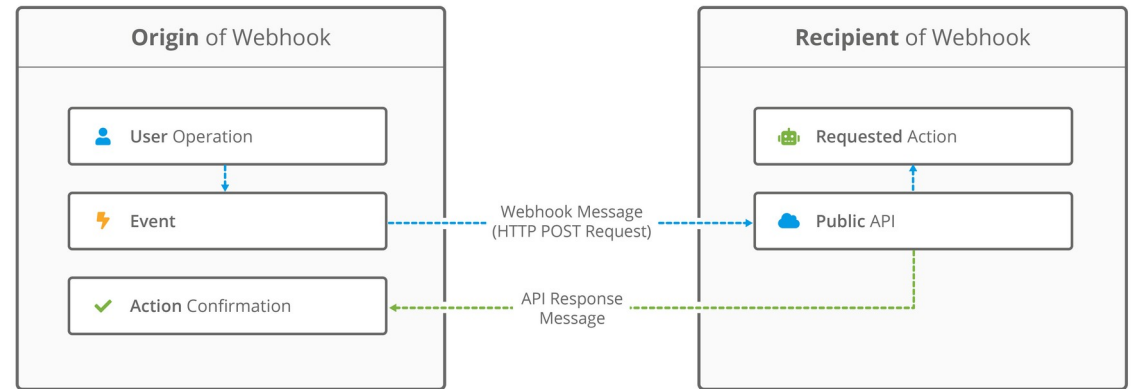
# Typical Integrated Workflow

- Developer pushes code and creates a PR
- GitHub/GitLab sends an event
- Jenkins triggers the correct pipeline
- Pipeline runs tests and checks
- Jenkins reports results back to the PR
- Merge is allowed or blocked automatically



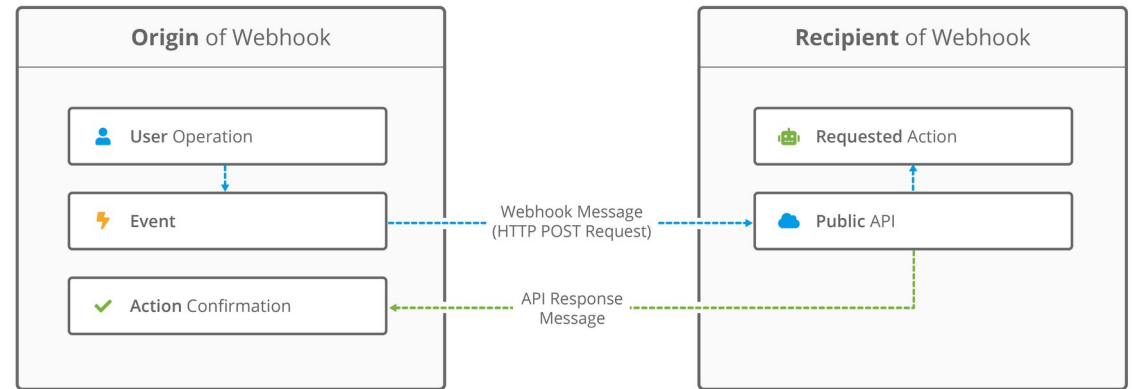
# Webhooks and Event-Based Builds

- A webhook is an HTTP callback sent by GitHub or GitLab when an event occurs.
- Examples of events
  - Push to a branch
  - Pull request opened
  - Pull request updated
  - Merge request created



