Hello Everyone!

Trainer: Sonal Mittal

Agenda: Day 1- 13 Apr 2025

- Overview on SDLC life cycle.
- Discuss on traditional Models and Agile methodologies.
- Discuss on DevOps process and principles.
- DevOps CICD pipeline and tools.
- DevOps and DevSecOps.

==========
==========

Application:
- A few lines of code written to provide a functionality to end user

Types of Application:

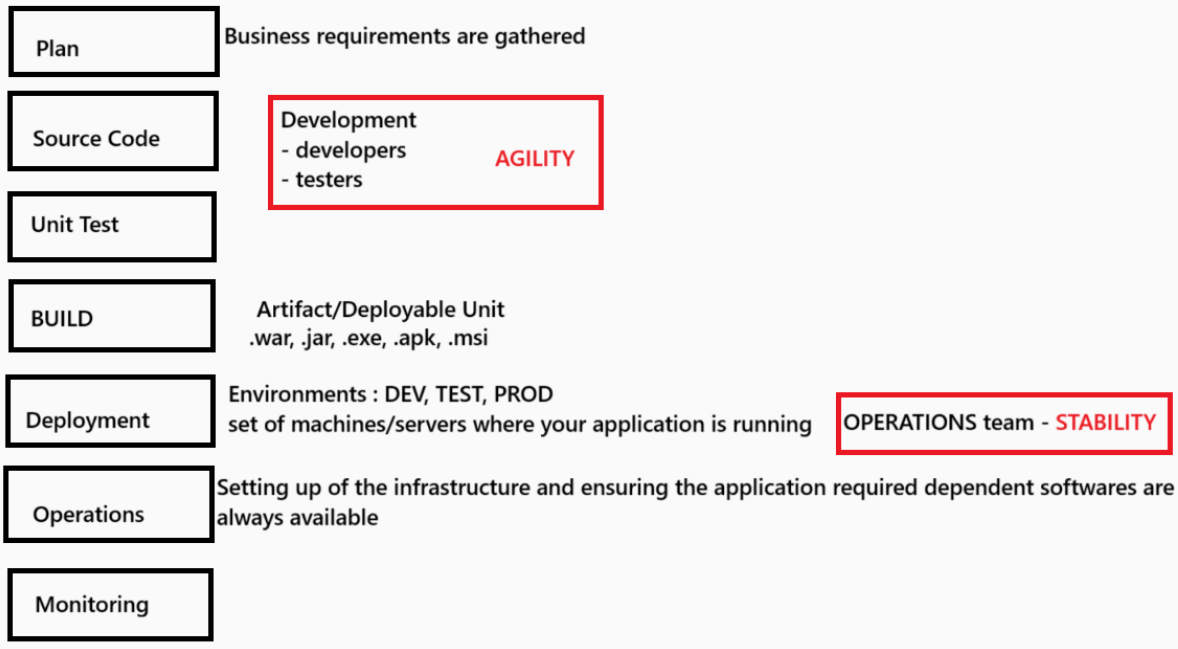
1. Standalone Applications:
- Software programs/Applications that are installed and run on our desktop
- software that runs on local computer OS, not through web browser

2. Web Applications: Application that can be accessed from any location, from any devices using internet

3. Enterprise Application:

- Authentication and Authorization
- Dynamic Application
- High Availability
- Persistent data
- Interactive and send notifications

SDLC phases



AGILE / Iterative Model

Agile team: Product owner, Scrum master, developer, tester
User stories: requirements to be developed and Build

Iteration: time frame of 2 to 4 weeks

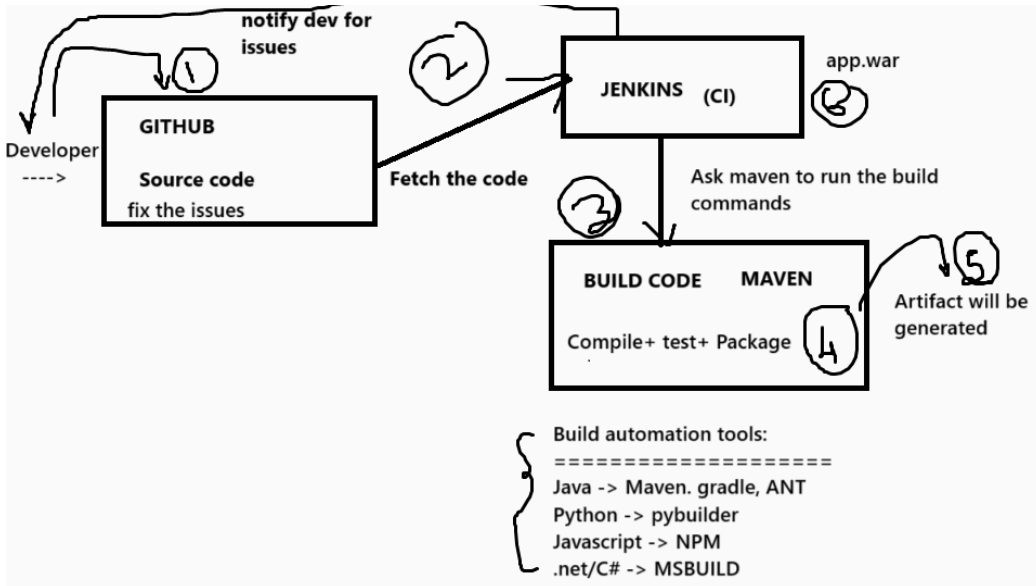
SPRINT 1: Login: Code > test > Build | demo
SPRINT 2: logout: code > test > build | demo | Deploy

Communication between development

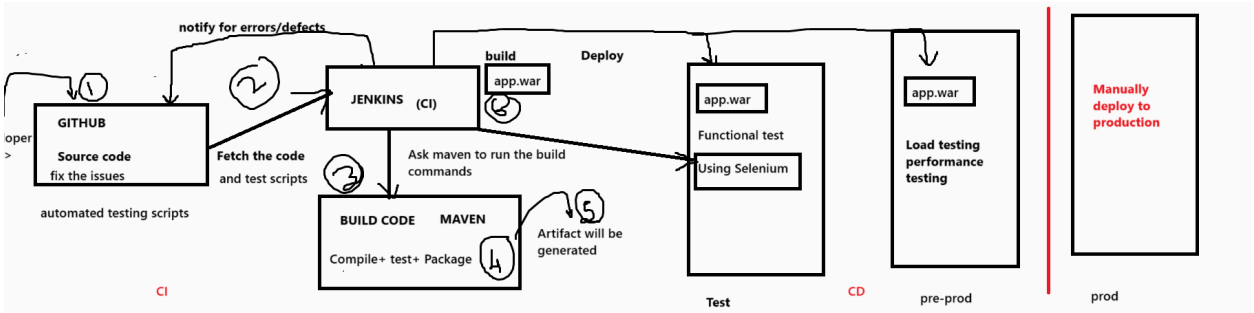
Disadvantages :

- No communication with operations team, time to fix the deployment issues is more
- Deployment is done by operation team on a given date and time
- Deployment is done manually, no automation tool is used by team
- Deployment may get delayed in case of failures during deployment
- no tools used to scale up the infrastructure

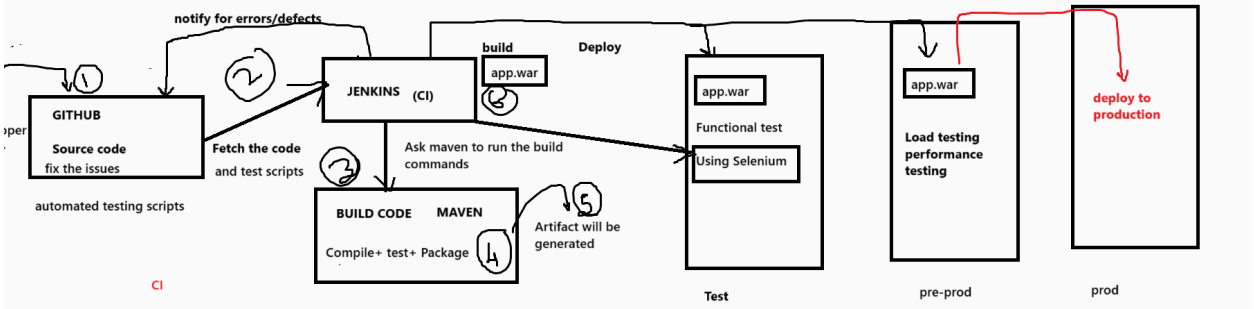
SDLC Phases	DEVOPS PRINCIPLES	TOOLS
SOURCE CODE Phase : --->	1. Continuous Version Control	GIT, GITHUB, BITBUCKET, GITLAB, TFS
BUILD PHASE ----------> (Continuous Test + Continuous Build)	2. Continuous Integration	Jenkins, GitHub Actions, GITLAB CICD, Travis , bamboo
Deployment Phase : ------>	3. Continuous Delivery	Docker, Ansible
Deployment Phase : ------>	4. Continuous Deployment	Kubernetes , Ansible
OPERATION PHASE -------->	5. Configuration management	Ansible, Terraform, puppet, saltstack, chef
MAINTAINENCE PHASE ----->	6. Continuous Monitoring	Prometheus, Grafana, ELK, Nagios, splunk
7. Communication and collaboration		



Build automation tools:
==========
Java -> Maven. gradle, ANT
Python -> pybuilder
Javascript -> NPM
.net/C# -> MSBUILD

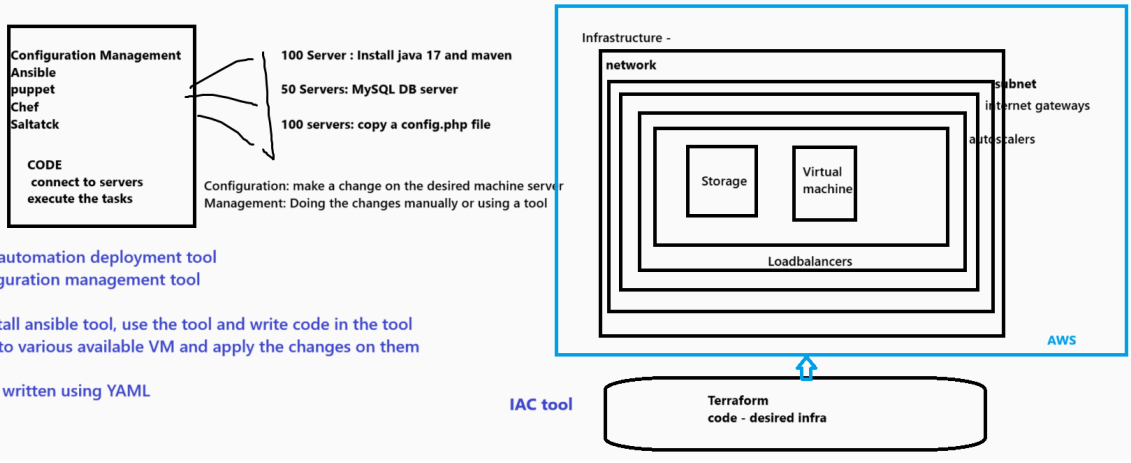


Continuous Delivery: We are using an automation tool to deploy the code on Test and Pre-prod environment but application is deployed ona give date and time manually into production env



Continuous Delivery: We are using an automation tool to deploy the code on Test and Pre-prod environment but application is deployed ona give date and time manually into production env

CONITNUOUS DEPLOYMENT : using an automation tool deploy the code in test--> pre-prod --> also into production



Ansible : A automation deployment tool
It is a configuration management tool

You will install ansible tool, use the tool and write code in the tool to connect to various available VM and apply the changes on them

The code is written using YAML

DEVSECOPS

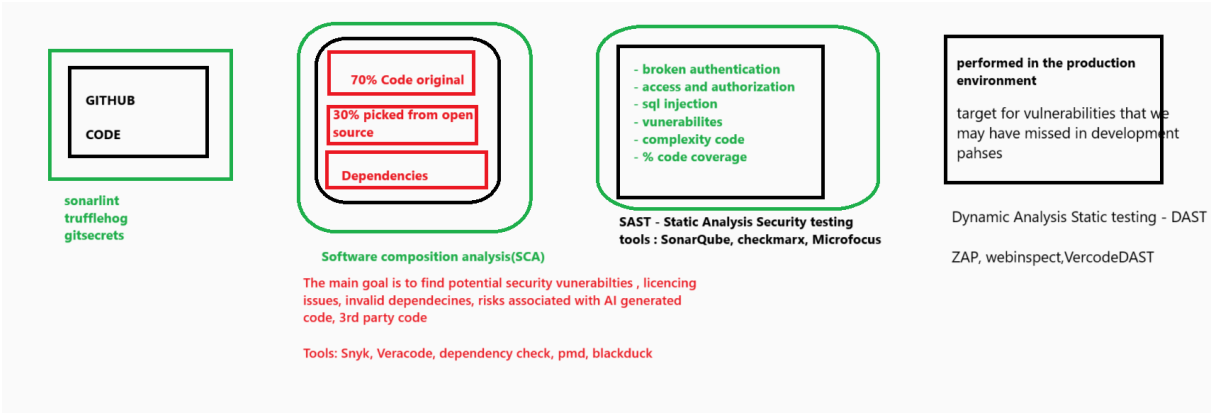
==========

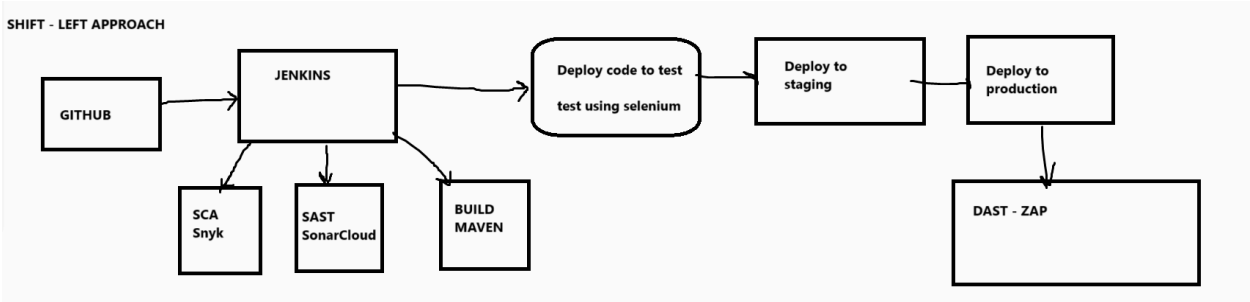
DEVSECOPS : DEVLOPMENT + SECURITY TESTING + OPERATIONS

Security testing :
- Type of testing in which we check if the application is vulnerable to cyber attacks.
- It is a non -functional testing which will provide us evidences if the application is safe and reliable.
- Focuses on : finding out systems or infrastructure that can exploited by attackers
- Identify risks and issues that can have negative impact on our application
- By looking at security testing result, we can fix these issues/vulerabilties

DEVOPS - main focus is to include automation tools at every stage of SDLC
DEVSECOPS -> main focus is to include security testing at every stage of SDLC

In DEVSECOPS we integrate security testing tools with DEVOPS tools





DAY TO DAY ACTIVITIES OF DEVOPS ENGG

Make sure that the pipeline is running smoothly—This is one of the most important tasks of a DevOps engineer to make sure that CI/CD pipeline is intact and fixing any issue or failure with it is the #1 priority for the day. They often need to spend time troubleshooting, analysing and providing fixes to issues.

