Versioning and tagging on source code and the artifacts using github and nexus.

We need to maintain different versions on source code to avoid unwanted changes and bugs in code and also we required to maintain versioning on artifact to quick revert back to the old working version.

Why artifact versioning if we already have source code versioning.

1. For quick revert back
2. For avoid build process multiple times
3. For version upgrade

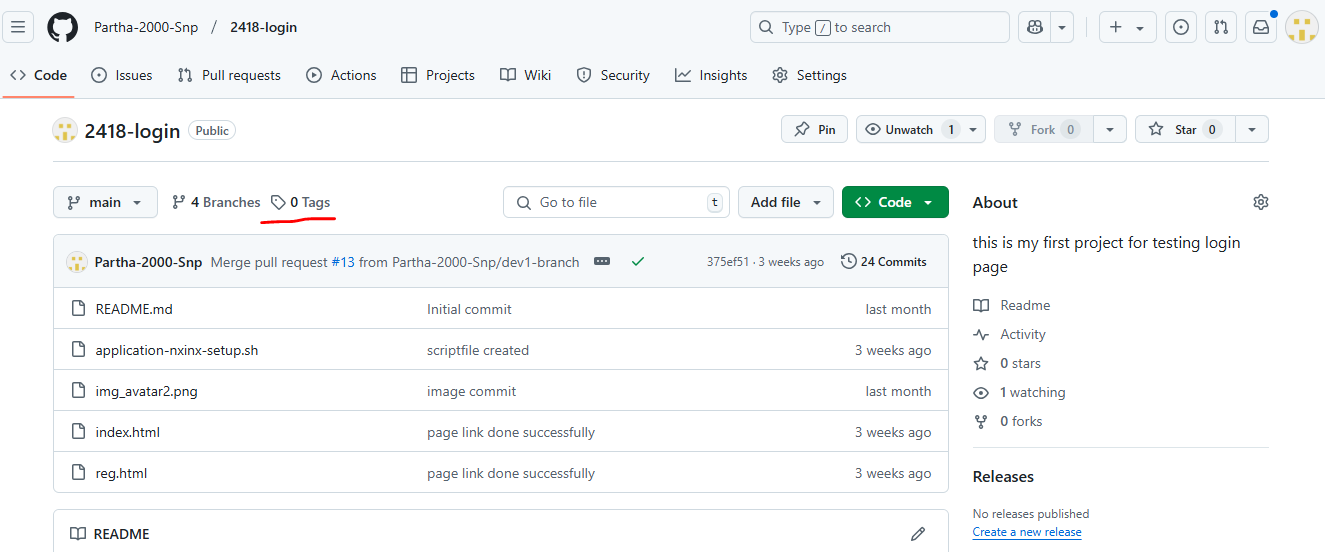
For code we can maintain version control from github.

This basically takes snapshot of targeted branch code and create a zip and tar format. Moving forward no more changes can be done on that only we can download that and use the code.

We will have a big advantage. As a developer we can just share the zip or tar file links to the devops team they can directly download it to their system and use it need to invite or clone process.

Step 1-

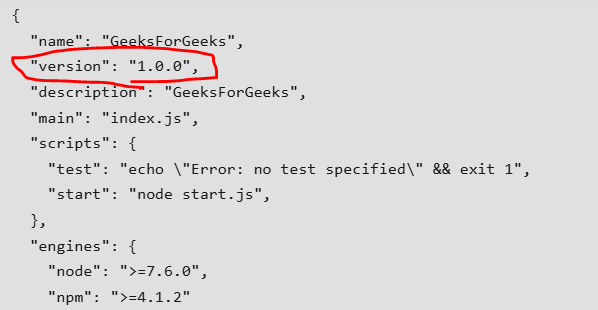
Go to git hub.



Here go for tags.

Note- for actual production application it will contains the metadata file for the project such as package.json. .pom files etc..

For those we can define the version on those metadata files for more visibility.

