Git Assignment

Name: K M Chandrashekhar

DevOps Batch – 1

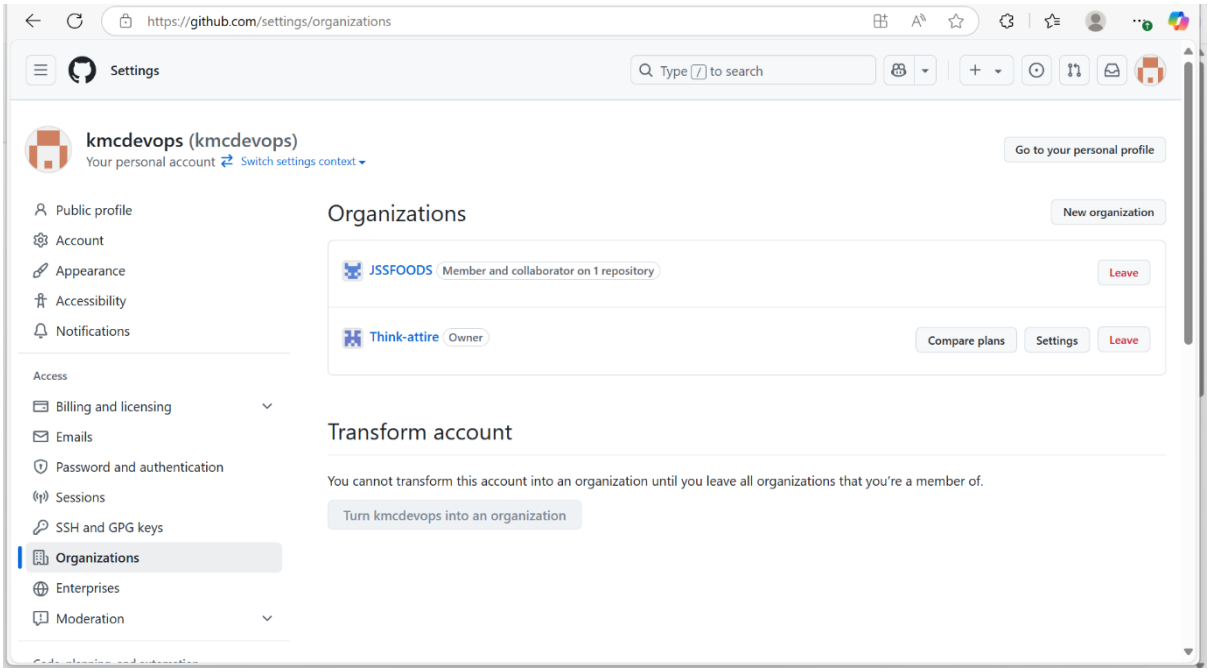
MAY 2025

1. **Create an organization in the github, add a common private repo, add collaborators or team members raise a pull request and get reviews.**

Step 1: Create a GitHub Organization

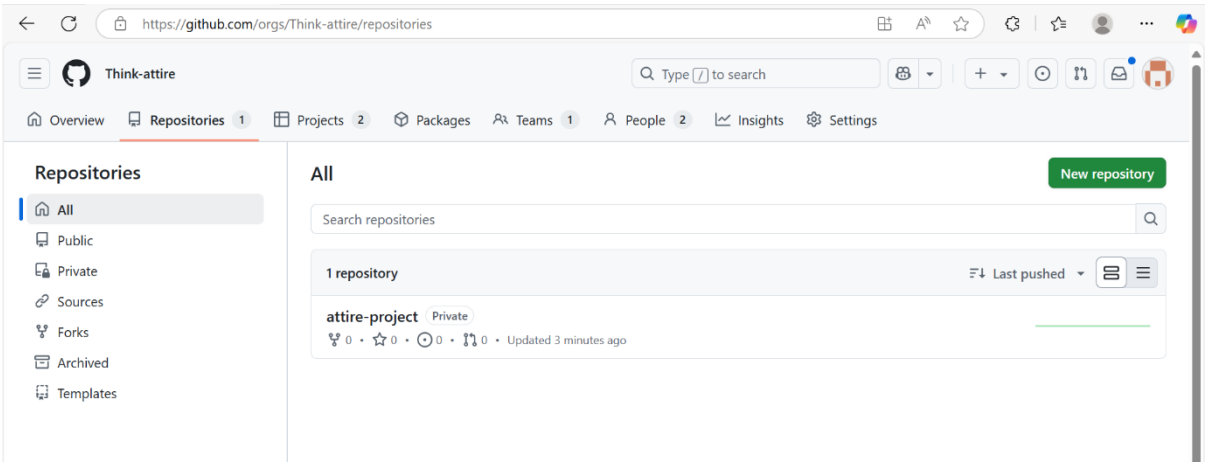
Go to: https://github.com/organizations/new

1. Choose a name for the org (e.g., Think-attire)
2. Select Free plan
3. Finish setup



Step 2: Create a Private Repository in the Organization

Go to your org: https://github.com/orgs/YOUR-ORG-NAME
Click Repositories > New
Name it (e.g., attire project)
Set **Visibility: Private**
Initialize with README if needed