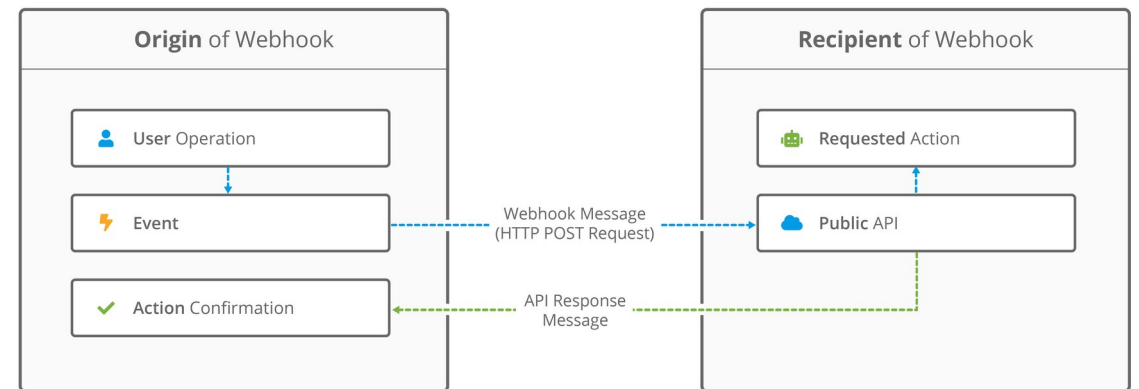
# Webhooks and Event-Based Builds

- Without webhooks
  - Jenkins polls repositories periodically
  - Builds are delayed
  - Unnecessary load is created
- With webhooks
  - Builds trigger immediately
  - Jenkins reacts in near real time
  - CI feedback is faster
- Webhooks turn Jenkins from a poller into a listener



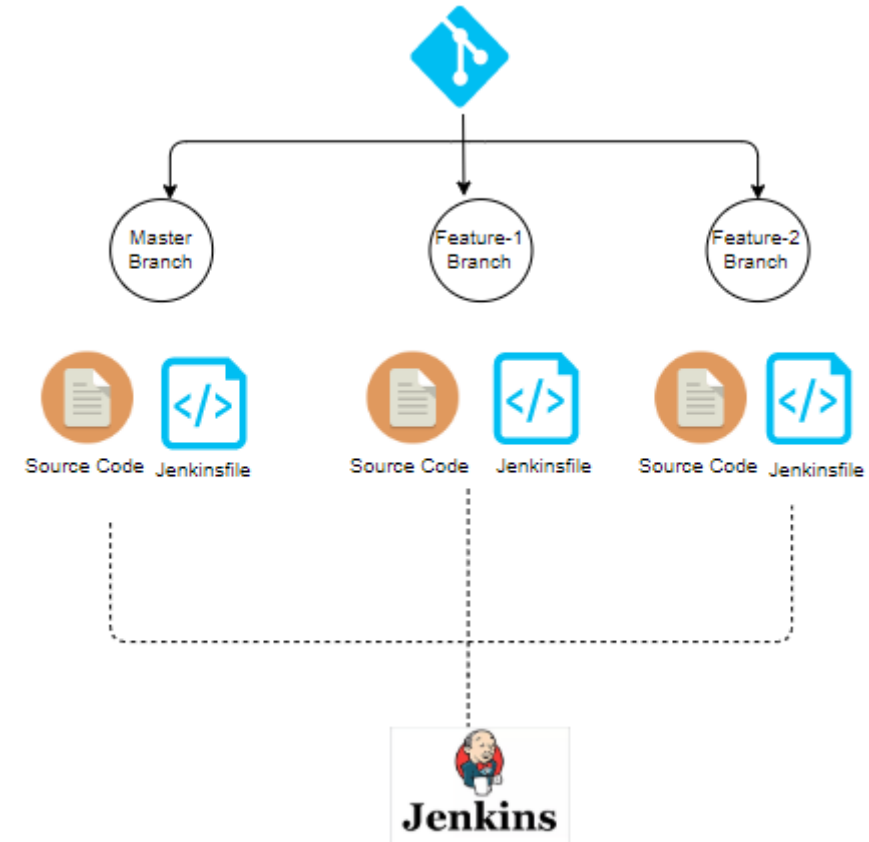
# Typical Webhook Flow

- Developer pushes code
- GitHub/GitLab sends webhook
- Jenkins receives event
- Jenkins determines which pipeline applies
- Build starts



# Multibranch Pipeline

- A single Jenkins job that automatically manages many pipelines
  - One for each branch in a source control repository.
- Instead of manually creating
  - One job per branch
  - One job per feature
  - One job per team
- Jenkins handles this automatically
- A multibranch pipeline job
  - Scans a Git repository
  - Discovers branches and pull requests
  - Creates a pipeline per branch
  - Runs the pipeline defined in that branch