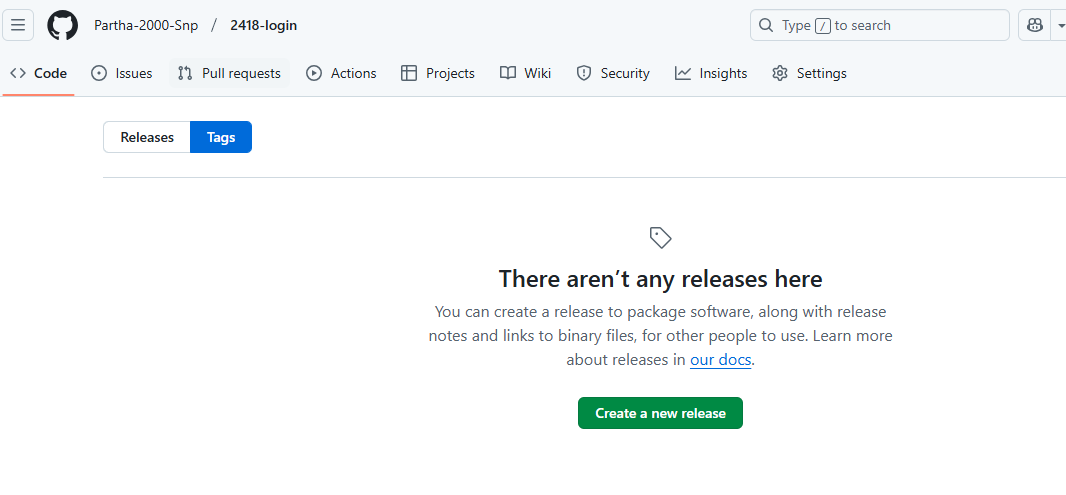


Step 2-

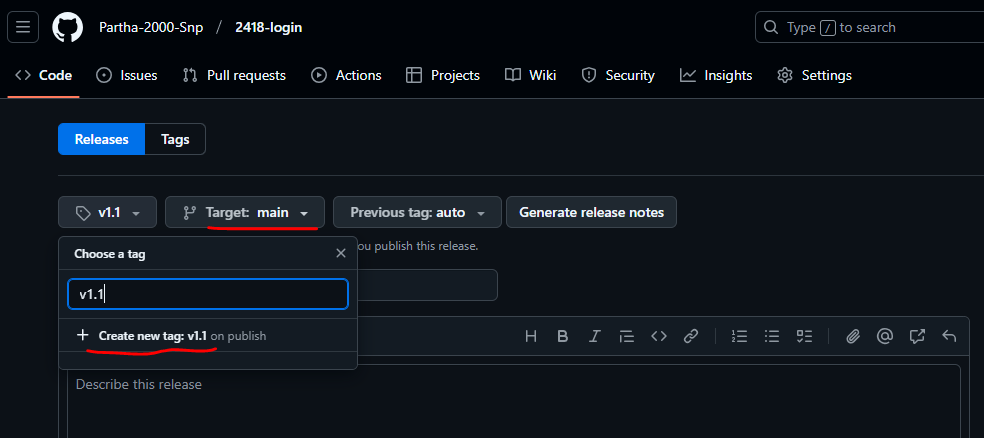
Go to targeted repository where do we want to assign tag.

Go for tag option.

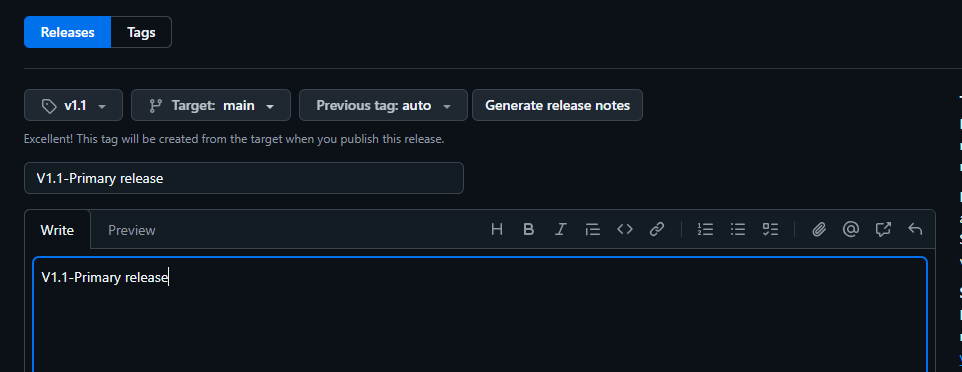
Go for create new tags.