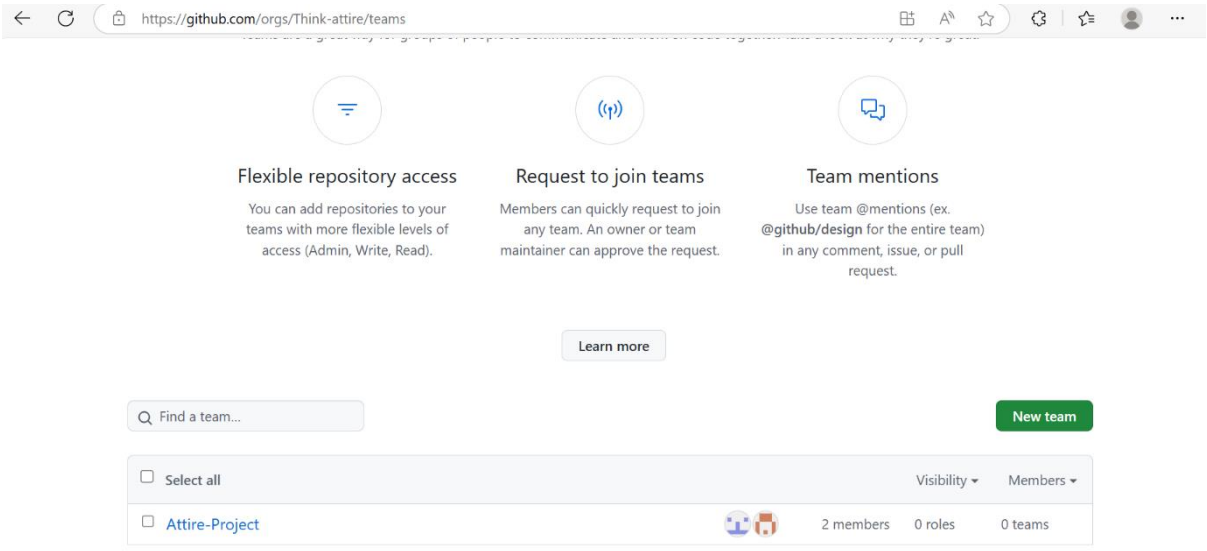


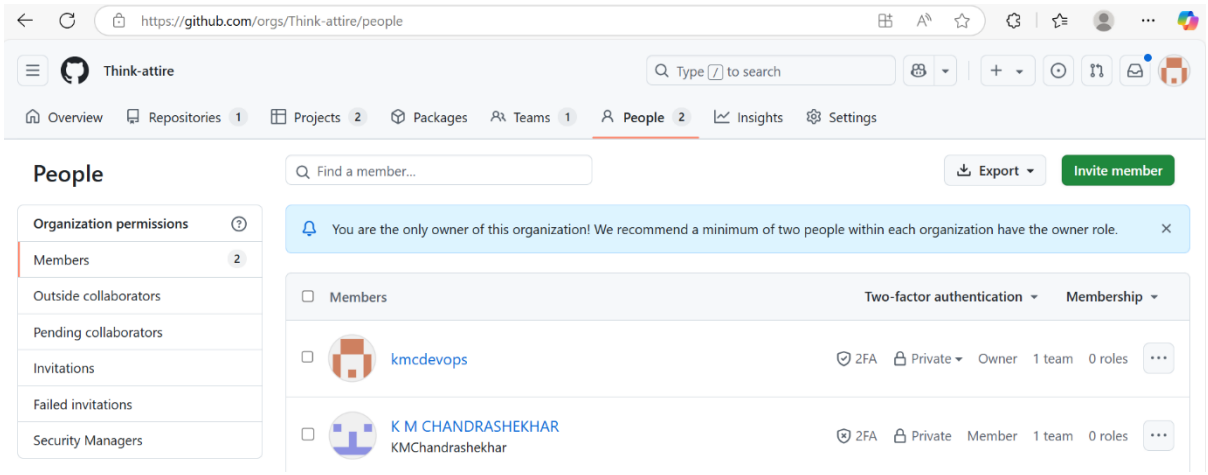
Step 3: Add Collaborators or Teams

In your org, go to People > Teams
Click New team

- Name: e.g., dev-team
- Description: Developers group

Add members (by GitHub username)
Under Repositories, add the repo (team-project) and assign **Write** or **Admin** access





Step 4: Clone the Repo and Create a Feature Branch

```
chandrashekhar@IT003-LT MINGW64 /d
$ git clone https://github.com/Think-attire/attire-project.git
Cloning into 'attire-project'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.

chandrashekhar@IT003-LT MINGW64 /d
$ cd team-project
bash: cd: team-project: No such file or directory

chandrashekhar@IT003-LT MINGW64 /d
$ cd attire-project/

chandrashekhar@IT003-LT MINGW64 /d/attire-project (main)
$ git checkout -b feature/home-page
Switched to a new branch 'feature/home-page'

chandrashekhar@IT003-LT MINGW64 /d/attire-project (feature/home-page)
$ git add .

chandrashekhar@IT003-LT MINGW64 /d/attire-project (feature/home-page)
$ git commit -m "Add basic home page layout"
On branch feature/home-page
nothing to commit, working tree clean

chandrashekhar@IT003-LT MINGW64 /d/attire-project (feature/home-page)
$ git push origin feature/home-page
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'feature/home-page' on GitHub by visiting:
remote:      https://github.com/Think-attire/attire-project/pull/new/feature/home-page
remote:
To https://github.com/Think-attire/attire-project.git
 * [new branch]      feature/home-page -> feature/home-page
```

```
chandrashekhar@IT003-LT MINGW64 /d/attire-project (feature/home-page)
$ echo "<h1>Home Page</h1>" > index.html

chandrashekhar@IT003-LT MINGW64 /d/attire-project (feature/home-page)
$ git add index.html
warning: in the working copy of 'index.html', LF will be replaced by CRLF the next time Git touches it

chandrashekhar@IT003-LT MINGW64 /d/attire-project (feature/home-page)
$ git commit -m "Add basic home page"
[feature/home-page cfea2b7] Add basic home page
 1 file changed, 1 insertion(+)
 create mode 100644 index.html

chandrashekhar@IT003-LT MINGW64 /d/attire-project (feature/home-page)
$ git push origin feature/home-page
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 298 bytes | 298.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Think-attire/attire-project.git
   b5fe4ab..cfea2b7  feature/home-page -> feature/home-page
```

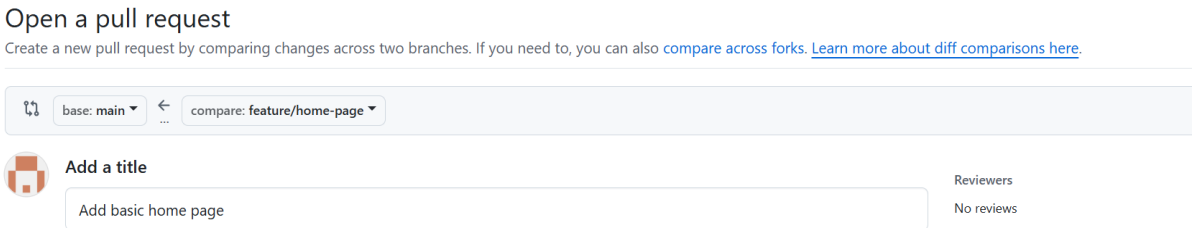
Step 5: Create a Pull Request (PR)