Interaction with other teams—Coordination and collaboration is the key for DevOps to be successful and hence daily integration with Dev and QA team, Program Management, IT is always required.

Work on Automation Backlog—Automation is the soul of DevOps so DevOps engineering need to plan it out and I can see DevOps engineer spending lots of time behind the keyboard working on Automating stuff on a daily basis.

Infrastructure Management—DevOps engineers are also responsible for maintaining and managing the infrastructure required for CI/CD pipeline and making sure that it’s up and running and being used optimally is also part of their daily schedule. Working on Backup, High Availability, New Platform setup etc.

Dealing with Legacy stuff—Not everyone is lucky to work on latest and newest things and DevOps engineers are no exception hence they also need to spend time on legacy i.e., in terms of supporting it or migrating to the latest.

Exploration—DevOps leverage a lot from the various tools which are available, there are many options as open source, so the team needs to regularly check on this to make sure the adoptions as required, this is something that also requires some effort not on a daily but regular basis. What are open-source options available to keep the cost at a minimum?

Removing bottleneck—DevOps's primary purpose is to identify the bottlenecks / Manual handshakes and work with everyone involved (Dev / QA and all other stakeholders) to remove them so the team spend a good amount of time in finding such things and build the Automation Backlog using this. How we can get builds faster?

Documentation—Though Agile / DevOps stresses less on the documentation, it is still the important one that DevOps engineer does on daily basis, Be it Server Information, Daily Week charted, Scrum / Kanban board, or Simple steps to configure / backup or modify the infrastructure, you need to spent a good amount of time in coming up these artifacts.

Training and Self Development—Self-learning and Training are very useful in getting a better understanding and many organizations encourage their employee to take the time out

and do some of these and same holds true for DevOps folks as well, So learn something new everyday…

Continuous Improvement as Practice—Last but not least, It's up to the DevOps folks to build awareness of the potential of CI/CD and DevOps practices and building a culture of leveraging it for doing things better, reducing re-work, increasing productivity and optimizing the use of existing resources.

==========

Reference to more notes:

https://github.com/Sonal0409/PGP-DevOps-Cohot1-August/tree/main/DevOpsFundamentals

==========

Day 2: 19-Apr-2025

==========

Agenda:

==========

> Connect to SL Lab for devops

> Practice the pre-required Linux commands on DevOps SL lab

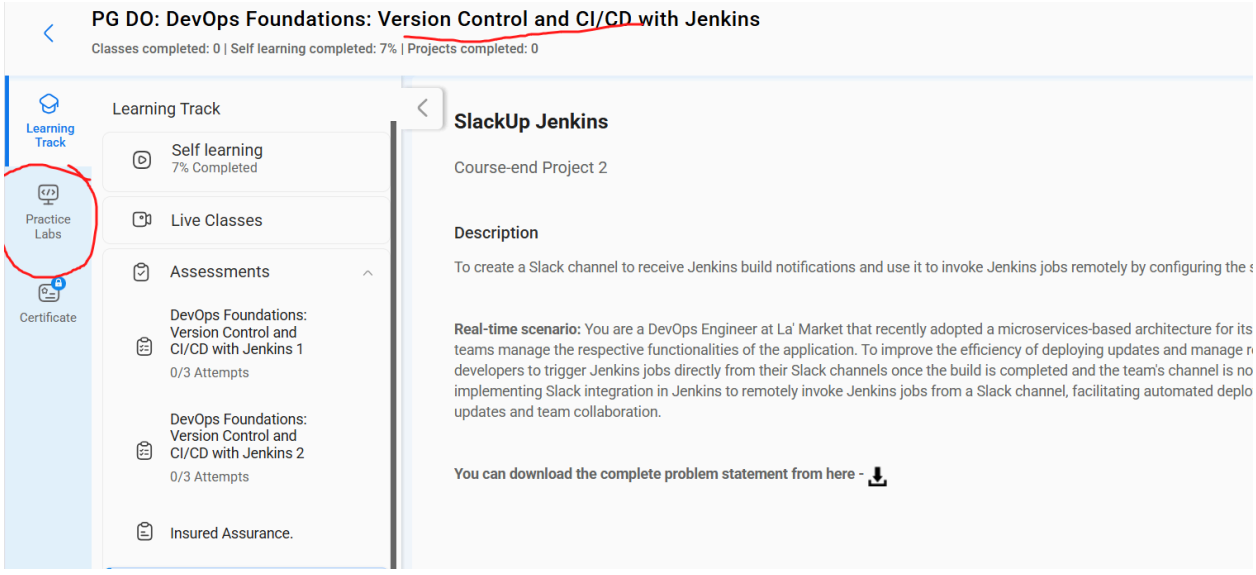
> Continuous Version Control

==========

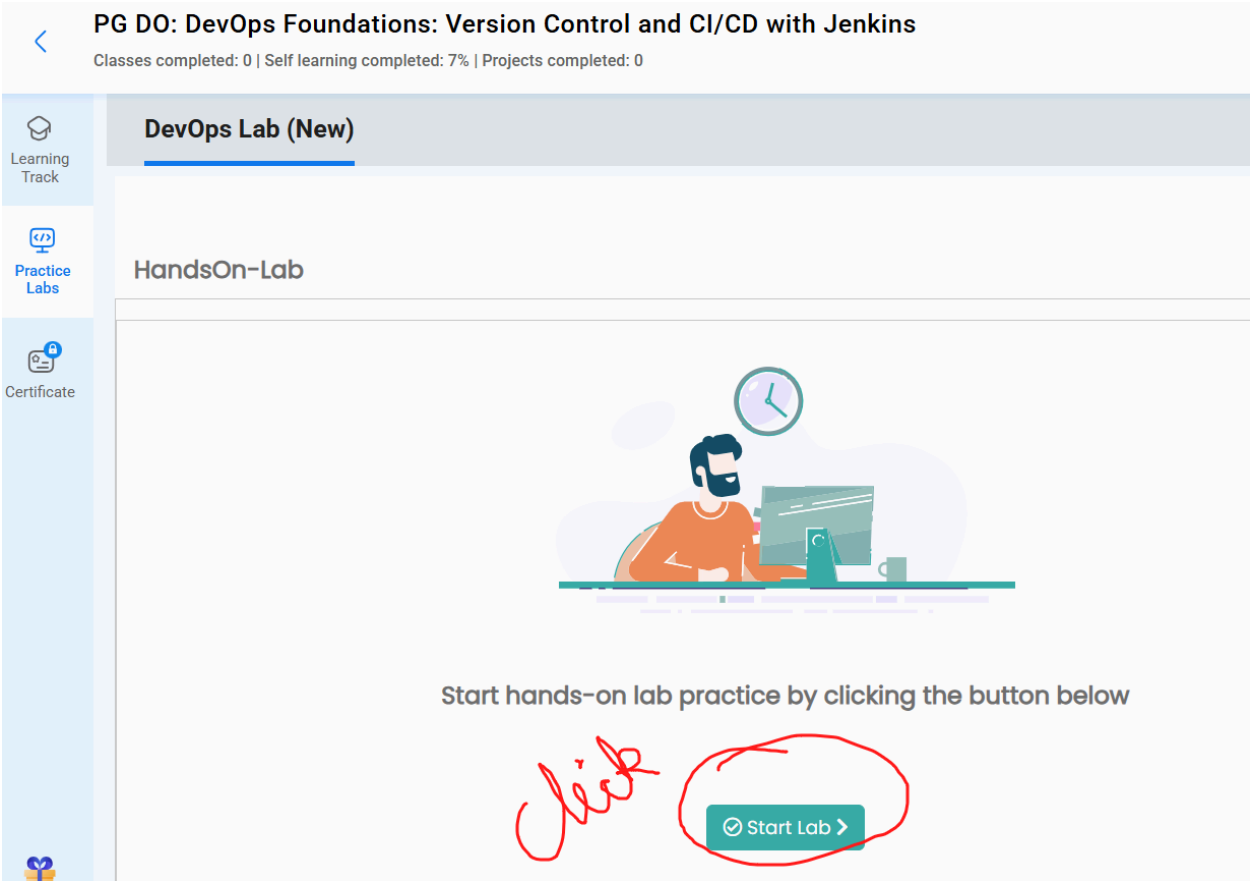
Connect to SL Lab on LMS:

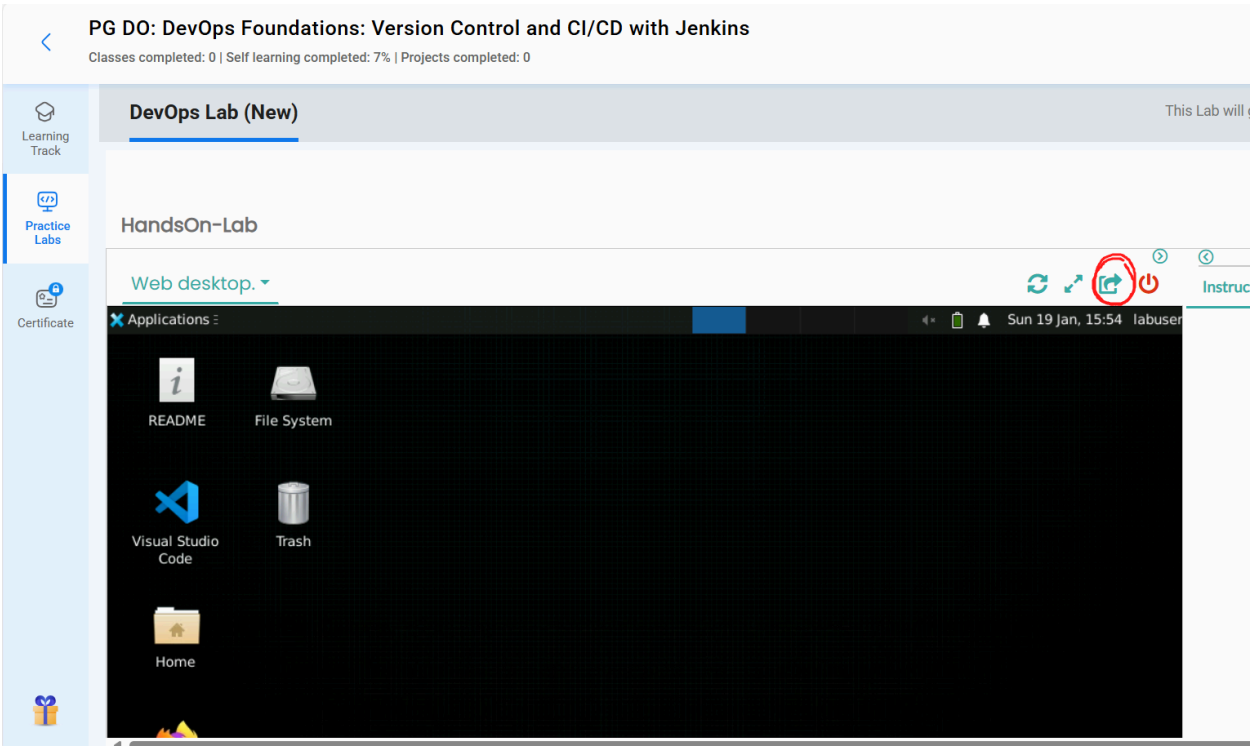
Note: For any lab related issues or concerns send your email-id and issue to LSM in the chat box immediately

Connect to SL Lab on LMS




PG DO: DevOps Foundations: Version Control and CI/CD with Jenkins
Classes completed: 0 | Self learning completed: 7% | Projects completed: 0
Join Community
Notes
Help
Learning Track
Practice Labs
Certificate
DevOps Lab (New)
This is the new DevOps Lab
Talking:
Your Labs are ready.
Launch lab
You are screen sharing
EN
Stop share


PG DO: DevOps Foundations: Version Control and CI/CD with Jenkins
Classes completed: 0 | Self learning completed: 7% | Projects completed: 0
Learning Track
Practice Labs
Certificate
DevOps Lab (New)
HandsOn-Lab
Start hands-on lab practice by clicking the button below
Start Lab



LINUX Commands

> Click on the Terminal button [it will be a square black button].
> Terminal application will open.