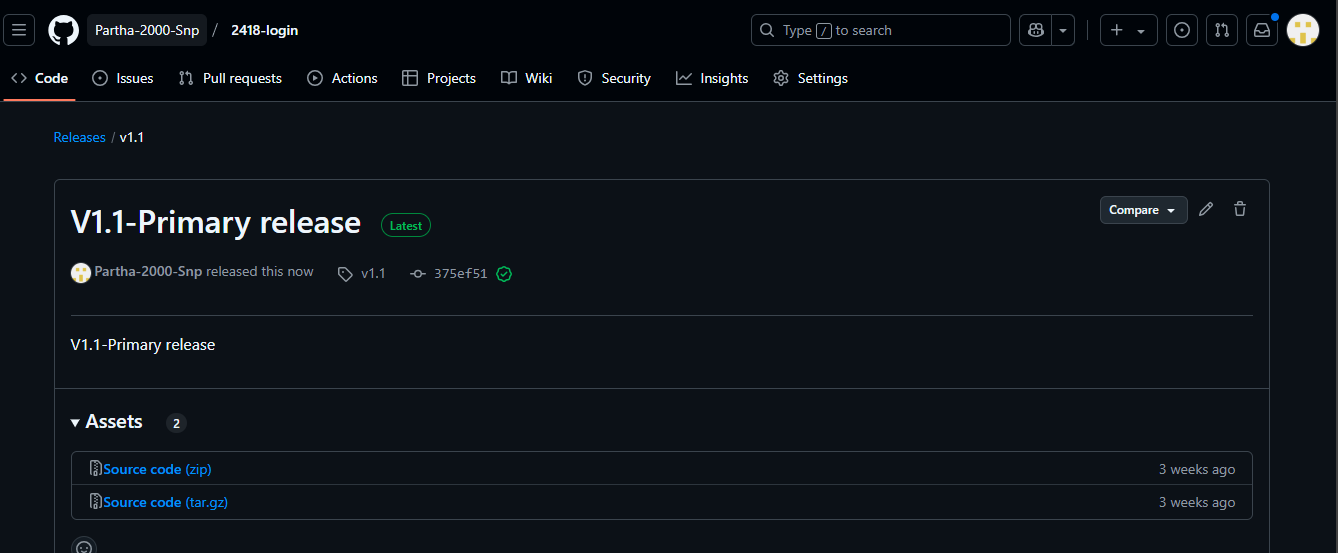
Type the tag name and choose to create and select branch.



Write the version name and the description about the version.



Go ahead and publish the release. Zip and tar format.



If you wat to deploy this version of the application we will provide those zip or tar file link to devops engineer to proceed further. So they can download it using “wget link” and work.