6. Create Pull Request (PR)

1. Go to the repo in GitHub
2. You'll see: **Compare & Pull Request**
3. Set:

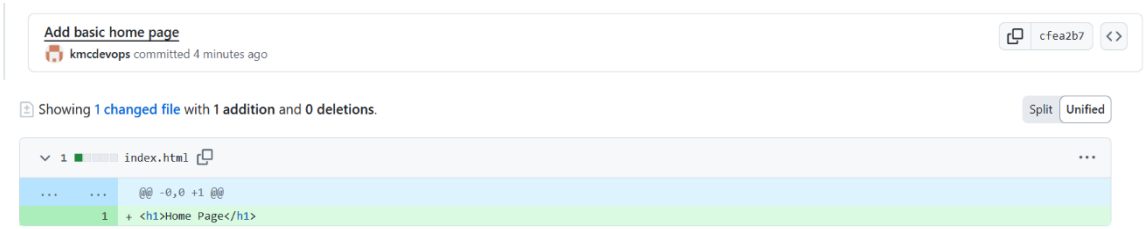
- Base: main
- Compare: feature/home-page

4. Add title, description, and click **Create Pull Request**
5. Assign reviewers (team members)



Then:

- Go to GitHub → Pull Requests → New PR
- Base: main
- Compare: feature/home-page : Now you should see the changes.



2. Compare git revert vs git reset vs git restore.

Git Revert: git revert creates a **new commit** that **undoes the changes** made by a previous commit — without changing the project history.

It's **safe** for public/shared branches (like main or master) because it doesn't rewrite history.
It's **safe** and **non-destructive** (unlike git reset --hard).

Real Scenario: Reverting a Buggy Deployment Commit

Scenario: You are working on a project with a team. You pushed a commit to main that accidentally broke the login functionality. Your team already pulled this commit.
Instead of using **git reset** (which rewrites history and is dangerous on shared branches), you use **git revert** to undo it safely.

Step 1: Create a git-revert-demo folder

```
chandrashekhar@IT003-LT MINGW64 ~/Private-repo/demo (master)
$ mkdir git-revert-demo

chandrashekhar@IT003-LT MINGW64 ~/Private-repo/demo (master)
$ cd git-revert-demo/

chandrashekhar@IT003-LT MINGW64 ~/Private-repo/demo/git-revert-demo (master)
$ git init
Initialized empty Git repository in C:/Users/chandrashekhar/Private-repo/demo/git-revert-demo/.git/
```

Step 2: Create a file and make the first commit

```
chandrashekhar@IT003-LT MINGW64 ~/Private-repo/demo/git-revert-demo (master)
$ echo "Hello World" > file.txt

chandrashekhar@IT003-LT MINGW64 ~/Private-repo/demo/git-revert-demo (master)
$ git add file.txt
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it

chandrashekhar@IT003-LT MINGW64 ~/Private-repo/demo/git-revert-demo (master)
$ git commit -m "Initial commit with Hello World"
[master (root-commit) 03f567f] Initial commit with Hello World
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt
```

Step 3: Add Content to the file.txt

```
chandrashekhar@IT003-LT MINGW64 ~/Private-repo/demo/git-revert-demo (master)
$ echo "This is a demo project." >> file.txt

chandrashekhar@IT003-LT MINGW64 ~/Private-repo/demo/git-revert-demo (master)
$ git add file.txt
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it

chandrashekhar@IT003-LT MINGW64 ~/Private-repo/demo/git-revert-demo (master)
$ git commit -m "Added demo project description"
[master 8210f1e] Added demo project description
 1 file changed, 1 insertion(+)
```

Step 4: Add a bug line (simulating a mistake)

```
chandrashekhar@IT003-LT MINGW64 ~/Private-repo/demo/git-revert-demo (master)
$ echo "Bug line added here" >> file.txt

chandrashekhar@IT003-LT MINGW64 ~/Private-repo/demo/git-revert-demo (master)
$ git commit -am "Added buggy line"
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it
[master 8be20bb] Added buggy line
 1 file changed, 1 insertion(+)
```

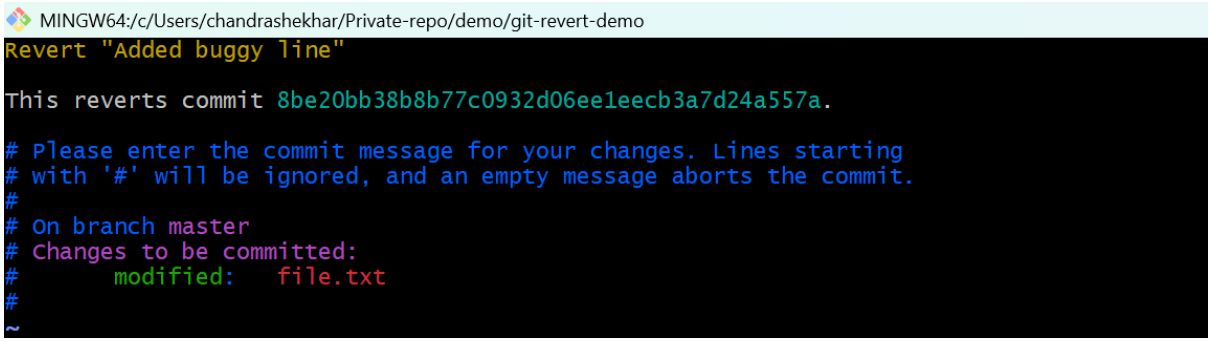
Step 5: Check your commit history

```
chandrashekhar@IT003-LT MINGW64 ~/Private-repo/demo/git-revert-demo (master)
$ git log --oneline
8be20bb (HEAD -> master) Added buggy line
8210f1e Added demo project description
03f567f Initial commit with Hello World
```

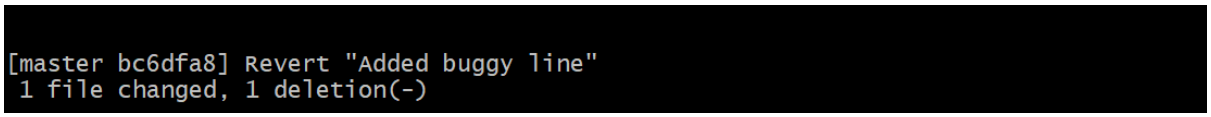
Step 6: Revert the buggy commit

Use the commit ID of the buggy line i.e., 8be20bb

```
chandrashekhar@IT003-LT MINGW64 ~/Private-repo/demo/git-revert-demo (master)
$ git revert 8be20bb
```



Press :wq if the default editor opens (to save the message and exit).