# Multibranch Pipeline

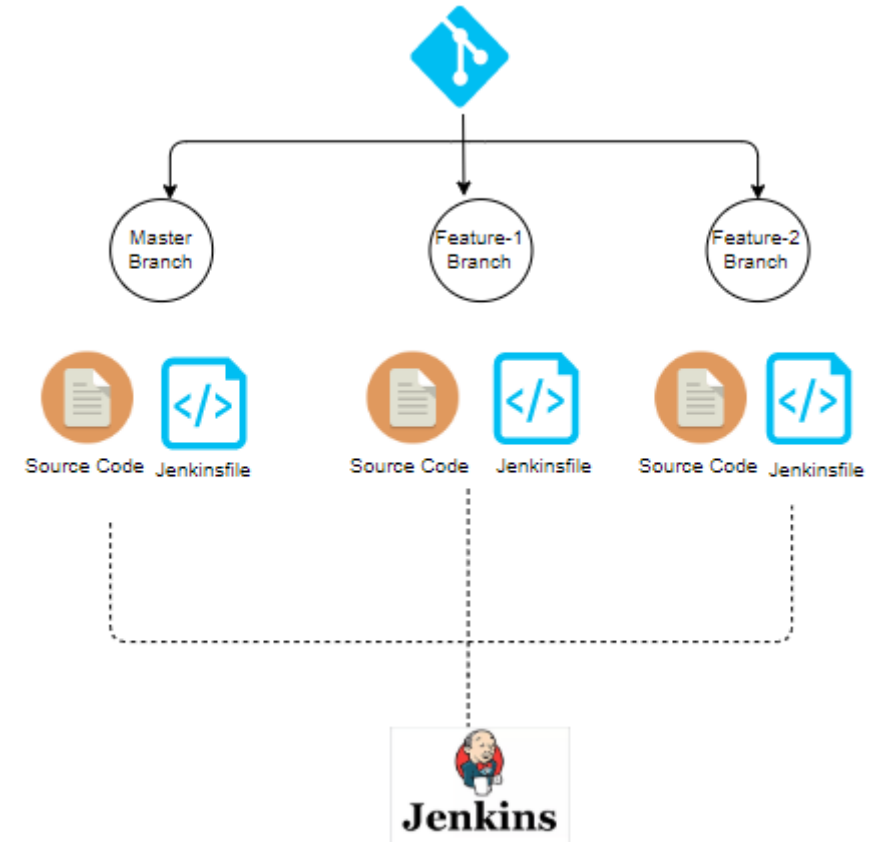
- Given the repo structure on the right
  - With a multibranch pipeline Jenkins creates four pipelines
  - Each pipeline
    - *Uses the Jenkinsfile from its own branch*
    - *Runs independently*
    - *Has its own build history*
    - *Can succeed or fail without affecting others*
    - *For example “if feature/login fails, main is unaffected”*
  - Jenkins reads a Jenkinsfile from the branch itself so each branch can define its own behavior.
  - main branch:
    - *Runs full tests*
    - *Builds artifacts*
    - *Deploys*
  - feature/\* branches:
    - *Run fast tests only*
    - *Skip deployments*
  - This is done using branch-aware logic inside the branch Jenkinsfiles

```
main  
feature/login  
feature/search  
release/1.0
```