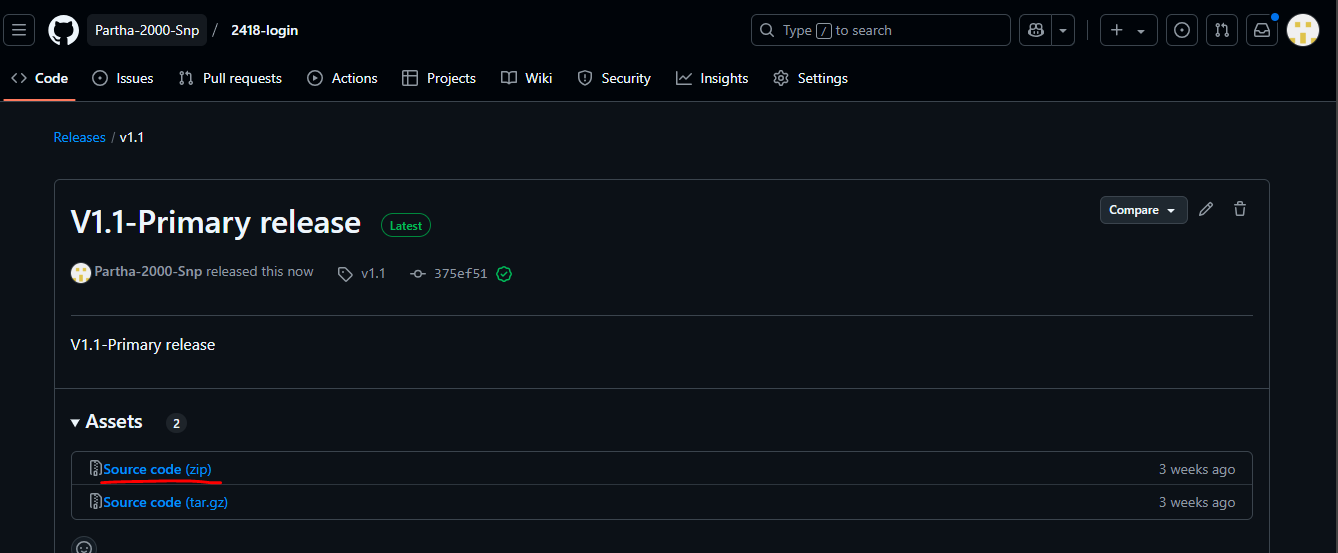
From here the devops team will get the code on build server and build the code to generate the artifact. Store the artifact to artifact library nexus, azure or any other third party.

From developer end again the changes done on code we will test it and assign another version to the code.

Let’s demonstrate.

We are providing the new version 1.1 zip link to developer

Right click on zip and copy the address.

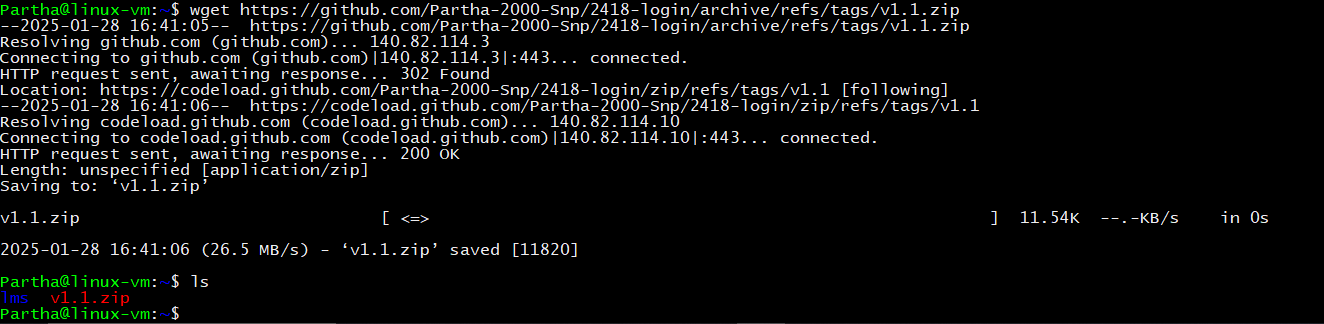


On devops side

>> wget link

Ex- wget <https://github.com/Partha-2000-Snp/2418-login/archive/refs/tags/v1.1.zip>

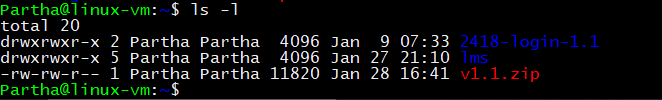
File will be downloaded



Let’s unzip it and deploy.