> Linux commands are case sensitive.

1. To check the OS

   # **uname**

2. Check the OS distribution and its version

   # **cat /etc/os-release**

3. Clean up the screen

   # clear

4. Check all the commands executed
   # history

5. Become a Root user

   # sudo su -

6. Create empty directory:

   # mkdir myfiles

7. List the directory :

   # ls

8. Go inside the directory:

   # cd myfiles

9. Path of directory or print the working directory

   # pwd

10. Create a empty file in the directory

    # touch file1

    # touch file2

11. List all the files in the directory

    # ls

12. Add content to the file

    # echo "this is the content to file" > file1

13. Check the content

    # cat file1

14. Append more content in the file using echo command

echo "this is the next line into file" >> file1

cat file1

Both the lines will be there.

12. Copy content of 1 file to another

cp file1 file2

cat file1

cat file2

USe the linux editors to add /modify/delete content

Vim editor:
==========

Create a file using vim editor:

vim file1

Press i

Enter required data/lines

To save the data and come out of the file

Press ESC key on your keyboard

Then enter
:wq

Press enter key

==========
Just for your information:
==========
h: Move the cursor to left by one position.
I: Move the cursor to right by one position.

j: Move the cursor to downward direction by one line.
k: Move the cursor to upward direction by one line.
You can also use your keyboard arrow keys instead of the letters.

NANO editor
==========

Create a file using nano editor

 # nano file4

Inside the file, you can directly write into the file

Then to save the modifications,
Press ctl x
Press y

Enter key
Comout of the file and save the file

Juts for your information:

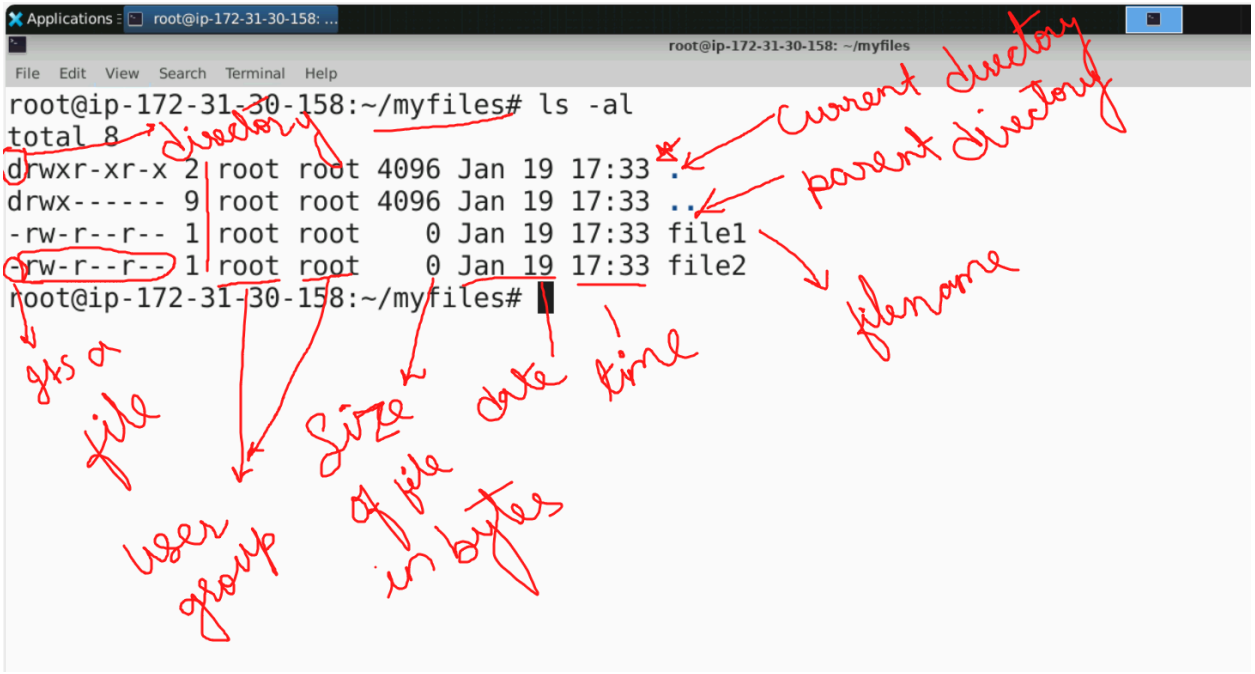
Nano Editor
Navigation shortcuts for Nano editor are :

Ctrl+F: move the cursor forward.
Ctrl+B: move the cursor backward.
Ctrl+Space: move forward one word.
Alt+Space: move backward one word.
Ctrl+V: move to the next page.
Ctrl+Y: move to the previous page.
Ctrl+N: move to the next line.
Ctrl+P: move to the previous line

==========

List more details about file and directory

 # ls -al



On a file you will see 3 types of permissions:

r : read
w: write
x: execute
- : no permission

There are 3 types of users on the file and permissions are given to them

u: User who created the file

g: the group members of the user that created the file

O: All the other users in the linux system

a : for all users+ group + other user

 # chmod g+w file1

 # ls -al

Remove permission:

 # chmod g-w file1

Add write permission to user and group user for file1

 # chmod ug+w file1

2nd method : octal format

==========

Permission are represented as a number

Read : r -> 4
Write: w -> 2
Execute : x -> 1
No permission : 0

Suppose we want to give

Read and write to user => 4+2 = 6
Read to group -> 4
No permission to others -> 0

 # chmod 640 file1

 # ls -al

 # chmod 777 file1 -> all permissions to all

 # ls -al

 # chmod 400 file1 -> only read permission to the user

 # ls -al

ADD and Remove packages on Ubuntu:
==========

 # sudo su -

 # apt-get update

 # apt-get install tree

 # apt-get remove tree -y

==========
Remove a file:
==========

rm file1

Remove a directory

rm -rf <dir_name>

==========

LINUX Practice Exercises

https://github.com/Sonal0409/Linux-Fundamentals.git

Commands used commonly while learning DevOps tool
==========

1. "ls": List files and directories in the current directory.
 - Example: "ls -l" (long listing format with detailed information)
2. "cd": Change directory.
 - Example: "cd /path/to/directory"
3. "pwd": Print working directory, displays the current directory path.
4. "mkdir": Make directory, creates a new directory.
 - Example: "mkdir new_directory"
5. "rm": Remove files or directories.
 - Example: "rm file.txt" (remove a file)
6. "cp": Copy files or directories.
 - Example: "cp file.txt /path/to/destination" (copy file to another location)
7. "mv": Move or rename files or directories.
 - Example: "mv file.txt /path/to/destination" (move file to another location)
8. "touch": Create an empty file or update file timestamps.
 - Example: "touch file.txt" (create a new file)
9. "cat": Concatenate and display file content.
 - Example: "cat file.txt"
10. "grep": Search for a pattern in a file or output.

- Example: "grep "pattern" file.txt"
11. "find": Search for files and directories within a specified path.
- Example: "find /path/to/search -name "file.txt""
12. "chmod": Change file permissions.
- Example: "chmod 755 file.txt" (gives read, write, execute permissions to the owner, and read/execute to others)
13. "chown": Change file owner and group.
- Example: "chown user:group file.txt"
14. "ps": Display information about running processes.
- Example: "ps aux"
15. "top": Monitor system processes in real-time.
16. "kill": Terminate processes by their PID (Process ID).
- Example: "kill PID" (replace PID with the actual process ID)
17. "tar": Archive files together.
- Example: "tar -czvf archive.tar.gz /path/to/directory" (create a gzip-compressed tar archive)
18. "wget": Download files from the web via command line.
- Example: "wget https://example.com/file.txt"
19. "curl": Transfer data to or from a server using various protocols.
- Example: "curl -O https://example.com/file.txt"
20. "df": Display disk space usage.
- Example: "df -h" (show usage in human-readable format)
21. "du": Estimate file and directory space usage.
- Example: "du -h file.txt" (show usage in human-readable format)
22. "free": Display memory usage.
- Example: "free -h" (show memory in human-readable format)
23. "ifconfig": Configure network interfaces (Note: Replaced by "ip" command in some distributions).
- Example: "ifconfig -a" (show all network interfaces)
24. "ping": Send ICMP Echo Request packets to a host.
- Example: "ping google.com"
25. "ssh": Secure Shell, provides a secure remote connection to a server.
- Example: "ssh user@hostname" (replace "user" and "hostname" with appropriate values)
26. "scp": Securely copy files between hosts using SSH.
- Example: "scp file.txt user@hostname:/path/to/destination"
27. "rsync": Efficiently synchronize files and directories between locations.
- Example: "rsync -av /path/source/ /path/destination"
28. "netstat": Network statistics and connections.
- Example: "netstat -tulnp" (show all listening ports and associated PIDs)
29. "nc": Netcat, a versatile networking utility for reading/writing data across TCP/UDP connections.
- Example: "nc -vz example.com 80" (check if port 80 is reachable)
30. "systemctl": Control system services (systemd-based systems).
- Example: "systemctl start service_name" (start a service)
31. "journalctl": Query and view logs from the systemd journal.

- Example: "journalctl -u service_name" (view logs for a specific service)
32. "cron": Schedule periodic tasks using cron jobs.
- Example: "crontab -e" (edit the user's cron jobs)
33. "at": Schedule a one-time task to be executed later.
- Example: "at now + 1 hour" (execute a command one hour from now)
34. "sed": Stream editor for filtering and transforming text.
- Example: "sed 's/pattern/replacement/g' file.txt" (replace all occurrences of "pattern" with "replacement")
35. "awk": Text processing tool for extracting and manipulating data in files.
- Example: "awk '{print $1}' file.txt" (print the first column of a file)
36. "sort": Sort lines of text files.
- Example: "sort file.txt"
37. "uniq": Report or omit repeated lines in a file.
- Example: "uniq file.txt"
38. "tar": Archive files together.
- Example: "tar -czvf archive.tar.gz /path/to/directory" (create a gzip-compressed tar archive)
39. "wget": Download files from the web via the command line.
- Example: "wget https://example.com/file.txt"
40. "curl": Transfer data to or from a server using various protocols.
- Example: "curl -O https://example.com/file.txt"
41. "df": Display disk space usage.
- Example: "df -h" (show usage in human-readable format)

Practice exercises on Linux:

https://github.com/Sonal0409/Linux-Fundamentals.git

==========

Version Control

==========

Verison is tracking of changes that has been done to a document or set of files
Tracking can be done manually or using a tool

- Version control is a tool or a system that will document/track the chnages that has been done to a file or set of files
- Version allows multiple people to work on the same set of information
- Allows us to have a snapshot of entire project

Why version Control tool is important??

1. Allows to keep history or backup the data of our files
2. Multiple team members can collaborate at common location(repository)
3. Track who did the change, when the change, where the change was done.

Benefits of Version Control System:
==========

- Restore to previous versions
- Understand what happened
- Backup of the files
- collaborate
- Store version of files
- If we have a bug we can find out what and when the code screwed up.

Version control System:

==========

1. Local version control System -

RCS (Revision control system)

- a practice of having VC tool in the local machine of the developer/user
- local VC tool will keep record of the changes made by the developer to the files.

Issuse:

- Multiple people cannot access those files and cannot collaborate

- if the machine is not available, the actual file and VC files will be deleted.

2. Centralized version Control system -
Example: SVN, perforce, drive, TFS

- In CVS a cebtral repository is maintained where all the versioned files are kept
- users can checkin and checkout files from different machines

Issues:
- Dependent on internet
- In case the central server fails, whole system goes down.

3. Distributed Version control System
==========
Local Repository:
==========
It is a location/folder within the local machine of the user/developer
where he will maintain versions of code files.

==========

Role of VCS in DevOps lifecycle

==========

- Planning stage - requirement documents can be version controlled.

- Design phase - Design documents, blueprints, solution documents can be version controlled

- development phase - source code, data, properties files, DB files, unit test cases can be version controlled

- Testing phase - test data documents, reports, Automation scripts can be version controlled

- Deployment phase - scripts, deployment commands, docker, Kubernetes manifest files can be version controlled

- Operations - Infrastructure scripts, playbooks(Ansible code is written in playbooks) can be version controlled

==========

Best practices for version control

==========

- Commit the code granularly
- commit with a clear message
- provide git configurations
- Utilize the concept of branching
- Always have the latest copy of the project code in your local
- set up roles, teams and give proper access to the repositories with team members
- Approval process has to be set
- Backup your repositories to data loss.
- git squashing, to combine the commits

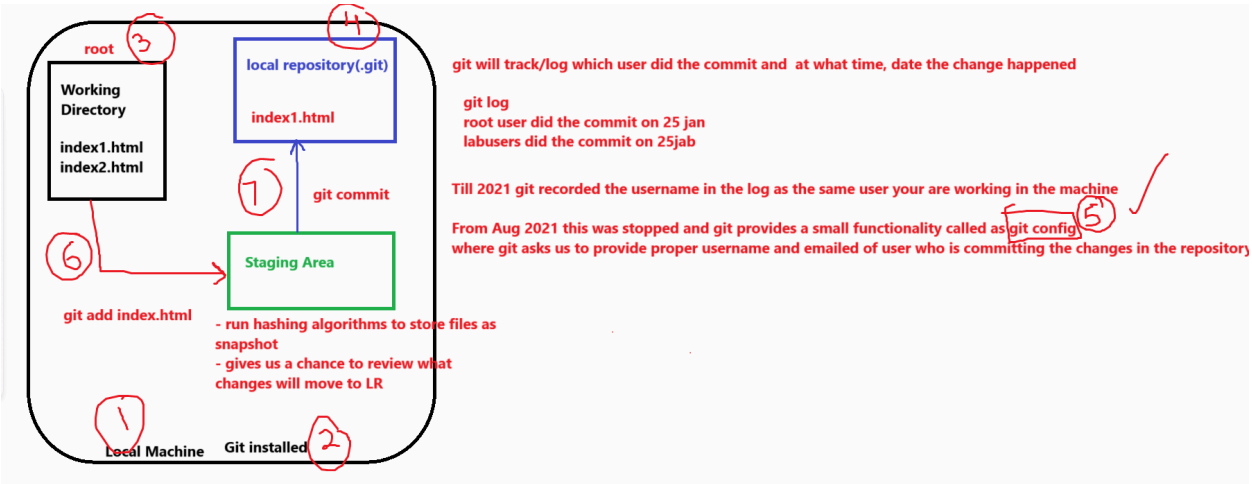
Git

===

- is a version control engine used for local version control
- it is an open source tool,, available for free
- available for all the OS

- GIT is used as a command line interface tool
- git comes with a GUI on windows OS
- Development IDEs come integrated with git and github
- Revert, branching, merging, stashing, rebase, tagging, commits etc.