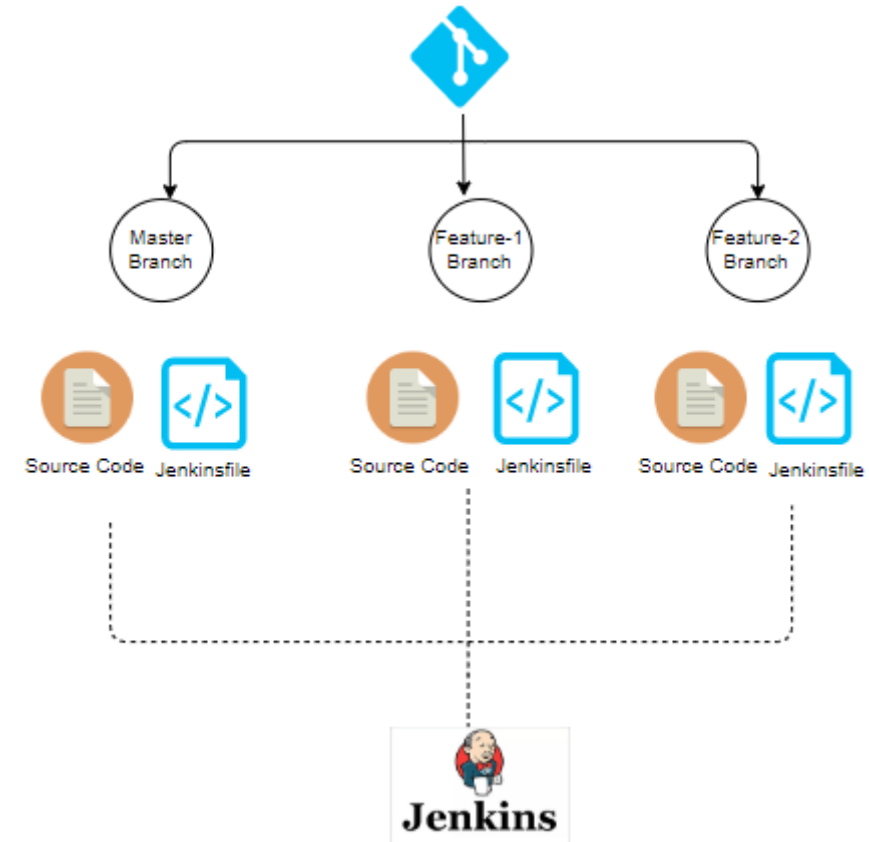
# Multibranch Pipeline

- Multibranch pipelines allow Jenkins to automatically build feature branches
- Developers don't need to
  - Ask for a Jenkins job
  - Copy an existing pipeline
  - Configure anything manually
- They just
  - Create a branch
  - Add a Jenkinsfile
  - Push code
  - Jenkins does the rest



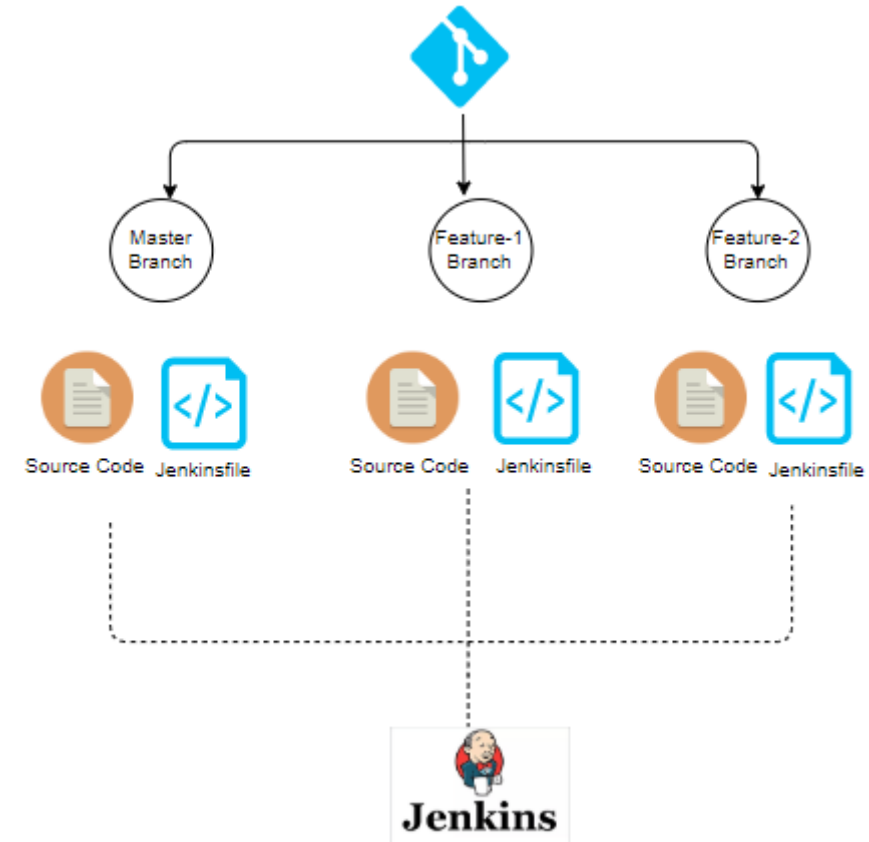
# Multibranch Pipeline

- Multibranch pipelines allow Jenkins to validate changes before merge
- Before code is merged
  - Jenkins builds the branch
  - Tests run automatically
  - Failures are caught early
- This prevents
  - Broken main branches
  - Last-minute integration issues
- CI happens before merge, not after