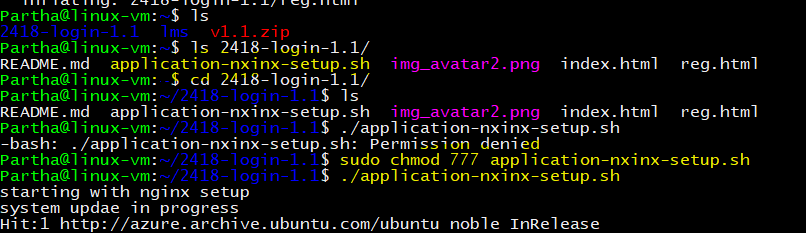
>> unzip zipped-file-name

Ex- unzip v1.1.zip

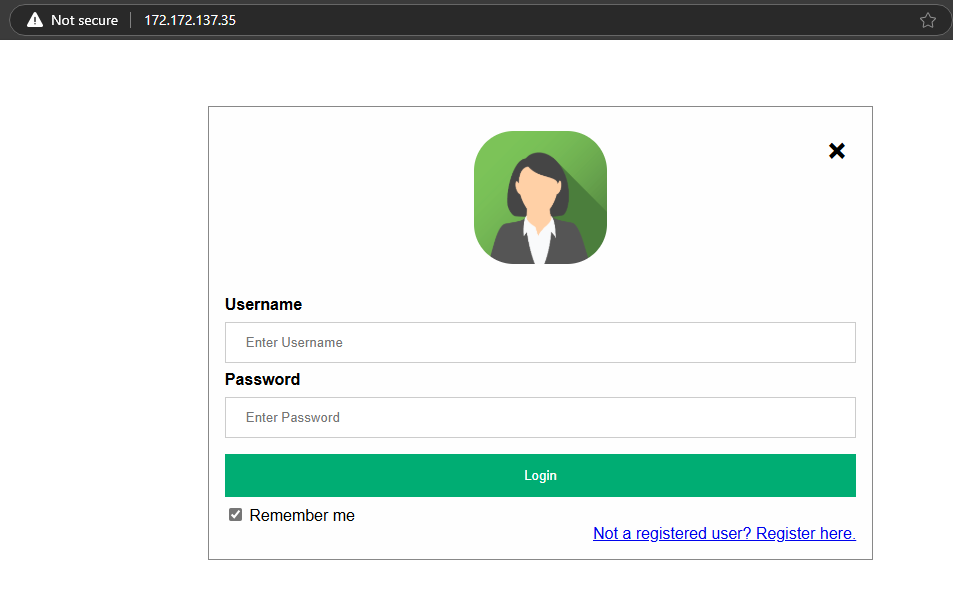


Let’s deploy this to nginx and test.

Go to unzipped folder and check for files.

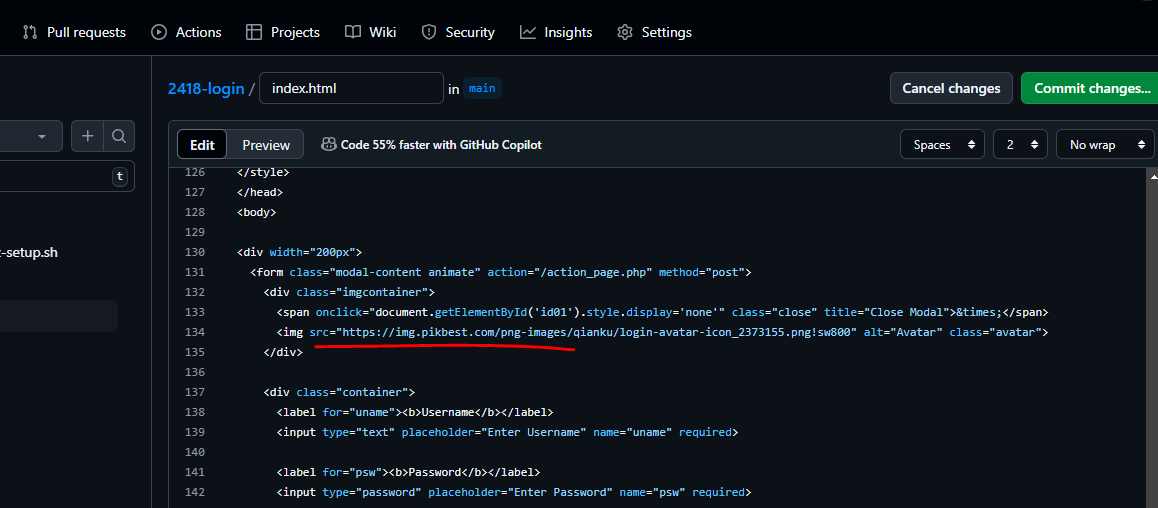


In my case I already had script file that can do all setup for application on nginx so I just run that and application setup was done.