You'll see that **"Bug line added in the file"** has been removed.

Git Reset: git reset is used to undo commits or changes in the working directory or staging area. It changes the current **HEAD**, **index (staging area)**, and optionally the **working directory** depending on the type of reset. **OR**

git reset moves the **HEAD** and optionally updates the **staging area** (index) and **working directory** depending on the type:

Type	Moves HEAD	Updates Staging	Updates Working Dir
--soft	Yes	No	No
--mixed (default)	Yes	Yes	No
--hard	Yes	Yes	Yes

Real-World Use Cases with Examples Let's walk through a scenario where you made some commits and want to undo them:

Setup: Create a sample repo

```
MINGW64:/c/Users/chandrashekhar/test/git-reset-demo

chandrashekhar@IT003-LT MINGW64 ~/test (main)
$ mkdir git-reset-demo

chandrashekhar@IT003-LT MINGW64 ~/test (main)
$ cd git-reset-demo/

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (main)
$ git init
Initialized empty Git repository in C:/Users/chandrashekhar/test/git-reset-demo/.git/

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ echo "line 1" > file.txt

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ git add .
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ git commit -m "1st Commit"
[master (root-commit) 53bb063] 1st Commit
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ echo "line 2" >> file.txt

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ git add .
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ git commit -m "2nd commit"
[master da9a97e] 2nd commit
 1 file changed, 1 insertion(+)
```

```
chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ echo "line 3" >> file.txt

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ git add .
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ git commit -m "3rd commit"
[master d559af1] 3rd commit
 1 file changed, 1 insertion(+)

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ git log --oneline
d559af1 (HEAD -> master) 3rd commit
da9a97e 2nd commit
53bb063 1st Commit
```

1. git reset --soft – Undo commits but keep all changes staged

Use case: You want to squash or modify previous commits but keep the changes ready to commit again. OR You made 3 commits but want to squash them into one OR You want to combine multiple commits into one

```
MINGW64:/c/Users/chandrashekhar/test/git-reset-demo

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ git reset --soft HEAD~1
```

- Moves HEAD to **Commit 2**
- All changes from **Commit 3** remain in the **staging area**

```
MINGW64:/c/Users/chandrashekhar/test/git-reset-demo

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ git reset --soft HEAD~1

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   file.txt


chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ git log --oneline
da9a97e (HEAD -> master) 2nd commit
53bb063 1st Commit

chandrashekhar@IT003-LT MINGW64 ~/test/git-reset-demo (master)
$ cat file.txt
line 1
line 2
line 3
```

2. git reset --mixed <commit> (default) – Moves HEAD Undo commits and unstage changes

Use case: You staged files accidentally but haven't committed yet **OR** You want to undo a commit and re-edit the files before recommitting.

Use when: You want to unstage files without losing your code changes.
$git reset --mixed HEAD~1

```
chandrashekhar@IT003-LT MINGW64 ~/reset-demo (master)
$ git log --oneline
156cc0f (HEAD -> master) 3rd commit
a9e35e8 2nd commit
f2b045d 1st commit

chandrashekhar@IT003-LT MINGW64 ~/reset-demo (master)
$ ^C

chandrashekhar@IT003-LT MINGW64 ~/reset-demo (master)
$

chandrashekhar@IT003-LT MINGW64 ~/reset-demo (master)
$ git reset --mixed HEAD~1
Unstaged changes after reset:
M       file.txt

chandrashekhar@IT003-LT MINGW64 ~/reset-demo (master)
$ git log --oneline
a9e35e8 (HEAD -> master) 2nd commit
f2b045d 1st commit

chandrashekhar@IT003-LT MINGW64 ~/reset-demo (master)
$ cat file.txt
1st line
2nd line
3rd line
```

What it does:

- **Moves HEAD** to the 2nd commit (a9e35e8)
- **Unstages** the changes from the 3rd commit (156cc0f)
- **Keeps** the file changes in your **working directory**

Result:

- The 3rd commit is **undone** (removed from commit history)
- But all the changes from that commit still exist in your files (not lost!)
- Those changes are now **unstaged**, ready to be re-added or modified

Use Case: You realized the 3rd commit was premature or incorrect, but you **still want the changes**, just not committed yet. This lets you fix or modify them before recommitting.

3. git reset --hard : You want to delete commits and the changes you made OR git reset hard is used to **forcefully reset** your Git repository to a previous commit

What Happens:

- **HEAD moves back** to the 1st commit
- The **3rd commit & 2nd commit is deleted**
- All **file changes made in the 3rd & 2nd commit are also erased** from:
  - The **commit history**
  - The **staging area**
  - The **working directory**

Warning: This is **destructive**! Don't use --hard unless you're **absolutely sure**

3. git restore: command, which is designed to **safely undo changes** to files **without touching commit history**.

Step 1: Create Directory, git Initialize and Create File add the content

Step 2: Modify the File

```
chandrashekhar@IT003-LT MINGW64 ~/reset-demo (master)
$ mkdir restore-demo && cd restore-demo

chandrashekhar@IT003-LT MINGW64 ~/reset-demo/restore-demo (master)
$ git init
Initialized empty Git repository in C:/Users/chandrashekhar/reset-demo/restore-demo/.git/

chandrashekhar@IT003-LT MINGW64 ~/reset-demo/restore-demo (master)
$ echo "line1" > file.txt

chandrashekhar@IT003-LT MINGW64 ~/reset-demo/restore-demo (master)
$ git add .
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it

chandrashekhar@IT003-LT MINGW64 ~/reset-demo/restore-demo (master)
$ git commit -m "1st commit"
[master (root-commit) 349ed0d] 1st commit
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt

chandrashekhar@IT003-LT MINGW64 ~/reset-demo/restore-demo (master)
$ echo "line2" >> file.txt

chandrashekhar@IT003-LT MINGW64 ~/reset-demo/restore-demo (master)
$ cat file.txt
line1
line2

chandrashekhar@IT003-LT MINGW64 ~/reset-demo/restore-demo (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   file.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