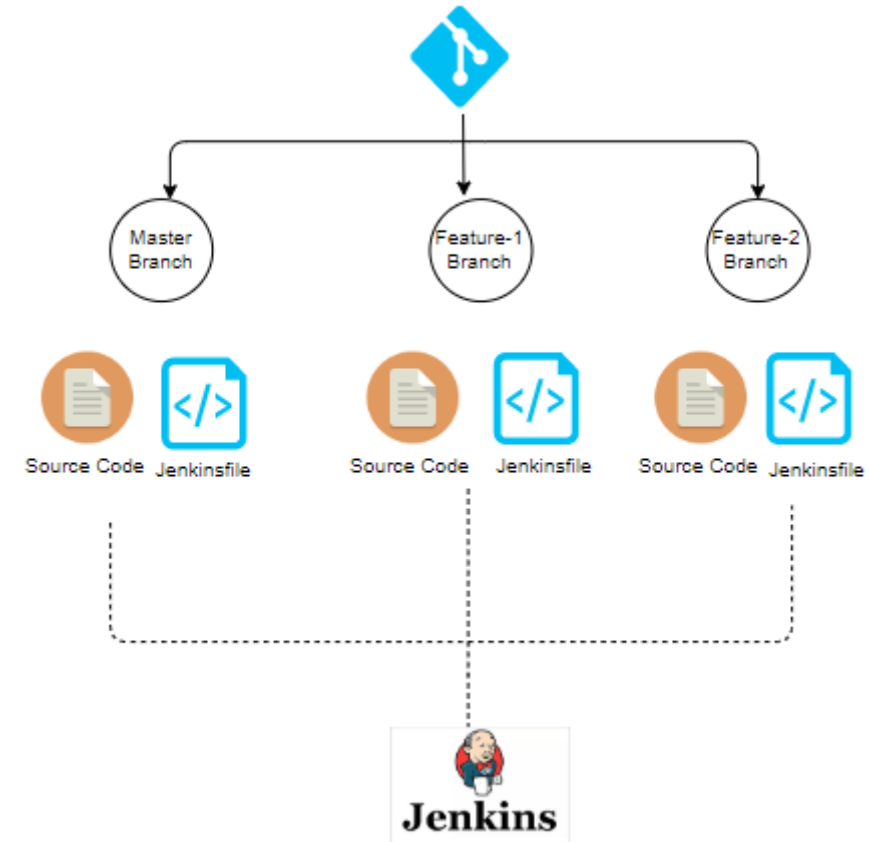
# Jenkinsfile Requirement

- A Jenkinsfile is mandatory for Jenkins to build a branch
  - The branch must contain a Jenkinsfile
  - The file defines
    - *Stages*
    - *Steps*
    - *Agents*
    - *Environment*
    - *Logic*
  - No Jenkinsfile = no pipeline