Let’s do some other changes and go for next version. V-1.2

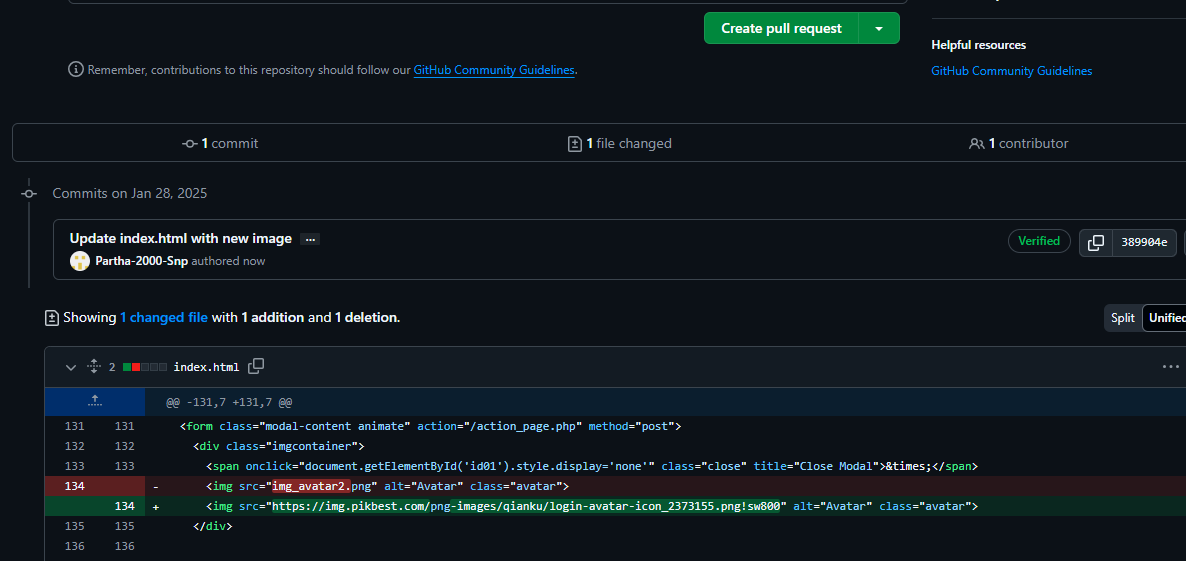
Updated the login page image.



Let’s commit it and take new version.

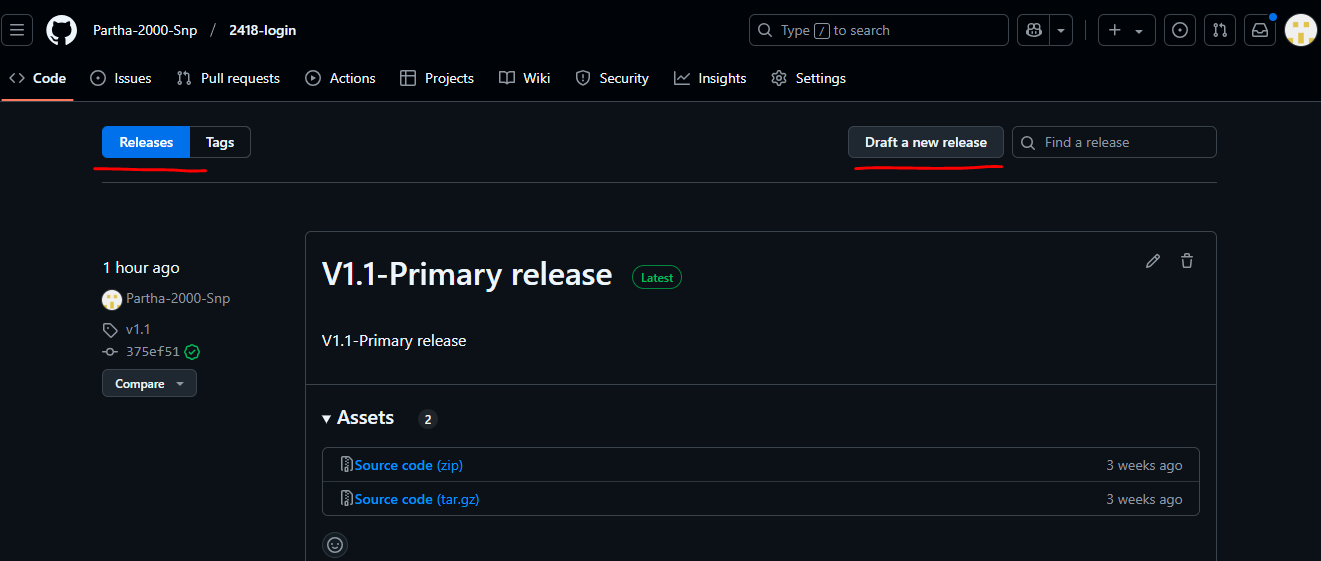
Committing to a new branch name as devops-branch.