How git works- how can we version control files using git :



Day 3: 20-Apr-2025

==========

Agenda:

> GIT workflows for a new file
> Git configurations
> GIt commit operations
> git workflow for a modified file
> git logs
> Git restore revert and reset operation

==========

Demo 1: Create a working directory and a local repository

==========

Switch to root user:

sudo su -

Step 1: Create a working directory

mkdir myproject

cd myproject

Step 2: Create some files in the working directory that we have to version control

touch index1.html index2.html

Step 3: Create a local repository:

git init

ls -al

.git folder is the local repository

==========

Demo 2: Configure the user and its email id which is going to commit into this local repo

GIT provides a simple git config tool that required 2 parameters

user.name
user.email

This configuration is not for authentication or permission
This is just for git log to track which user has commited into the repository

git config --global user.name admin

git config --global user.email admin@gmail.com

Remove the configuration

git config --global --unset user.name

Add the configuration again

git config --global user.name admin123

Demo 3: Verison control the new files in Local repo

git status

add the untracked file to staging area

git add index1.html index2.html

check the status of the files

git status

One file is staged and other is unstaged (untracked)

Commit the staged files

git commit -m "added file index1.html and index2.html"

Check the git log for commit details and config details

git log

Check the files in the local repo
git ls-files

==========

Demo 4:
Take an existing file which is already tracked by git
Make some modification on one of the files

Now check the status of the file

Check the files that you have in Local repo

git ls-files

Use the echo command to add the content to the file

echo "added content to the file" > index1.html

git status

File will be with status as modified

Add and commit all the modification

git commit -a -m "added modifications to the file"

==========

Demo 5:
How can we find the difference between the previous verison fo the file and the new version of the file

Take an exisitng file -> index1.html

vim index1.html

add some new content

Save the file --> press ESC key --> press :wq --> press enter key

git status

git diff index1.html

Discard the changes that are in working directory

git restore index1.html

All the new changes will be discarded from index1.html

==========

Again make changes in to the index1.html

vim index1.html

add some new content

Save the file --> press ESC key --> press :wq --> press enter key

Check the status

git status

Stage the changes
git add .

See the difference between changes that have been staged

git diff --staged index1.html

Move back the changes to unstage area or working directory

git restore --staged index1.html

Discard the changes from working directory

git restore index1.html
==========

Revert Command in GIT
==========

Git gives us an Operation called as Revert using which
one can revert the changes done in a commit ID to it previous version

In case of revert operation
we are making changes on the local repository by reverting to the previous version of the file
→ so revert command will always generate a new commit ID

Revert command is performed on a single commit id

We can revert a single commit at a time

As soon as revert command is executed
git will open an nano editor for us to give the reason/message why are we reverting this commit.

As soon as we save the editor , Git will also generate a new commit id for the revert operation.

Demo:
==========

git status

You working tree should be clean

git ls-files

Delete a file that is in local repo and working directory

Git command to delete a file from LR and WD

git rm index1.html

git status

git commit -m "deleted file"

git log --oneline

git revert <commitID>

Add message in the nano editor on first line and save the file

Save on nano editor
Press CTL x
Press y
Press enter key

A new commit will be generated for the revert operation

git log --oneline

File is back

ls

git ls-files

==========
GIT reset:
==========

Soft:
==========

echo "add content" >> index1.html

git add .

git commit -m "done modification1"

```
# echo "add content again " >> index1.html

# git add .

# git commit -m "done modification2"

# echo "add content again 123 " >> index1.html


# git add .

# git commit -m "done modification3"


# git log --oneline


# git reset --soft <commitID>
```

--soft:

Observation:
> In the git log, the commit history is reset -> all the commits which are above the given <commitID> get deleted
> Now the HEAD is the <Commitid> given in the command
> All the changes of the deleted commits will be back to the staging area.

git commit -m "all modifications together"

==========

--mixed

git reset --mixed <commit id>

Observation:
> In the git log, the commit history is reset -> all the commits which are above the given <commitID> get deleted

> Now the HEAD is the <Commitid> given in the command
> All the changes of the deleted commits will be back to working directory

git add .

git commit -m "all modifications together"

git status

==========

--hard

==========

Observation:

> In the git log, the commit history is reset -> all the commits which are above the given <commitID> get deleted
> Now the HEAD is the <Commitid> given in the command
> All the changes of the deleted commits will also be permanently deleted

git log --oneline

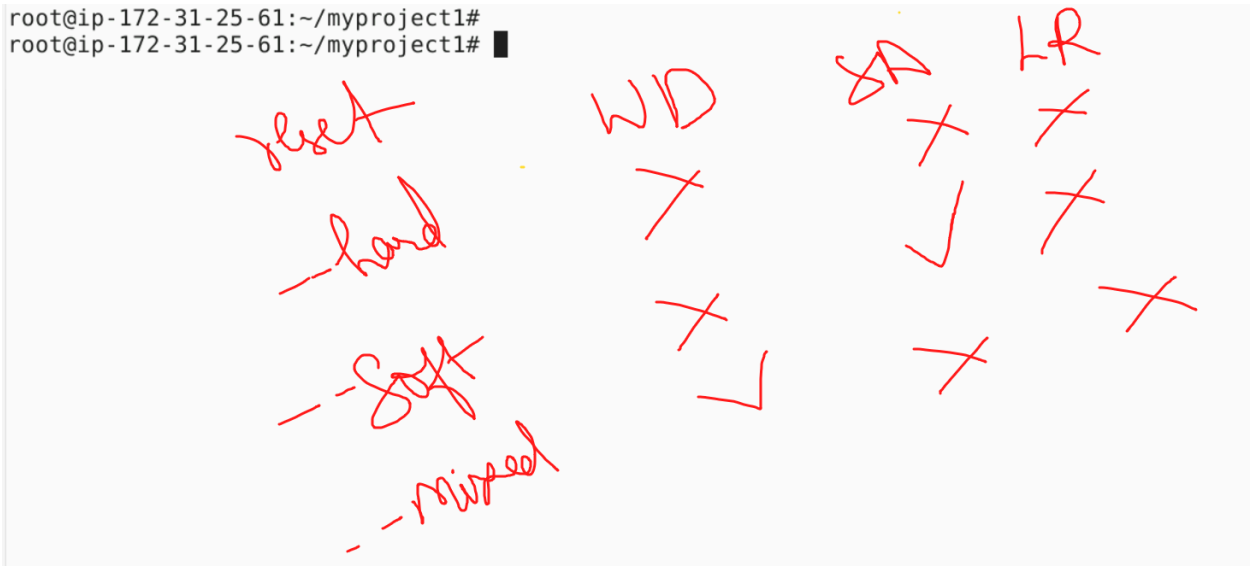
Select the commit id which is at the bottom of your log.

git reset --hard <commit id>

Example : git reset --hard 04f432e

See all the reset operation

 # git reflog



Branching & Merging
==========

Command to see how many branches do we have in out local repo:

$ git branch

$ git status ⇒ working tree should be clean

Command to create a branch with name as feature1

$ git branch feature1

Command to switch to branch with name feature1

$ git checkout feature1

Switched to branch feature1

Create a new file on the branch feature1

$ touch login

$ git status

Untracked file

$ git add login
$ git commit -m “done on branch”
$ git log --oneline

Merging the feature1 branch to master branch
==========

By merging we mean

The source branch → login has commits that will be merged to destination branch master

Always Switch to the destination branch where we have to merge

$ git checkout master

$ git merge feature1 master

$ git log --oneline

All commits of login are present in master also, login file is also there on master branch

==========

Rename a branch

 # git branch -m feature1 dev

Delete a Branch which has been merged:

 # git branch -d dev

Recover a branch back using the hash

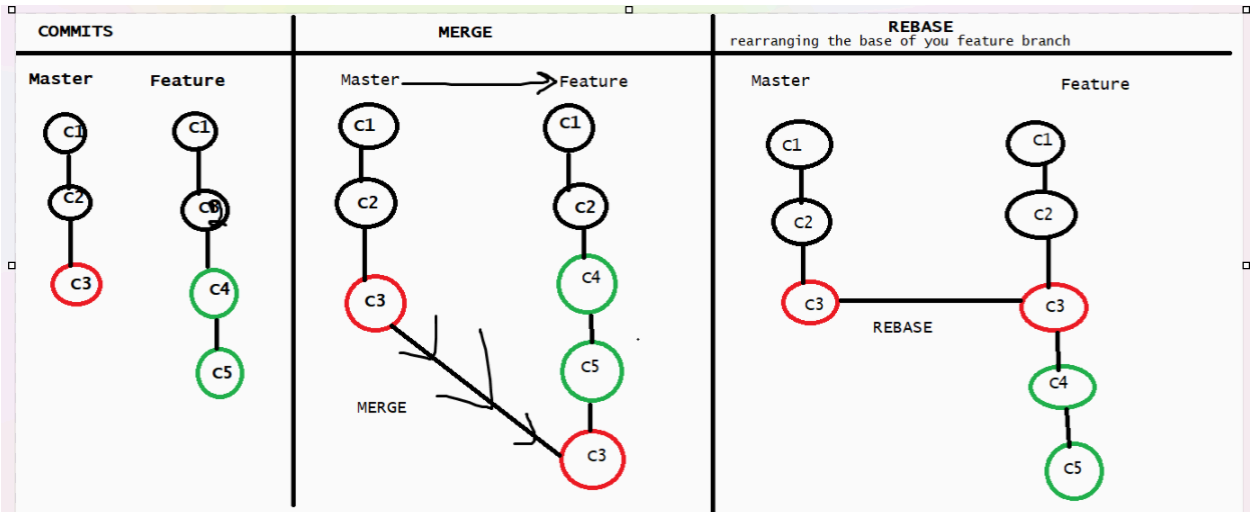
 # git checkout -b feature_dev <commit_hash>

==========

Agenda: 26th April
==========

- Rebase merging strategy
- Conflicts and resolve merge conflicts
- Stashing in git
- GITHUb repo
- Pull, push, clone and forking
- Pull request in github

Demo1: Rebase:
==========



sudo su -

mkdir myproject

cd myproject

touch index1.html index2.html

git config --global user.name sonal04
git config --global user.email sonal04@gmail.com

git init

git add index1.html

git commit -m "done c1"

git add index2.html

git commit -m "done c2"

git log --oneline

Create a new branch

git branch feature

git branch

Make a new commit on master branch

echo "content on index1.html" > index1.html

git add .

git commit -m "done C3"

Checkout to feature branch

git checkout feature

touch file1

git add .

git commit -m "done C4"

touch file2

git add .

git commit -m "done C5"

Apply the rebase merge strategy

git rebase master

git log --oneline

You will see commit history is rewinded.

==========

Stashing in GIT:
==========

git checkout master

git status

Working tree has to be clean

echo "content on index1.html" > index1.html

git stash

Changes will be gone from working directory

git status

git stash list

git show stash@{0}

Get back the changes from stash

git stash pop stash@{0}

==========
CONFLICTS and resolve Merge conflicts :
==========

git checkout master

git add .
git commit -m "done"

Create a new branch

git branch login_branch

You are still on master branch

Create a login file

echo "add username" > login

echo "add password" >> login

git add .

git commit -m "done on master"

Switch to the login branch

git checkout login_branch

Create new file login with new changes

Create a login file

echo "add forget password " > login

echo "add sign in code " >> login

git add .

git commit -m "done on login branch"

Now switch to master

Merge login to master

git merge **login_branch** **master**

You will get Conflict now

To resolve conflict on master branch→

Open the login file and remove all the extra special characters

vim login

Remove all special characters.