```
chandrashekhar@IT003-LT MINGW64 ~/reset-demo/restore-demo (master)
$ git restore file.txt

chandrashekhar@IT003-LT MINGW64 ~/reset-demo/restore-demo (master)
$ cat file.txt
line1
```

Result:

- line2 is removed
- file.txt now contains only line1
- You discarded local changes

Action	Command
Discard changes in file	git restore file.txt
Unstage a file	git restore --staged file.txt
Restore file from old commit	git restore --source=<commit> file.txt

3. Compare git checkout vs git switch.

git switch : command is a modern, user-friendly way to change branches **or** create new branches **in Git.**

git checkout : A multi-purpose command used to:

- Switch between branches
- Restore files
- Create new branches

Use Cases of git switch

Use Case	Command	Description
Switch to existing branch	git switch branch-name	Move to another branch
Create and switch to new branch	git switch -c new-branch	Shortcut to checkout -b
Switch back to main/master	git switch main	Common during merging or comparison

Simple Example with **`file.txt`**

Create Directory(Repository) , git initialize and create file add content

```
chandrashekhar@IT003-LT MINGW64 /d/switch-demo (master)
$ mkdir switch-demo

chandrashekhar@IT003-LT MINGW64 /d/switch-demo (master)
$ cd switch-demo/

chandrashekhar@IT003-LT MINGW64 /d/switch-demo/switch-demo (master)
$ cd

chandrashekhar@IT003-LT MINGW64 ~
$ cd /d

chandrashekhar@IT003-LT MINGW64 /d
$ cd switch-demo/

chandrashekhar@IT003-LT MINGW64 /d/switch-demo (master)
$ git init
Reinitialized existing Git repository in D:/switch-demo/.git/

chandrashekhar@IT003-LT MINGW64 /d/switch-demo (master)
$ echo "line1" > file.txt

chandrashekhar@IT003-LT MINGW64 /d/switch-demo (master)
$ git add .
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it

chandrashekhar@IT003-LT MINGW64 /d/switch-demo (master)
$ git commit -m "Initial commit"
[master (root-commit) 9678d93] Initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt
```

Step 2: Create and Switch to New Branch

```
chandrashekhar@IT003-LT MINGW64 /d/switch-demo (master)
$ git switch -c feature-1
Switched to a new branch 'feature-1'

chandrashekhar@IT003-LT MINGW64 /d/switch-demo (feature-1)
$
```

Now you're on a new branch called feature-1.

Add line in feature branch and switch to main branch

git switch main But line 2 not added in main as because we created in feature branch.

Common Use Cases

Task	Old (`git checkout`)	New (`git switch`)	Description
Switch to existing branch	git checkout main	git switch main	Changes to main branch
Create and switch to new branch	git checkout -b dev	git switch -c dev	Creates and moves to dev
Restore a file	git checkout -- file.txt	Use git restore file.txt	Separate command for clarity

4. Use git diff for file and branch comparison.

git diff is used to: See changes in your working directory vs the last commit

- Compare branches
- Compare commits
- Compare staged vs unstaged changes

- **Simple Eg for git Diff for specific file**

```
MINGW64:/d/diff-demo

chandrashekhar@IT003-LT MINGW64 /d
$ mkdir diff-demo

chandrashekhar@IT003-LT MINGW64 /d
$ cd diff-demo/

chandrashekhar@IT003-LT MINGW64 /d/diff-demo
$ git init
Initialized empty Git repository in D:/diff-demo/.git/

chandrashekhar@IT003-LT MINGW64 /d/diff-demo (master)
$ echo "line1" > file.txt

chandrashekhar@IT003-LT MINGW64 /d/diff-demo (master)
$ git add file.txt
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it

chandrashekhar@IT003-LT MINGW64 /d/diff-demo (master)
$ git commit -m "initial commit"
[master (root-commit) a86f898] initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt
```

- **Compare Working Directory vs Last Commit**

Edit the File and use the command git diff file.txt

```
MINGW64:/d/diff-demo

chandrashekhar@IT003-LT MINGW64 /d/diff-demo (master)
$ echo "line2" >> file.txt

chandrashekhar@IT003-LT MINGW64 /d/diff-demo (master)
$ git diff file.txt
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it
diff --git a/file.txt b/file.txt
index a29bdeb..c0d0fb4 100644
--- a/file.txt
+++ b/file.txt
@@ -1 +1,2 @@
 line1
+line2
```

Output: Shows that line2 was added since the last commit.

2. Compare Staged vs Last Commit

```
chandrashekhar@IT003-LT MINGW64 /d/diff-demo (master)
$ git add file.txt
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it

chandrashekhar@IT003-LT MINGW64 /d/diff-demo (master)
$ git diff --cached
diff --git a/file.txt b/file.txt
index a29bdeb..c0d0fb4 100644
--- a/file.txt
+++ b/file.txt
@@ -1 +1,2 @@
 line1
+line2
```

Output: Shows staged changes that will go into the next commit.

3. Compare Two Branches

Step 1: Create a new branch and switch to new branch

```
chandrashekhar@IT003-LT MINGW64 /d/diff-demo (master)
$ git switch -c feature
Switched to a new branch 'feature'

chandrashekhar@IT003-LT MINGW64 /d/diff-demo (feature)
$ echo "line3" >> file.txt

chandrashekhar@IT003-LT MINGW64 /d/diff-demo (feature)
$ git commit -am "Add line3 in feature branch"
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it
[feature 8e41cbb] Add line3 in feature branch
 1 file changed, 2 insertions(+)
```

Now switch back to master

```
chandrashekhar@IT003-LT MINGW64 /d/diff-demo (feature)
$ git switch master
Switched to branch 'master'

chandrashekhar@IT003-LT MINGW64 /d/diff-demo (master)
$ git diff main..feature
fatal: ambiguous argument 'main..feature': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'

chandrashekhar@IT003-LT MINGW64 /d/diff-demo (master)
$ git diff master..feature
diff --git a/file.txt b/file.txt
index a29bdeb..83db48f 100644
--- a/file.txt
+++ b/file.txt
@@ -1 +1,3 @@
 line1
+line2
+line3

chandrashekhar@IT003-LT MINGW64 /d/diff-demo (master)
$ git diff feature..master
diff --git a/file.txt b/file.txt
index 83db48f..a29bdeb 100644
--- a/file.txt
+++ b/file.txt
@@ -1,3 +1 @@
 line1
-line2
-line3
```

Compare master and feature:

git diff master..feature

Shows what's in feature branch that's not in main.

git diff feature..master

Shows what's in main but **not** in feature.

4. Compare a File Between Branches

git diff main..feature -- file.txt

MINGW64:/d/diff-demo