Not going to have pull request and marge to main branch.

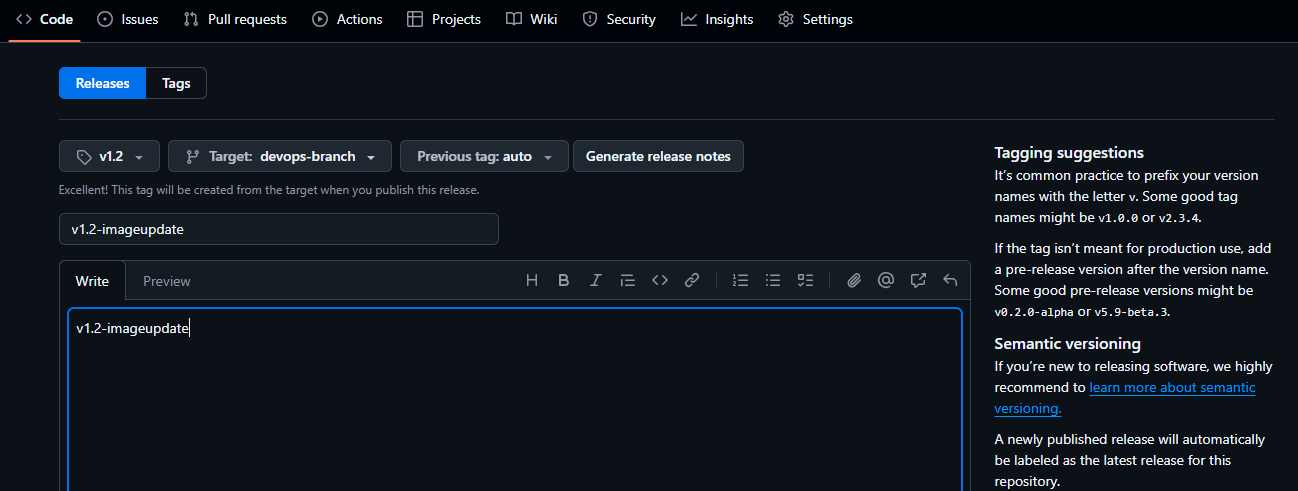


As we have flexibility to have tag on any branch I will assign tag on this changed branch directly.

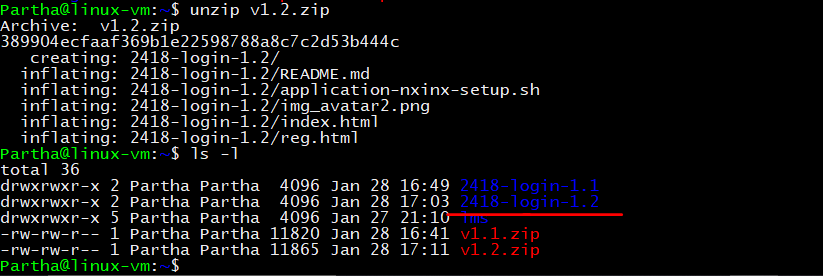
Go for tag and release then draft a new release.