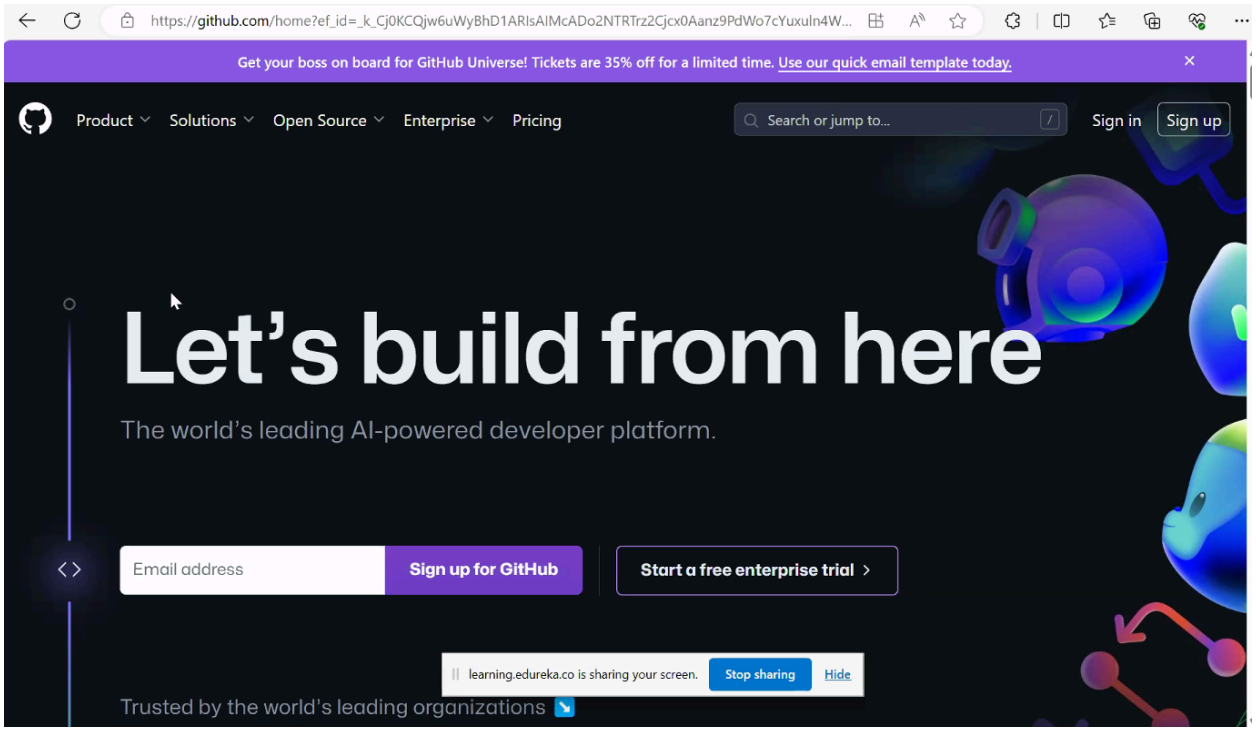
Save the file(press ESC key and :wq -> press enter key)

git commit -m "done merging"

==========

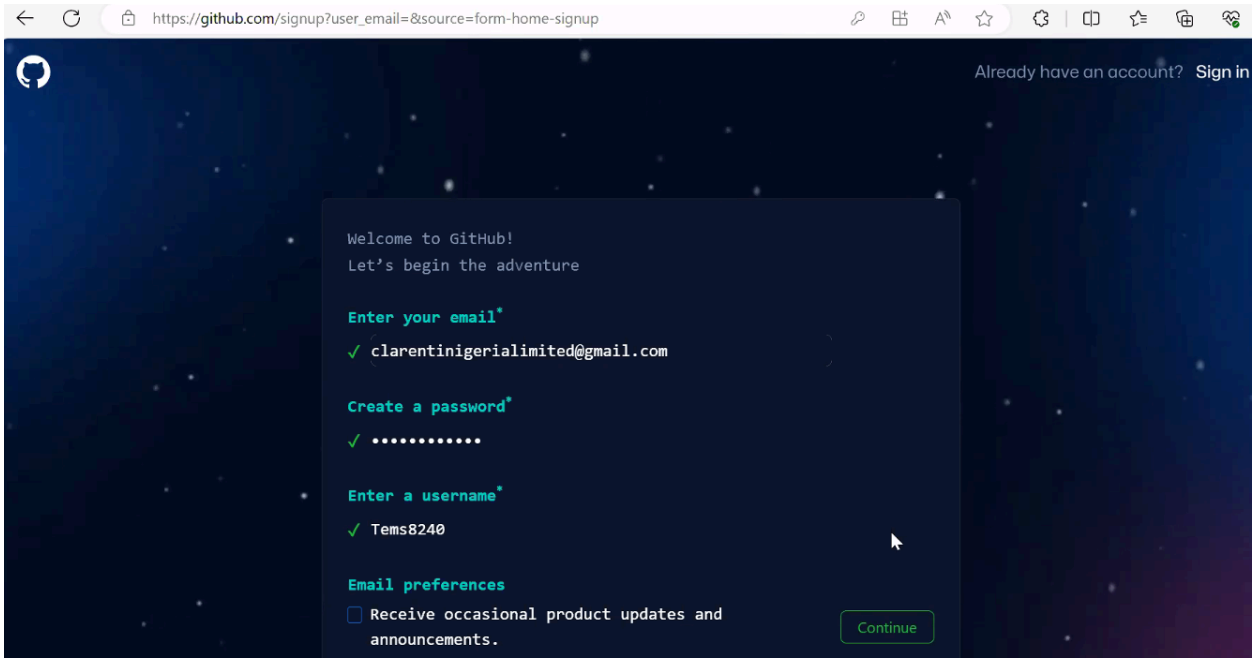
To create account in github:

https://github.com/



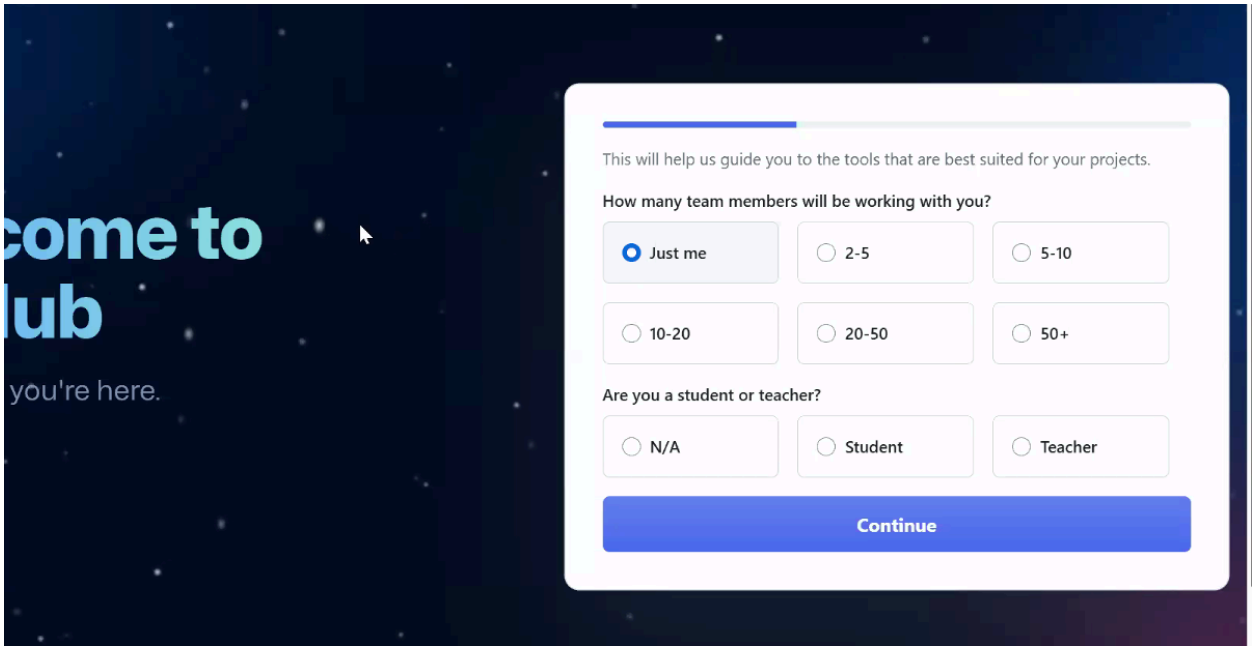
Click on signup

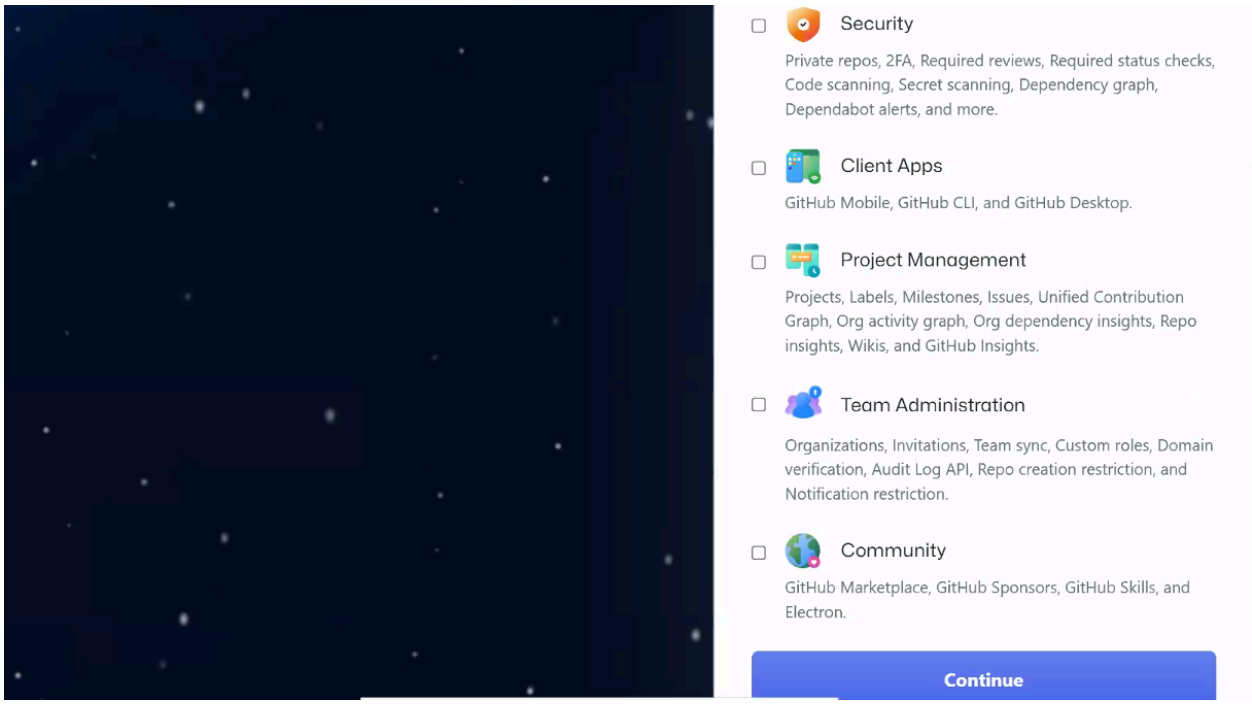
Given valid email address
Give password
Give username
Click on continue.



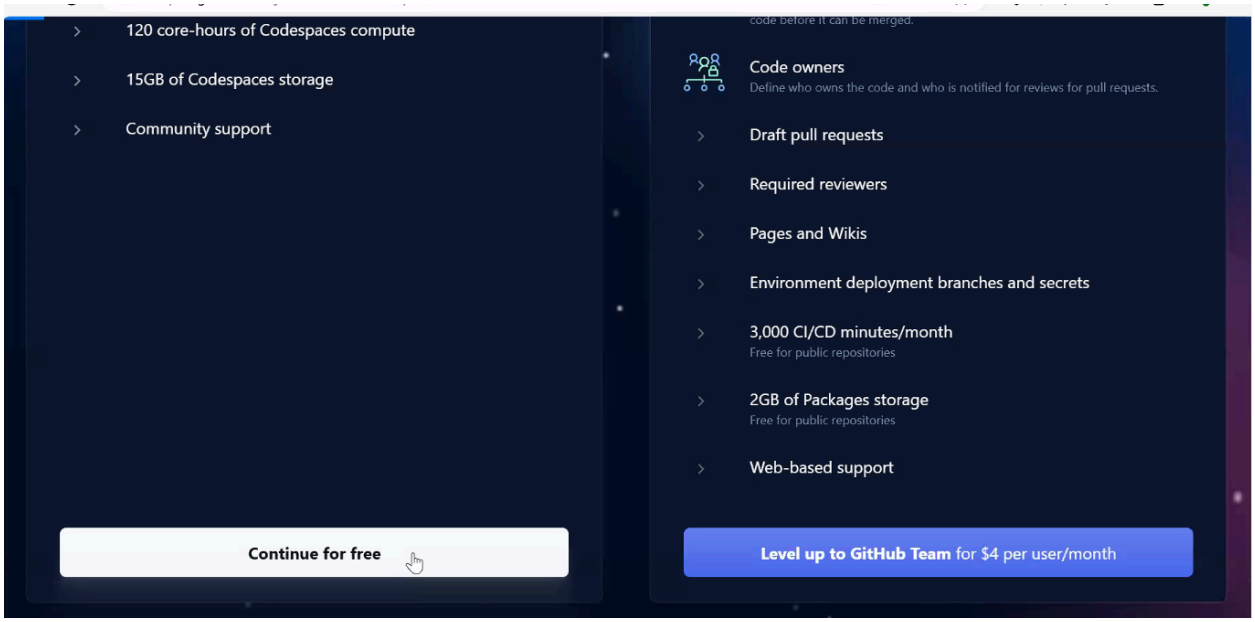
An email will be sent on your mail id
Verify the email
And create the account

> Login to github





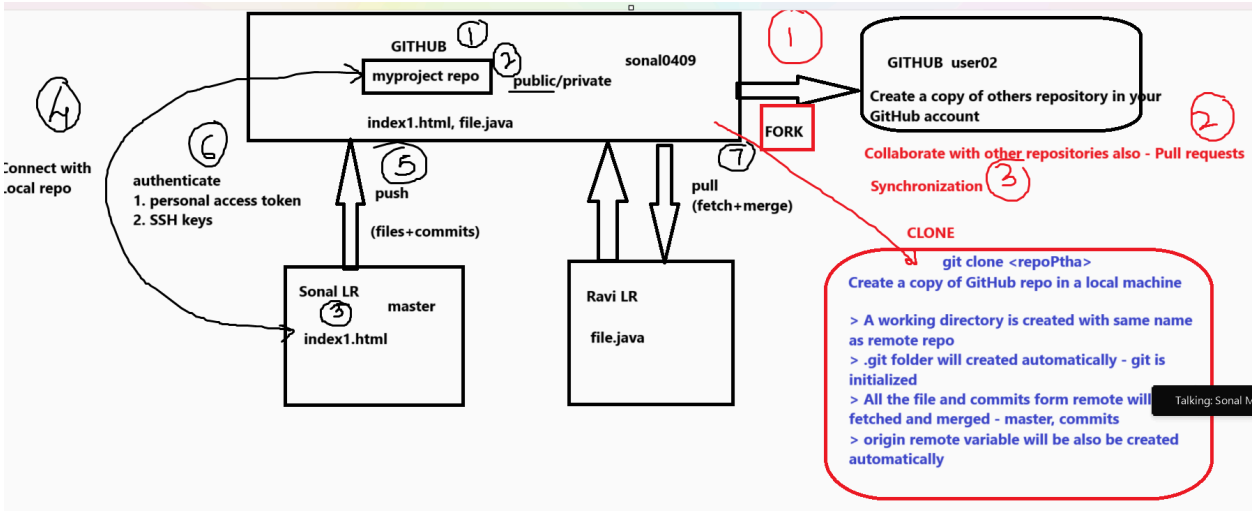
Continue for free



You will be on the github dashboard.