```
chandrashekhar@IT003-LT MINGW64 /d/diff-demo (master)
$ git diff master..feature -- file.txt
diff --git a/file.txt b/file.txt
index a29bdeb..83db48f 100644
--- a/file.txt
+++ b/file.txt
@@ -1 +1,3 @@
 line1
+line2
+line3

chandrashekhar@IT003-LT MINGW64 /d/diff-demo (master)
$ git diff feature..master -- file.txt
diff --git a/file.txt b/file.txt
index 83db48f..a29bdeb 100644
--- a/file.txt
+++ b/file.txt
@@ -1,3 +1 @@
 line1
-line2
-line3
```

Output: Shows changes in file.txt specifically between branches

4. **Understand merge conflicts and their resolution: -** A **merge conflict** happens when **two branches change the same part of the same file**, and Git cannot decide which change to keep.

```
MINGW64:/d/merge-conflict-demo/merge-conflict

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo (master)
$ mkdir merge-conflict && cd merge-conflict

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (master)
$ git init
Initialized empty Git repository in D:/merge-conflict-demo/merge-conflict/.git/

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (master)
$ echo "line1" > file.txt

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (master)
$ git add file.txt
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (master)
$ git commit -m "initial commit"
[master (root-commit) 130605e] initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 file.txt
```

Step 1: Create Two Branches from master branch and add the content in line 2 for same file.txt

```
chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (master)
$ git switch -c feature-A
Switched to a new branch 'feature-A'

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (feature-A)
$ echo "line2 from feature-A" >> file.txt

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (feature-A)
$ git commit -am "Add line2 in feature-A"
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it
[feature-A 378db6e] Add line2 in feature-A
 1 file changed, 1 insertion(+)

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (feature-A)
$ git switch master
Switched to branch 'master'

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (master)
$ git switch -c feature-B
Switched to a new branch 'feature-B'

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (feature-B)
$ echo "line2 from feature-B" >> file.txt

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (feature-B)
$ git commit -am "Add line2 in feature-B"
warning: in the working copy of 'file.txt', LF will be replaced by CRLF the next time Git touches it
[feature-B 94d053b] Add line2 in feature-B
 1 file changed, 1 insertion(+)
```

Now both branches modified the same line of file.txt.

Step 2: Merge and Create Conflict

```
chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (feature-B)
$ git merge feature-A
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
Automatic merge failed; fix conflicts and then commit the result.
```

```
MINGW64:/d/merge-conflict-demo/merge-conflict
line1
<<<<<<< HEAD
line2 from feature-B
=======
line2 from feature-A
>>>>>>> feature-A
~
~
~
~
~
```

Git is asking: Which line should I keep? feature-B or feature-A?

Step 3: Resolve the Conflict

```
chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (feature-B|MERGING)
$ vi file.txt

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (feature-B|MERGING)
$ git add file.txt

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (feature-B|MERGING)
$ git commit -m "Resolve merge conflict between feature-A and feature-B"
[feature-B 310ec67] Resolve merge conflict between feature-A and feature-B

chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (feature-B)
$ git merge feature-A
Already up to date.
```

Conflict resolved.

Real-World Scenario

You and a teammate both edit README.md on different branches:

- You write: App supports login via email
- They write: App supports login with OAuth

You both commit and later merge — conflict happens because **Git doesn't know which text is correct**. You must manually **review and decide**.

Scenario 1: Multiple Developers Edit the Same File Section

File: config.yml

Developer A (branch: feature-login)
auth:
  method: password

Developer B (branch: feature-social-login)
auth:
  method: oauth

When merged → **Conflict**, because both edited the **same line** differently.

Solution: Decide whether to support both or pick one method in the merge.

Scenario 2: Rename File in One Branch, Edit in Another

Developer A:

- Renames app.js to main.js in branch-A

Developer B:

- Edits app.js in branch-B

When merging → Git doesn't know whether to keep the new name or preserve the edit.

Solution: Manually copy the edits into main.js, remove old app.js.

Scenario 3: Deletion vs Modification Conflict

Merge conflict: Git is confused — one person deleted it, another modified it.

Solution: Decide whether to keep the file with changes or remove it.

Scenario 4: Same Dependency Updated Differently

File: package.json

- **feature-A** adds "axios": "^0.27.0"

- **feature-B** adds "axios": "^1.3.0"

Merge conflict on the version number.

Solution: Choose the correct version

Scenario 5: Binary File Conflict

- Both users edit and commit changes to logo.png
- Git can't merge binary files

Conflict occurs

Solution: Keep one version, rename the other, or manually choose which to use.

Scenario 6: Rebase Conflict During Pull

You're on main and run:

git pull --rebase origin main

conflict happens because your local commits modify the same lines as the ones just pulled.

Solution:

git status

Fix conflicts

git add .

git rebase –continue

Conflict Type	Cause	Solution
Same line edit	Edited by multiple branches	Manual merge
Rename + edit	One renamed, one edited	Manual fix
Delete + edit	One deleted, one edited	Decide to keep or delete
Binary files	Both edited	Choose one
Dependency/version change	Different versions of same package	Pick compatible version

Use git tag to create and manage tags

git tag is used to mark specific points in a repository's history — usually for:

Releases (e.g., v1.0, v2.1.3)

Stable builds

Deployment versions

Tags are like bookmarks for important commits.

When to Use git tag (Real-World Use Cases)

Scenario	Description
Release versioning	Mark production-ready builds (`v1.0`, `v2.1.5`)
QA checkpoint	Tag code for QA before new release
Deployment tracking	Tag every deployment with timestamp or release ID
Rollback reference	Easy to revert if a deployment breaks
Package publishing	Many CI/CD pipelines fetch by tag

Step-by-Step Usage of git tag

Step 1: View Commit History

git log – oneline