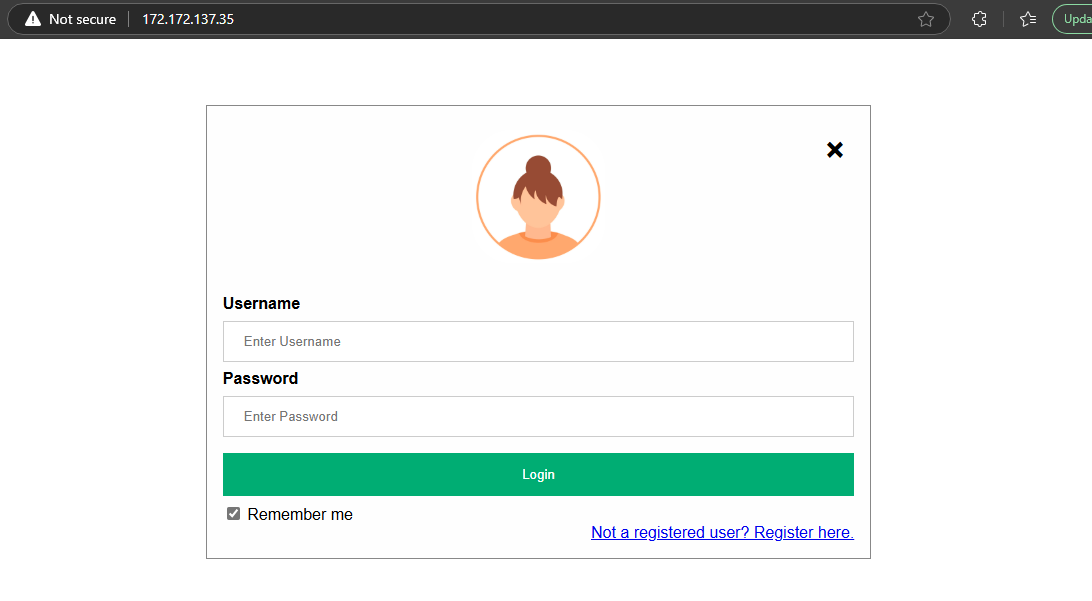
Here, we will select branch and name for the version.



Get this version 1.2 to server and test.



Run version 1.2



If you did not like it can revert back to previous version code easily as that code is not getting corrupted because by tagging we are taking snapshot of it and storing it safely.

In real time code will not be deployed directly as code because framework code need to build first to convert to binary artifact to run it properly.

Here in same way we will take LMS application assign tag for it on code take different versions to build server build it and run it.

Here we do need to store the build artifacts as version wise for easy use release update and quick revert back we will use nexus to do those all.

**Why Build is Needed**

1. **HTML/CSS/JavaScript**:
   * These applications are static.
   * No preprocessing is required because browsers can directly interpret HTML, CSS, and JavaScript files.
2. **Node.js Applications**:
   * Involves server-side code written in JavaScript (or TypeScript).
   * May need dependencies installed using npm install before running the application.
   * Example build steps:
     + Bundling assets (e.g., Webpack).
     + Transpiling ES6+ code or TypeScript to JavaScript.
     + Optimizing assets for production.
3. **Angular Applications**:
   * Involves TypeScript, SCSS, and other advanced tools.
   * Needs a build step to:
     + Compile TypeScript into JavaScript.
     + Minify JavaScript and CSS.
     + Bundle the application into files the browser can use.

**Basic Build Steps**

**1. Node.js Applications**

* **Install Dependencies**:

bash

CopyEdit

npm install

* **Build for Production** (if applicable):

bash

CopyEdit

npm run build

* **Start the Application**:

bash

CopyEdit

npm start

**2. Angular Applications**

* **Install Dependencies**:

bash

CopyEdit

npm install

* **Build the Application**:

bash

CopyEdit

ng build --prod

* This creates an optimized dist/ folder with all static files for deployment.

**3. React Applications (Example)**

* **Install Dependencies**:

bash

CopyEdit

npm install

* **Build the Application**:

bash

CopyEdit

npm run build

* This generates a build/ folder with all deployable files.

**Deploying After Build**

* After building the application, you deploy the generated folder (e.g., dist/ for Angular or build/ for React) to a web server like:
  + **Apache/Nginx** (for static files).
  + **Node.js runtime** (for server-side apps).
  + **Cloud platforms** (e.g., Azure, AWS, Netlify, Vercel).

If you want, I can guide you through a specific app setup or deployment process!