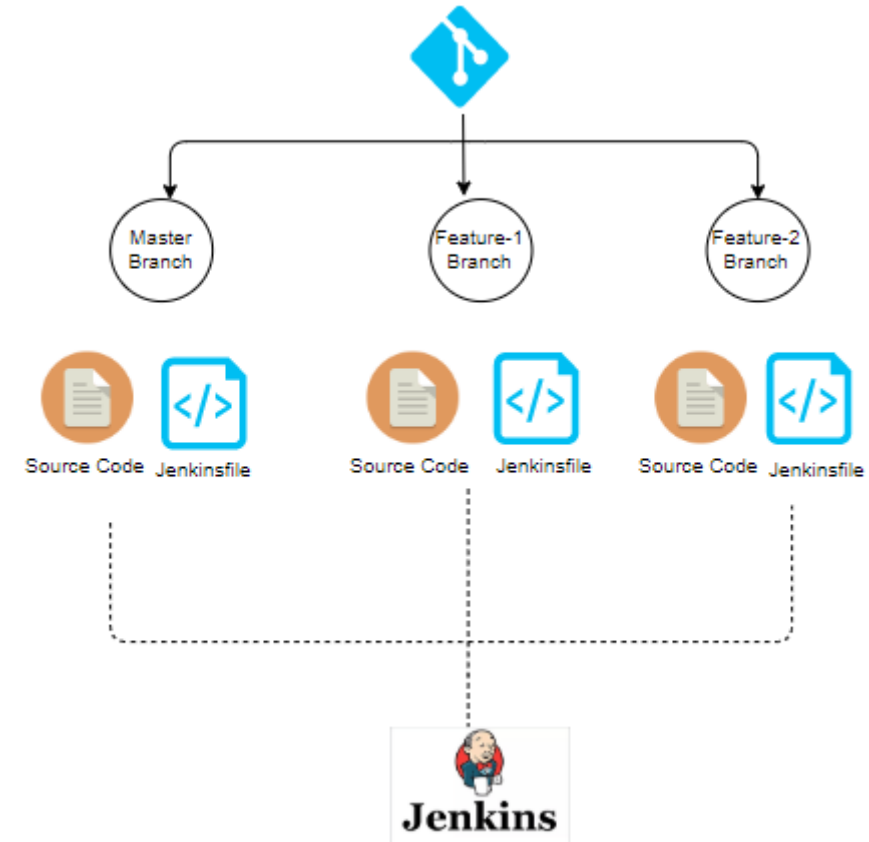
# Multibranch Pipeline Benefits

- Because the Jenkinsfile is part of the branch
  - Pipelines are versioned
  - Changes are reviewed
  - CI evolves alongside the application
- This enables
  - Branch-specific logic
  - Example
    - *New branch experiments with a new test framework*
    - *Old branches continue using the old one*
  - Experimental pipelines
    - *Try new pipeline ideas*
    - *Test tooling changes*
    - *Validate performance improvements*
    - *Without impacting production pipelines*



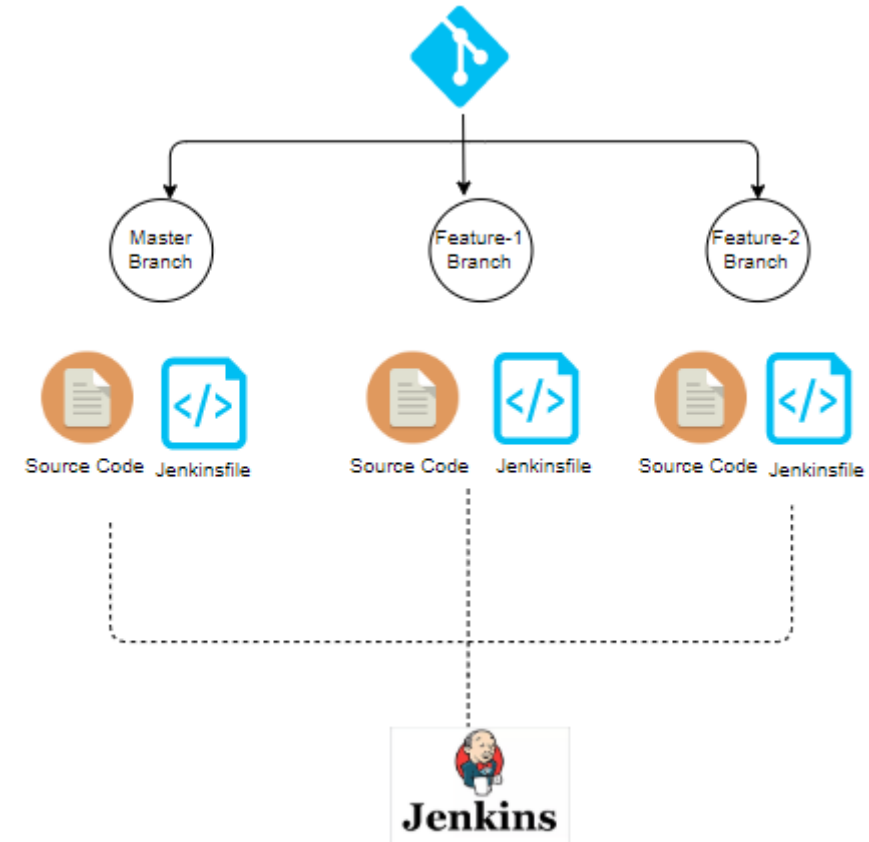
# Branch Discovery

- Jenkins scans the repository
  - Finds branches and PRs
  - Decides which ones to build
  - This can happen periodically through polling
  - Or can happen in response to webhooks
- This behavior is configurable



# Common Discovery Options

- Jenkins can be configured for discovery
- Discover all branches
  - Builds everything
  - Useful for small teams or learning environments
- Exclude certain branches
  - Examples:
    - *Ignore docs/\**
    - *Ignore archived branches*
    - *Ignore experimental branches*
- Build only branches with changes
  - Avoids rebuilding inactive branches
  - Reduces load on Jenkins



# Questions