```
chandrashekhar@IT003-LT MINGW64 /d/merge-conflict-demo/merge-conflict (feature-B)
$ git log --oneline
310ec67 (HEAD -> feature-B) Resolve merge conflict between feature-A and feature-B
94d053b Add line2 in feature-B
378db6e (feature-A) Add line2 in feature-A
130605e (master) initial commit
```

Step 2: Create a Tag at the Latest Commit or Or for a specific commit:

git tag v1.0

This creates a **lightweight tag** pointing to the commit.

Step 3: Push the Tag to Remote

git push origin v1.0

Push **all** tags:

git push –tags

Step 4: List Tags

git tag

Step 5: Add Annotated Tag (Better for Releases)

Annotated tags store more info (message, date, author):

git tag -a v1.0 -m "Release version 1.0 - stable build"

then push

git push origin v1.0

Step 6: Checkout a Tag (Read-Only)

git checkout v1.0

Note: You are now in **detached HEAD** state. To make changes:

git checkout -b fix-v1.0

Understand GitHub Actions for CI/CD.

GitHub Actions is a built-in CI/CD tool in GitHub that lets you **automate workflows** such as:

- Run tests
- Build your app
- Deploy to servers or cloud (AWS, Docker, etc.)

It's defined using **YAML files** inside. github/workflows/ directory in your repo.

Why Use GitHub Actions for CI/CD?

CI (Continuous Integration)	CD (Continuous Deployment)
Run unit tests automatically	Push code to staging/production
Build app when code is committed	Deploy Docker containers
Ensure quality before merging	Auto-release apps after approval

Common Use Cases

Task	Example Action
Run tests	npm test, pytest
Build Docker	docker build, docker push
Deploy to AWS	scp, aws-cli, terraform
Publish to npm	npm publish
Send Slack notification	8398a7/action-slack
Build Android/iOS apps	actions/upload-artifact, Fastlane

Benefits of GitHub Actions

Native to GitHub (no 3rd-party setup)
Easy to connect with secrets, tags, branches
Huge community of pre-built actions
Free for public repos

Deploy .jar file of movie analyzer assignment to the webapps folder of tomcat server/application and see what happens.

A. BACKEND
** Java requires a build option. It is a complied language. Cannot run java directly just by giving java and the .jar file (java <my_app.jar>)**
for java-based applications we use maven
maven----->pom.xml (maven looks for pom.xml to run)

Steps to Build Java:
---> Clone the repository (git clone <repo_url>)
---> Go to cd movie-analyzer-->ls-->backend
---> Run mvn clean package where pom.xml is located to build the app using maven
---> Generated output .jar file is in the target/directory {cd movie-analyzer-->ls-->backend
-->ls-->target}

Steps to Deploy JAR File in to Tomcat:
---> Edit pom.xml by changing from jar to war
Find this line <packaging>jar</packaging> and replace with<packaging>war</packaging>

Inside the pom.xml:
Highlighted are the changes to edit inside the file

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
```

http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<parent>

<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>3.3.0</version>
<relativePath/>
</parent>

<groupId>com.moviereview</groupId>
<artifactId>movie-review-backend</artifactId>
<version>1.0.0</version>

<packaging>war</packaging>

<name>Movie Review Backend</name>
<description>Movie Review Backend Service - DevOps Demo</description>
<properties>
<java.version>21</java.version>
<maven.compiler.source>21</maven.compiler.source>
<maven.compiler.target>21</maven.compiler.target>
<maven.compiler.parameters>true</maven.compiler.parameters>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

---> Add Tomcat dependency:
Inside the pom.xml:
Highlighted are the changes to edit inside the file

<dependencies>
<!-- Spring Boot Starters -->
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-tomcat</artifactId>
<scope>provided</scope>
</dependency>

```
 <dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-data-jpa</artifactId>
 </dependency>

<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-actuator</artifactId>
 </dependency>

 <!-- PostgreSQL Driver -->
 <dependency>
 <groupId>org.postgresql</groupId>
 <artifactId>postgresql</artifactId>
 <scope>runtime</scope>
 </dependency>

 <!-- HTTP Client for Model Server -->
 <dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-webflux</artifactId>
 </dependency>
```

---> Edit MovieReviewApplication.java

Go to this path

{backend/src/main/java/com/moviereview/MovieReviewApplication.java}

Inside the MovieReviewApplication.java:

Highlighted are the changes to edit inside the file

```
package com.moviereview;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.web.reactive.function.client.WebClient;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication
public class MovieReviewApplication extends SpringBootServletInitializer {
```

```
@Value("${model.server.url}")
 private String modelServerUrl;

 public static void main(String[] args) {
 System.out.println(" Starting Movie Review Backend Service");
 System.out.println(" This service is designed to be resilient to downstream
failures");
System.out.println("Database connection will be tested at runtime, not startup");
 System.out.println(" Model server connection will be tested per request");

 SpringApplication.run(MovieReviewApplication.class, args);
 }

@Override
protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
 return builder.sources(MovieReviewApplication.class);
 }
 @Bean
 public WebClient webClient() {
 return WebClient.builder()
 .baseUrl(modelServerUrl)
 .build();
 }
}
```

---> Build WAR by running this: mvn clean package (make sure to be inside the backend directory and run)
---> Get into the target directory and run ls & now will able to see the .jar file converted in to
.war file
---> Copy the .war file into webapps folder of tomcat
{To check webapps folder path---> cd /opt/tomcat10/webapps}

sudo cp /home/ec2-user/movie-analyzer/backend/target/movie-review-backend 1.0.0.war /opt/ tomcat/ webapps

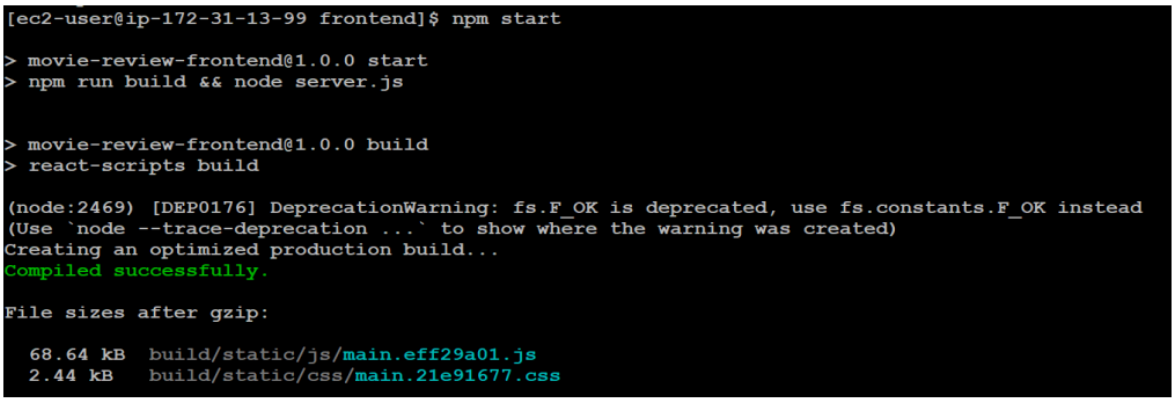
---> To view output on the terminal
curl http://<Public_IP>:8080/movie-review-backend-1.0.0/api/admin/health
==========