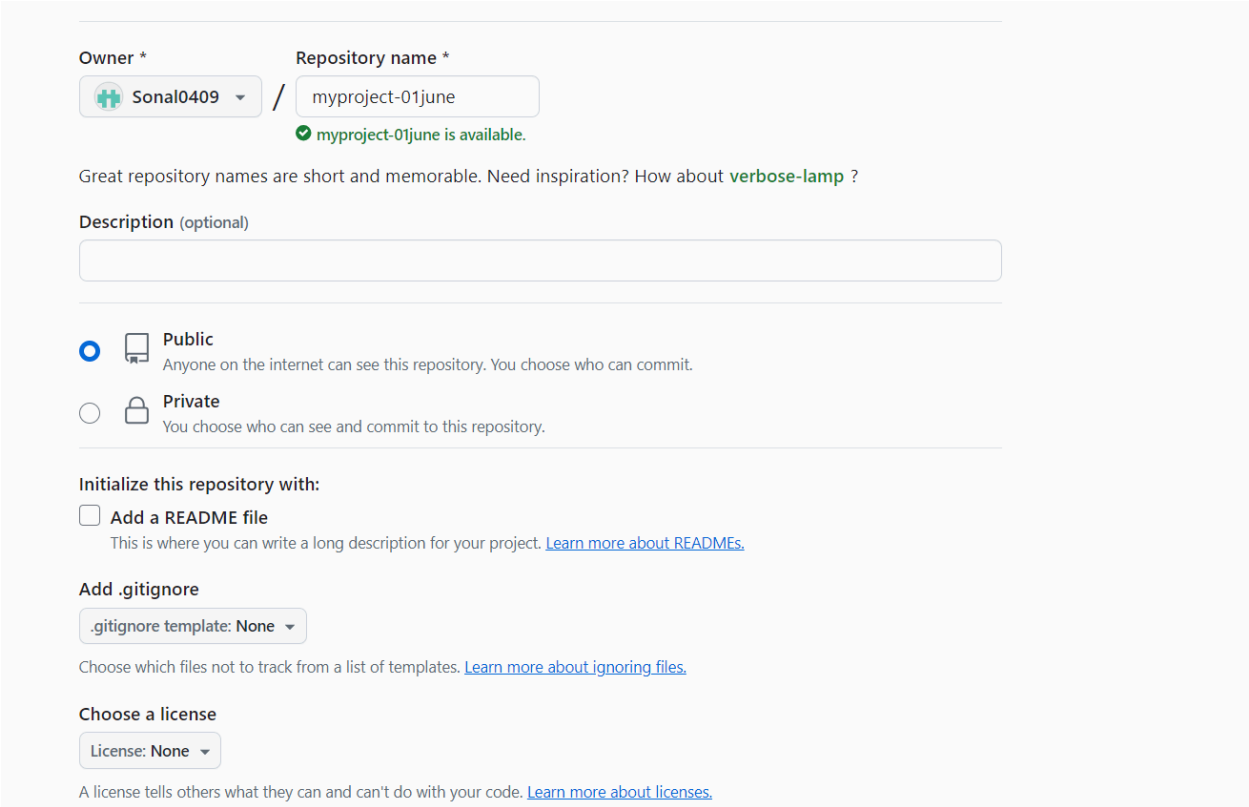
Working on Remote repository:

==========



Create a repository:

1. > click on New button to create a remote repo
   > Name to the repository
   > select public repo
   > click on Create repository



Connect Local repo to remote repo

Go to git and execute this command

$ git remote add origin <your remote repo Path>

Example like this:

…

git remote add origin https://github.com/Sonal0409/myproject27June_sonal.git

…

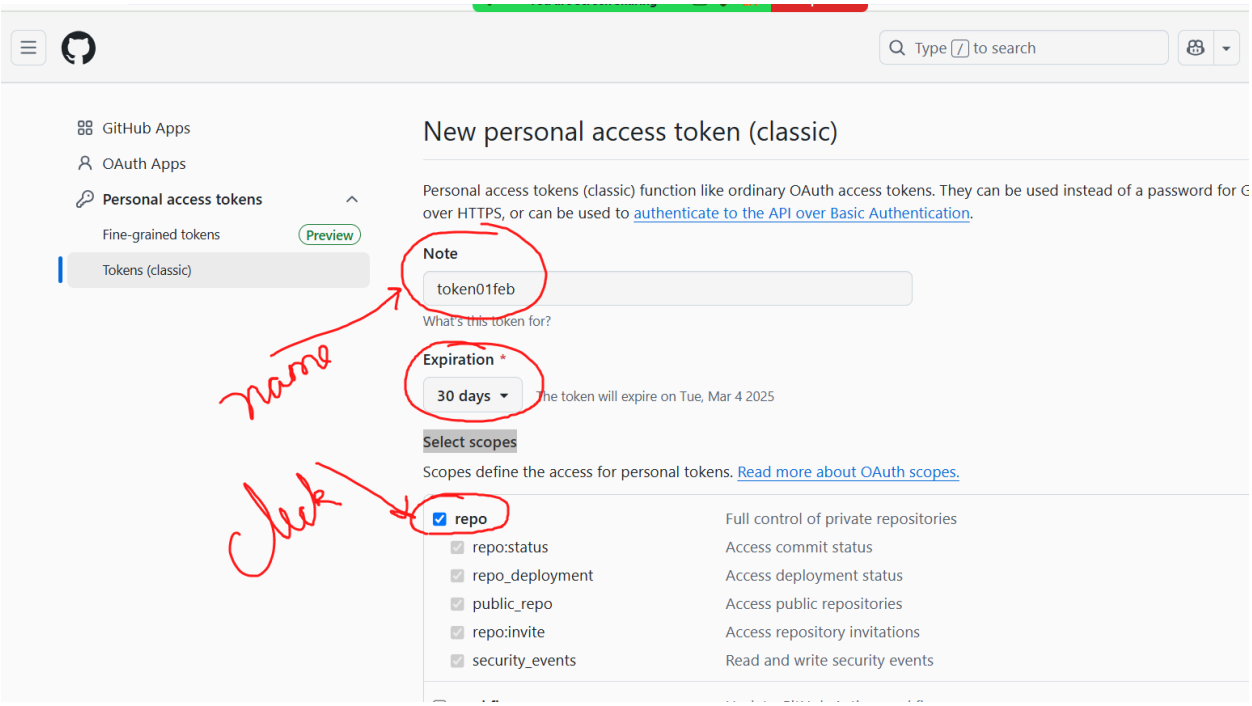
Execute the command to check local and remote repo is set:

$ git remote -v

Create a Personal Access token on GITHUB to push your changes

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS

GO to this link: https://github.com/settings/tokens



Push Changes to GITHUB repo

git push origin master

Here origin means variable storing path of remote repo

It will now ask to enter username

Username for 'https://github.com':

Here enter the token and press enter key

Again it will ask for password

Password for 'https://ghp_mJ92gregYqNggrggd8AY1NxkWpYigter345Q6PHAM7VxuLg26F4wB@github.com':

JUST PRESS ENTER KEY, do not enter anything for password

It will send the files and commits to remote.

==========

PULL Workflow command

==========

Create a new file on the remote repository master branch

Now go to lab machine and execute below command

 # git pull origin master

Pull command will fetch new files and changes and merge them with local repo and working directory

==========

FORKING IN GITHUB

==========

SYNCHRONIZATION & COLLABORATION:

==========

Copying the github repo of a user into your own github account

Forking operation will allow you to freely experiment changes on repo copied from other

Demo: Collaboration with other repo on github

==========

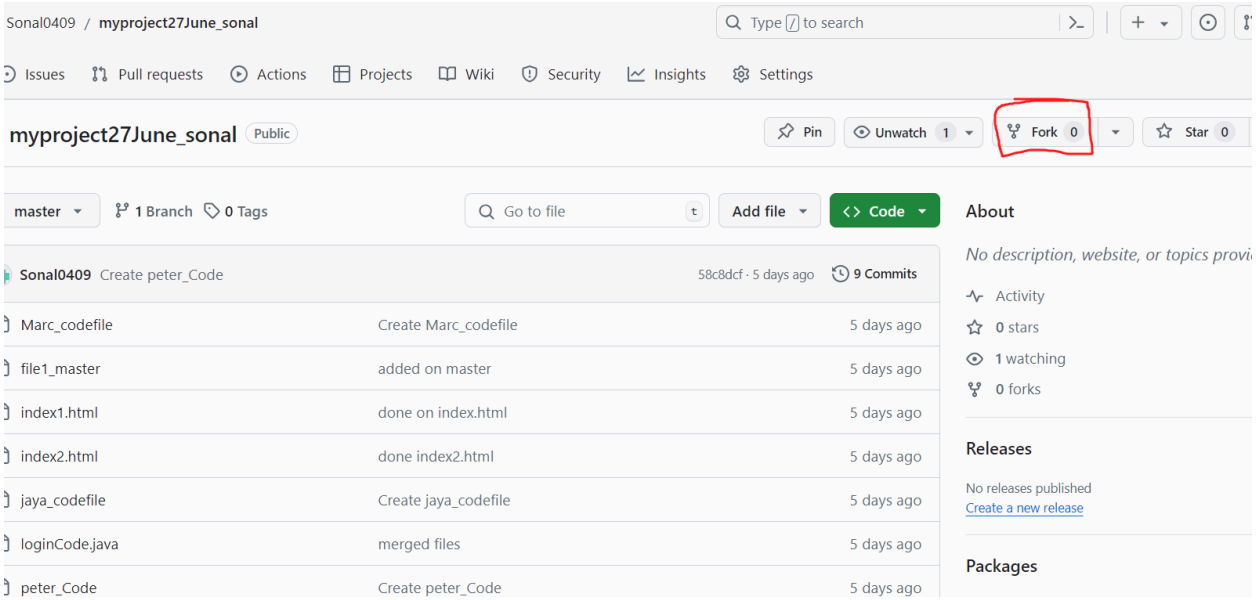
Step 1:

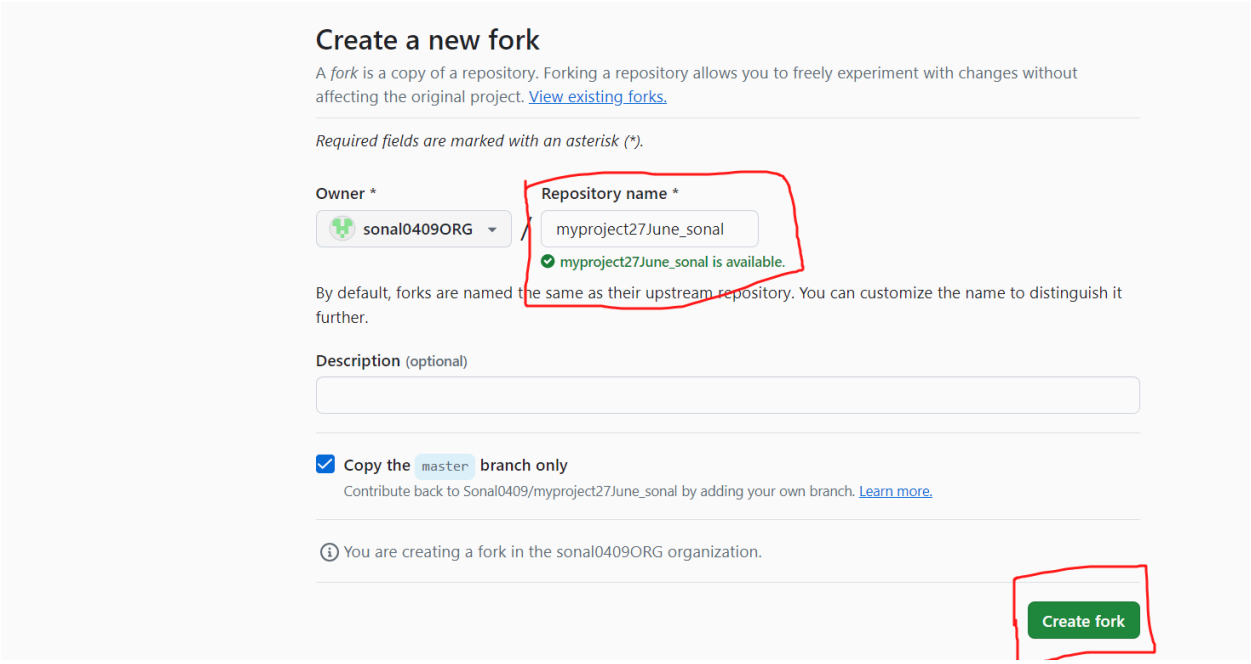
Take a parent repository and make copy of it in your github account