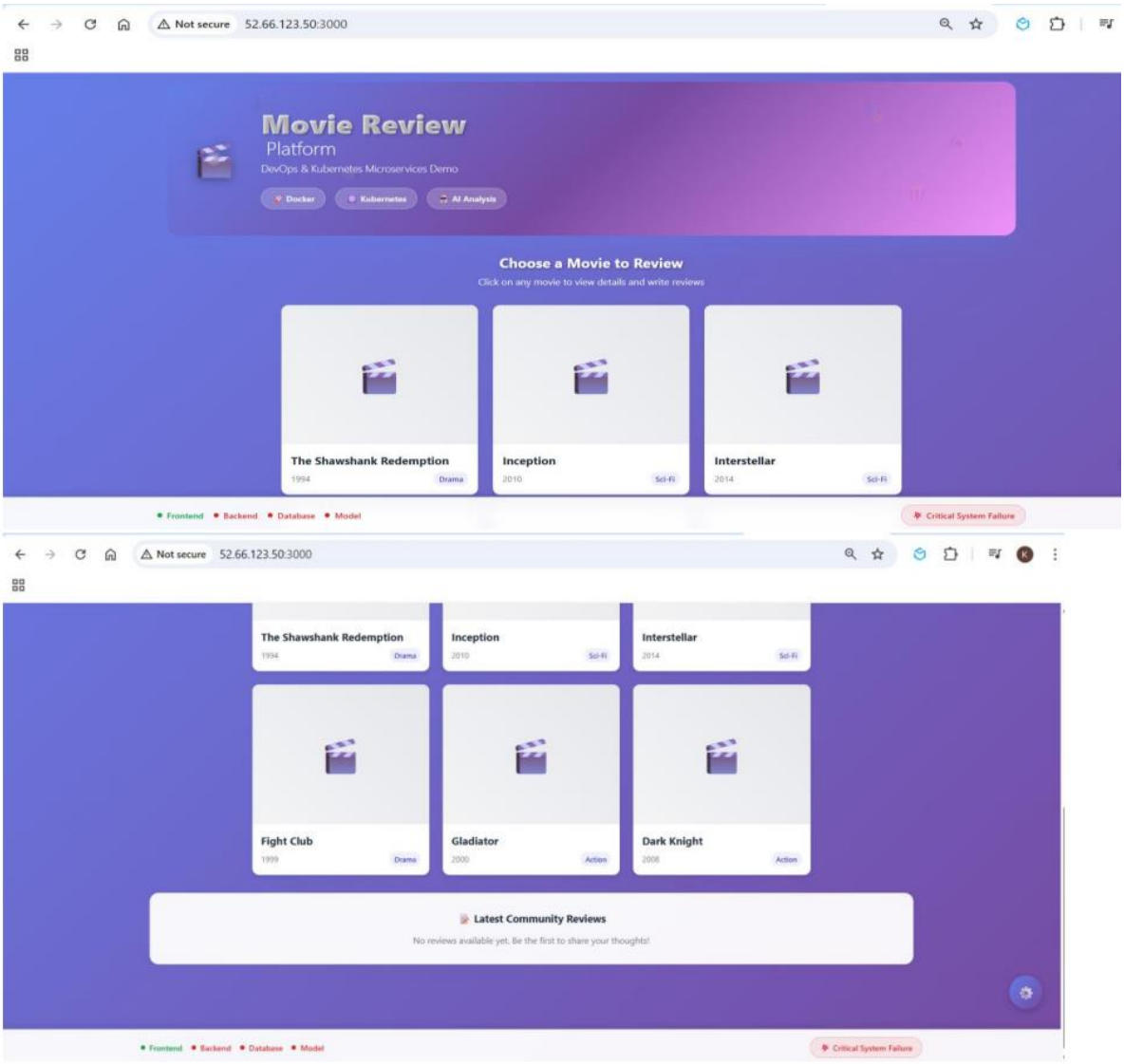
Output: Spring Boot backend has been successfully deployed and running on Apache Tomcat. A JSON response is shown in the below snippet confirms that .war file is deployed without errors.

To access the output via browser: http://<Public_IP>:8080/movie-review-backend-1.0.0/api/admin/health
To access the output via terminal: curl http://<Public_IP>:8080/movie-review-backend-1.0.0/api/admin/health

FRONTEND
for node js based applications we use npm
node.js----->package.json (node.js looks for package.json to run the application)

Steps to start an application:
---> Make sure to use nvm to install npm
---> Clone the repository (git clone <repo_url>)
---> Go to {cd movie-analyzer---> frontend}
---> Run npm install (to install dependencies)
---> Run npm start (to start frontend application)
---> Run {http://<ip_address>:3000} in browser to see frontend UI of the application




Not secure
52.66.123.50:3000
Movie Review
Platform
DevOps & Kubernetes Microservices Demo
Docker
Kubernetes
AI Analysis
Choose a Movie to Review
Click on any movie to view details and write reviews
The Shawshank Redemption
1994
Drama
Inception
2010
Sci-Fi
Interstellar
2014
Sci-Fi
Frontend
Backend
Database
Model
Critical System Failure
Not secure
52.66.123.50:3000
The Shawshank Redemption
1994
Drama
Inception
2010
Sci-Fi
Interstellar
2014
Sci-Fi
Fight Club
1999
Drama
Gladiator
2000
Action
Dark Knight
2008
Action
Latest Community Reviews
No reviews available yet. Be the first to share your thoughts!
Frontend
Backend
Database
Model
Critical System Failure