> You should be logged into github

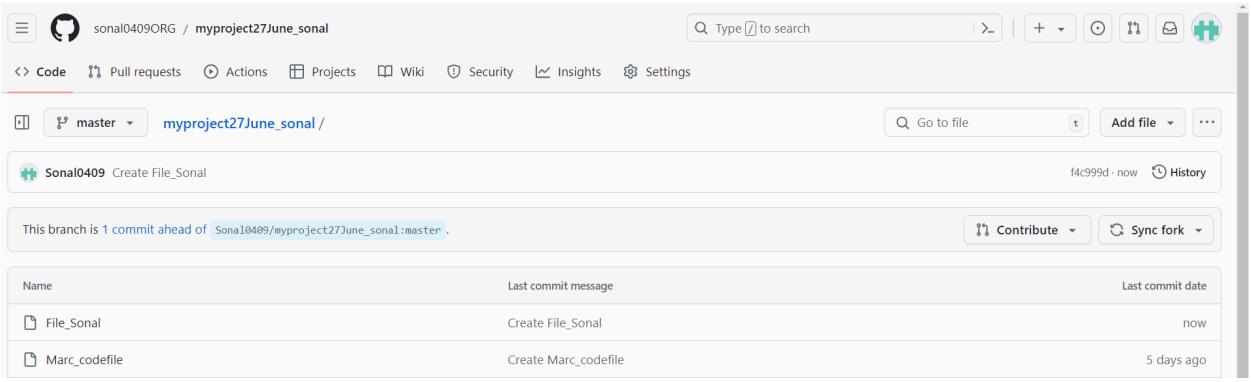
> In the browser of your machine open this repository: https://github.com/Sonal0409/myproject27June_sonal.git

> Click on the fork button



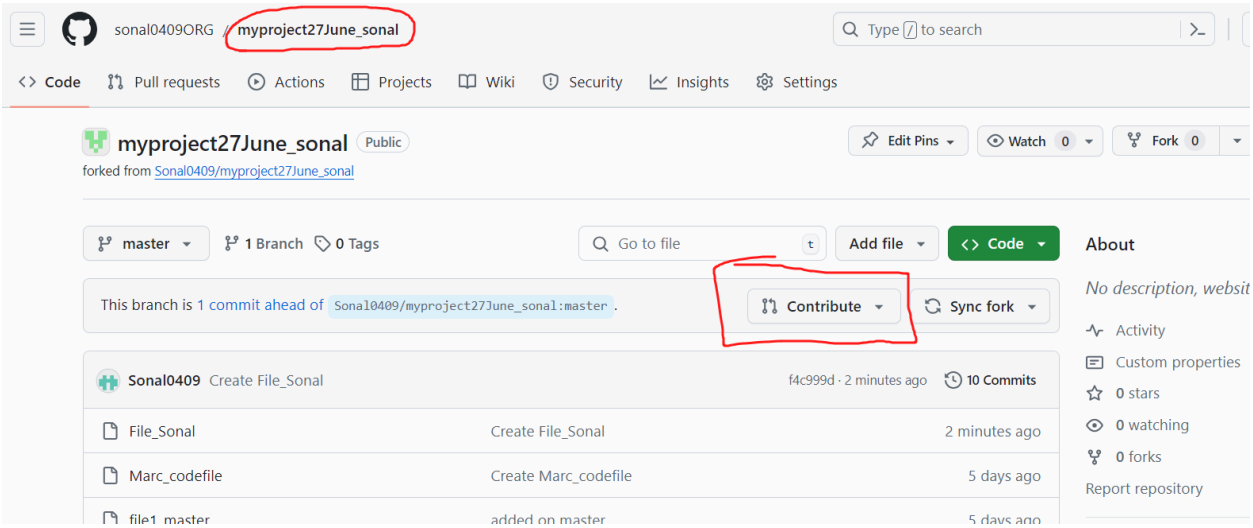


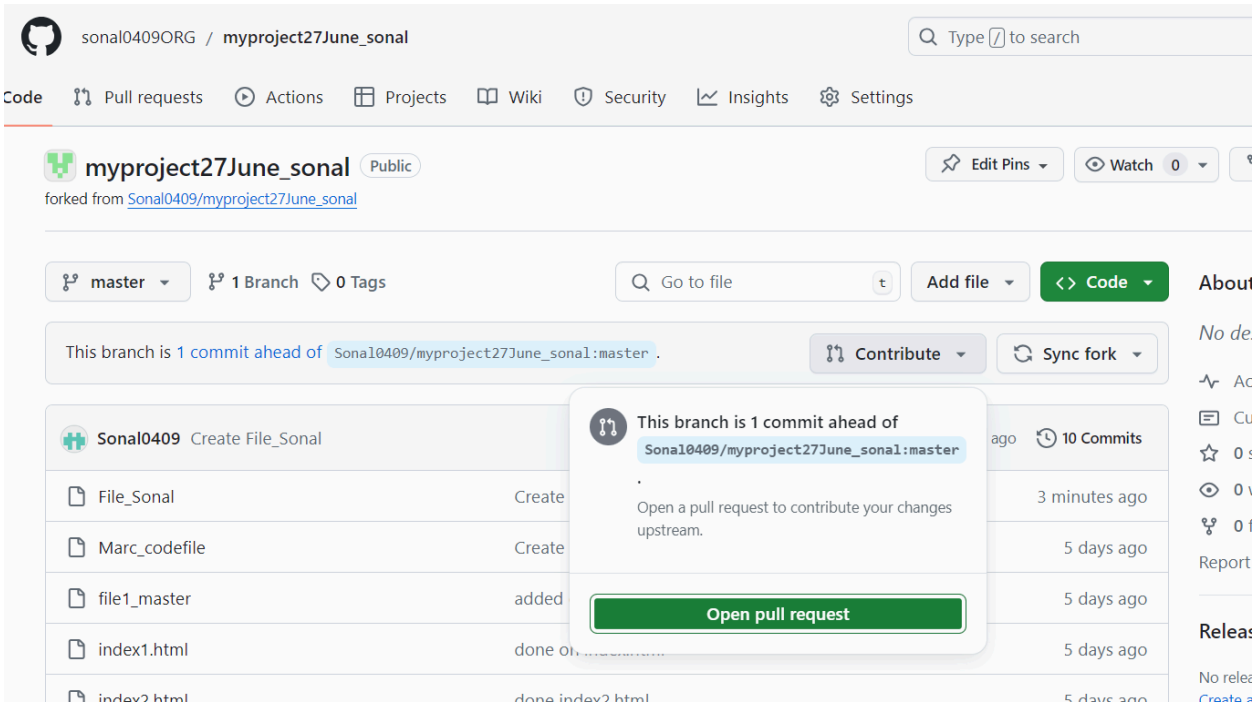
Step 2:
Make some changes by creating a file in your copied repo on your github account



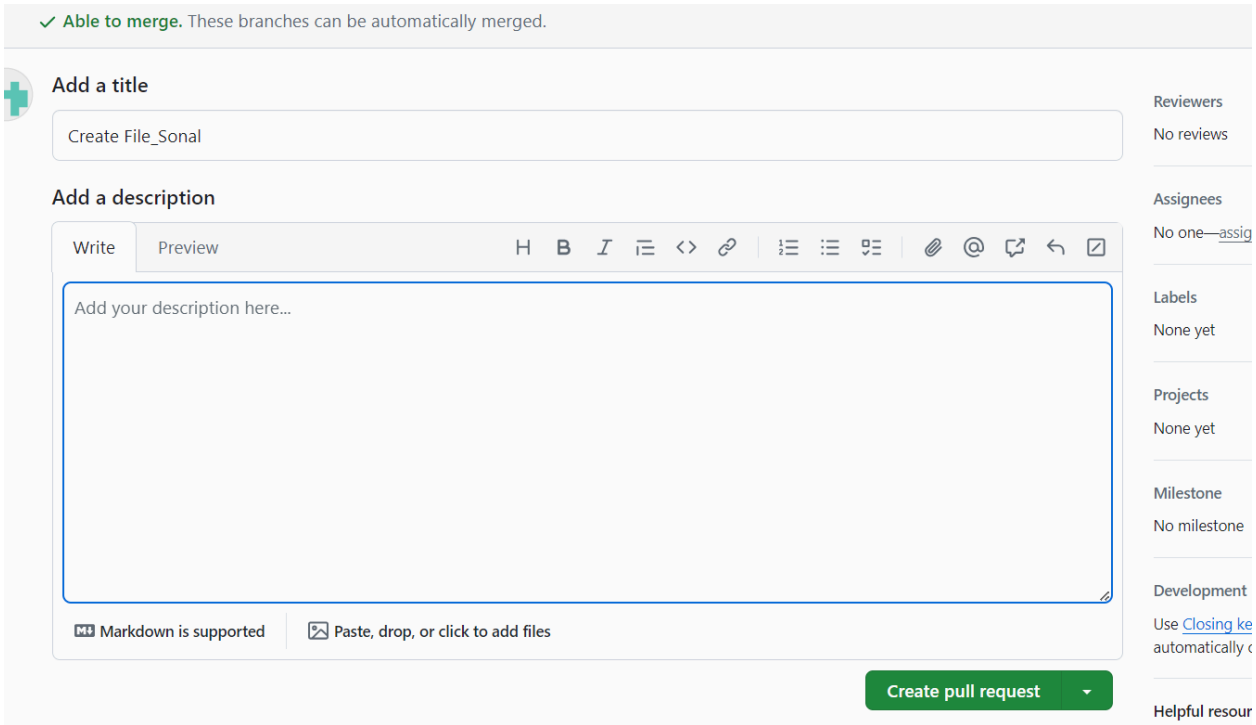
Step 3: Raise a request with parent repo to merge you files and changes

We will contribute and raise Pull requests in github





Click on create pull req

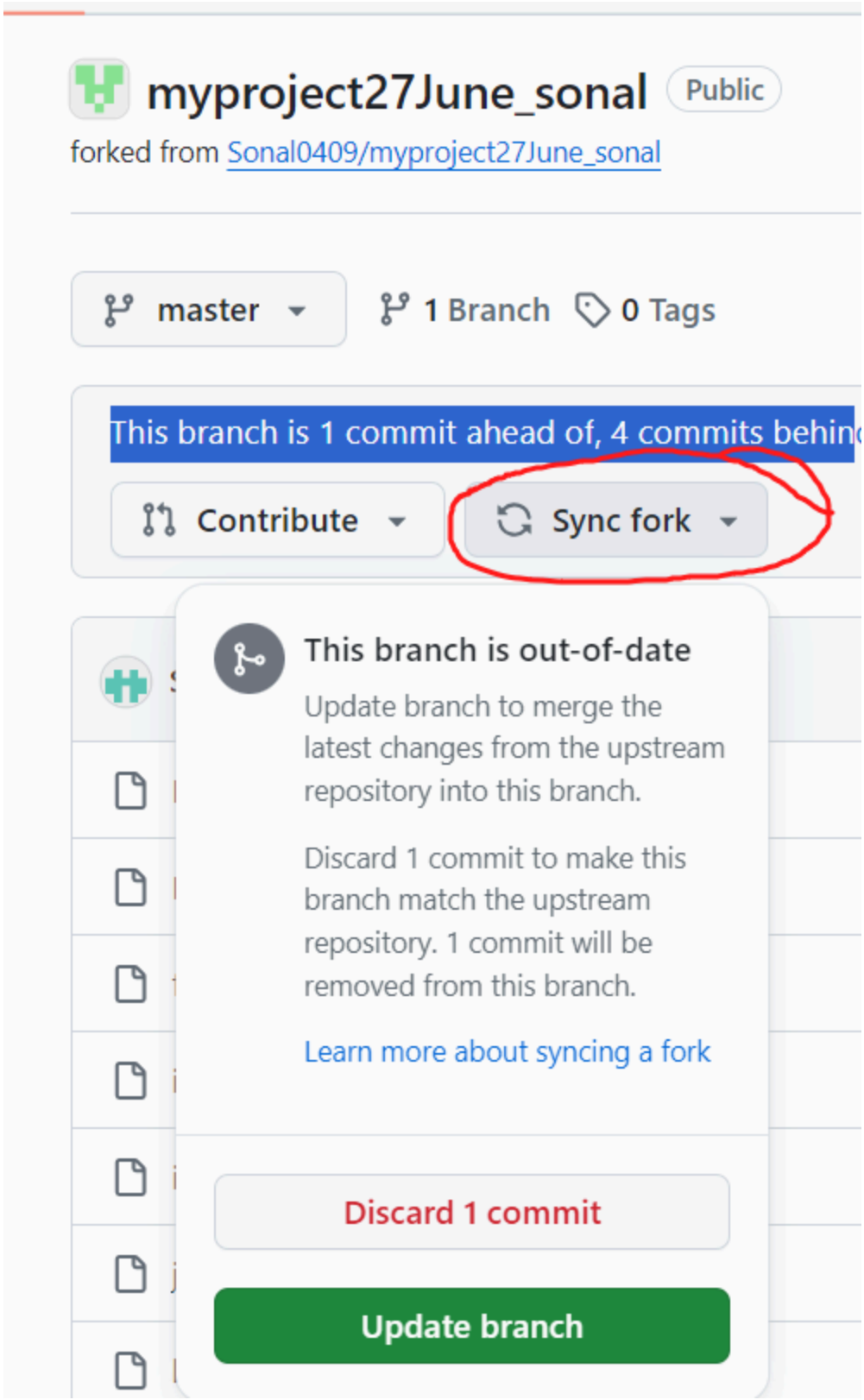


Sync the 2 repositories

==========

If parent repo is having changes/files thata r enot in your copied github repo

Then you can click on Sync button → update branch button and merge the changes between 2 repo


myproject27June_sonal
Public
forked from Sonal0409/myproject27June_sonal
master
1 Branch
0 Tags
This branch is 1 commit ahead of, 4 commits behin
Contribute
Sync fork
This branch is out-of-date
Update branch to merge the latest changes from the upstream repository into this branch.
Discard 1 commit to make this branch match the upstream repository. 1 commit will be removed from this branch.
Learn more about syncing a fork
Discard 1 commit
Update branch

Day 5: 27 April 2025

==========

Agenda:

==========

- Cloning in git
- Continuous Integration with jenkins
- Integrating Jenkins with GITHUB
- Integrating Jenkins with Build tool -Maven

Clone in GIT:

==========

Go to the terminal

sudo su -

git clone https://github.com/Sonal0409/myproject26sept-sonal.git

It will create a working directory with name as
myproject-26april-sonal

It will create a git repo

cd myproject-26april-sonal

ls -al

git log --oneline

git remote -v

==========

CONTINUOUS INTEGRATION:
==========

With Devops we have to automate the entire process of Build
This can be achieved by using an automation tool that will manage integration with
 - Version control tool - to fetch the code
 - Build tool - to compile, review, test and package the code
Various build tool : Maven, Ant , gradle, MSbuild, pybuilder

 The integration/automation tool is Jenkins
 We have many integration tools like:
 Jenkins
 Gitlab CICD
 travis
 bamboo
 teamcity
 Github actions
 Airflow

The integration service on various cloud platforms:
 AWS- code pipeline
 Azure-Azure pipelines
 GCP - Code build

Digital ocean - pipelines

Jenkins:
==========
It is an automation tool that integrates with various tools to provide an automated SDLC pipeline
Jenkins can automate entire build and automation process
It is an automation server, Jenkins support building, deploying and automating any project.
Pls note:
jenkins is not a build tool
jenkins is not a testing tool
jenkins is not a deployment tool
jenkins is not a monitoring tool

Jenkins features:
==========
> Open source tool, free
> Enterprise version of Jenkins can also be used which is provided by cloudbees.
> Jenkins is a java based tool
> for us to install and work with Jenkins we have to first install java(JDK) version 17 OR 21
> Jenkins can be installed on windows, Mac or linux OS
> Jenkins is a plugin based tool
plugin - an extension, a small tool that enhances the capability of the main tool
> Jenkins has 100s of plugins to support integration process with various tools
For example: If we want Jenkins to send notification to our slack channel than we download

slack notification plugin on jenkins that will help to configure jenkisn+slack integration
> CICD pipelines : In Jenkins we make use of declarative pipeline syntax to write pipeline code
Pipeline -> a set of tasks which are executed one after the other in a sequence
To create the pipeline - we have to write pipeline code
It is written in declarative pipeline syntax - which is based on groovy scripting
> Secure tool -> security features -> Matrix based security or project based security
> We can create users and give them role based access to them
> We can setup a high availability server and can ensure regular backup of Jenkins server
> Jenkins Controller -> main VM where we have all the pipelines created but they are not exeucted on this VM
Rather the controller is connect to agents where the piplines and tasks are executed.
This setup is called master and Agent setup in jenkins.

We will work on the jenkins dashboard instead of the terminal.

Setup jenkins on the Lab
==========

Go to browser of the Lab
Give URL as : localhost:8080

Username: admin
Password: Root123$

You will be on jenkins dashboard

For the setup to be complete, we have to update the jenkins plugins

Go to manage jenkins → Scroll down to Plugins

Click on updates:

Select the checkbox for updates and click on update button

Give few mins for updates to complete

Now reset jenkins

Scroll down and click on checkbox to restart jenkins server

==========

Hardware req for jenkins:
4GB RAM and 2CPU core

Steps to Install jenkins on UbuntuOS:
==========

https://www.jenkins.io/doc/book/installing/linux/#debianubuntu

\
Steps to Install jenkins on Amzon Linux or RHEL OS:
==========

https://www.jenkins.io/doc/book/installing/linux/#red-hat-centos

Install jenkins on Windows:
==========
https://www.jenkins.io/doc/book/installing/windows/

Steps to Install jenkins on Ubuntu 22

Install java version 17

sudo apt update

sudo apt install openjdk-17-jdk -y

Install jenkins on Ubuntu 22:

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]"
https://pkg.jenkins.io/debian-stable binary/ | sudo tee
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
systemctl start jenkins

Install Jenkins on Amazon Linux 2023

sudo dnf install java-17-amazon-corretto-devel
sudo wget -O /etc/yum.repos.d/jenkins.repo
https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import
https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
sudo yum upgrade
sudo yum install jenkins
sudo systemctl daemon-reload

==========

Demo 1: Create a JENKINS job that will execute linux commands.
==========

Create a new job in jenkins
Click on + sign to create new item/job/project
Give a name to the job : Job1
Select freestyle project and click on OK button
Go to build Steps→ select Execute Shell→ give commands like
touch file1
echo “hello Jenkins!”

Save the job → click on build Now button.

See the console output and check the workspace directory.

Jenkins Integration with Git and Github-> Source code management tool
==========

Demo 2: Create a Jenkisn job that will clone a github repository in jenkins workspace

Create a new job in jenkins
Click on + sign to create new item/job/project
Give a name to the job : CloneRepo
Select freestyle project and click on OK button
On the project click on Source code management
Select git option
Give git hub repo path
https://github.com/Sonal0409/myproject-03Feb-Sonal.git

Branch name as ===> Master

Save the job

Click on Build now

Click on workspace

Repository will be cloned in jenkins workspace

==========

Demo: Maven Integration with jenkins
==========

Jenkins--> Manage Jenkins-->Tools

Under GIT==> leave it same

Under maven ==> lets install it automatically

Type:
Name as : mymaven and
check the install automatically box.
Save the changes

We can check maven installation in the following location

 # cat /var/lib/jenkins/hudson.tasks.Maven.xml

When jenkins installs a tool its scope is only /var/lib/jenkins directory -> JENKINS_HOME

==========

Package Job
****

Go to jenkins--> new item--> Name= Package==> freestyle project
==> source codemanagement==>select git==> give git repo
https://github.com/Sonal0409/DevOpsCodeDemo.git

build ==> invoke top level maven target==>mymaven

goal = package

==> save==> build now
==> click on build number and see the console

==> go to workspace ==> target/addressbook.war

==========

Jenkins Continous integration Pipeline

==========

Create a new job → Integration-piepline-> select pipeline template

Add below code in pipeline section

```
pipeline{

   agent any

   tools{
      maven 'mymaven'
   }

   stages{

      stage('Checkout code '){
         steps{

            git 'https://github.com/Sonal0409/DevOpsCodeDemo.git'

         }
      }

       stage('compile code '){
```

```
        steps{

            sh 'mvn compile'

        }
    }

    stage('Test code '){
        steps{
            sh 'mvn test'
        }
    }

    stage('Package code '){
        steps{

            sh 'mvn package'

        }
    }


  }
}
```

Save the job→ build Job → click on stages

Day 6: 03 May

==========

Agenda:

==========

- Jenkisnfile
- Multibranch pipeline
- Continous Deployment with tomcat server - CICD pipleine
- Slack Integration with jenkins

RESTART jenkins on the Lab
==========

Go to browser of the Lab
Give URL as : localhost:8080

Username: admin
Password: Root123$

You will be on jenkins dashboard

For the setup to be complete, we have to update the jenkins plugins

Go to manage jenkins → Scroll down to Plugins

Click on updates:

Select the checkbox for updates and click on update button

Give few mins for updates to complete
Now restart jenkins

Scroll down and click on checkbox to restart jenkins server.

==========

Jenkins--> Manage Jenkins-->Tools

Under GIT==> leave it same

Under maven ==> lets install it automatically

Type:
Name as : mymaven and
check the install automatically box.
Save the changes

==========

Jenkinsfile:

==========

In Jenkins is it not a right approach to directly write pipeline code on Jenkins job itself
we cannot allow multiple team members to loginto Jenkins server and edit the pipeline code
Also the code is not version controlled if written directly in Jenkins

So the correct method is -> to write pipeline code local machine and push it to a repo in GitHub
in this way , all will be able to collaborate with your pipeline code
We will be able to maintain version of the pipeline code

how will jenkins fetch this pipeline code???

Solution:

in Jenkins we use a file called as Jenkinsfile
This file will store pipeline code
this file has no extension

And we just have to create a Jenkins pipeline job --> select the option to obtain the code from GitHub

Create a pipeline job and select pipeline code form SCM

Give the following repo name:https://github.com/Sonal0409/jenkinsfile-demo03May.git

And run the job.

==========

MultiBranch:

==========

We should have a repository in GitHub
In the repo we maintain source code and test cases in multiple branched
The repository should have multiple branches
Each branch should have its own activities or tasks like compile, testing, realse, build, deployment etc etc
We want to create a Jenkins pipeline job for every branch automatically
This pipeline will run the desired tasks related to that branch
In addition to this if a branch has name as Prod -> no pipeline Job for it
The pipeline job should have same name as the branch name
all of this process should be automated

Solution:

==========

Jenkins should create pipeline job --> where can we write pipeline code?? -> Jenkins file

We go to GitHub --> go to repo --> go to required branches

add jenkinsfile --> write pipeline code in the file

The prod branch we will not write any pipeline code-> no jenkisn file

In Jenkins server -->we need to create a job --> that scan the GitHub repo branches

which ever branch has jenkisnfile(pipeline code) -> automatically create a pipeline job for it

which ever branch doesnot have jenkisnfile(pipeline code) -> DO NOT create a pipeline job for it

Also the pipeline jobs created will ahve same name as branch name

Multibranch demo github repo:

https://github.com/Sonal0409/MultiBranchDemo.git

Create a new Job → give name as MultibranchDemo → select project as Multibranch Pipeline → select the source as git and give repo as https://github.com/Sonal0409/MultiBranchDemo.git

Save the job and you will see multiple pipeline jobs created

==========

Install Tomcat Server on the lab

==========

Open the terminal on the lab

```
# sudo su -
```

```
# apt update
```

```
# apt install tomcat9 tomcat9-admin -y
```

Update user information for tomcat

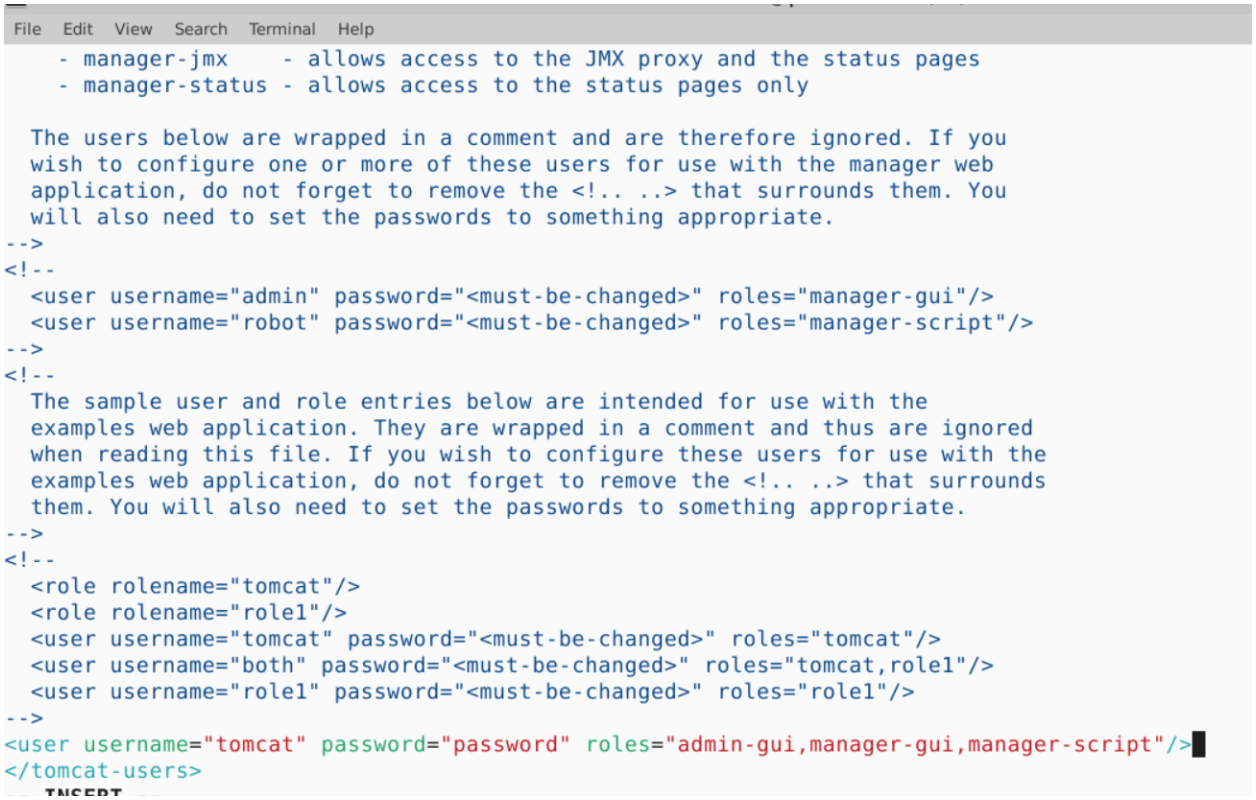
 # vim /etc/tomcat9/tomcat-users.xml

Scroll to the bottom of the file and just above the end tag </tomcat-users>

paste the below line

```
<user username="tomcat" password="password" roles="admin-gui,manager-gui,manager-script"/>
```

There should not be any space before the line



Save the file (:wq!)

Open the server.xml file and change the connector port of tomcat

vim /etc/tomcat9/server.xml

Scroll down to Connector port tag as shown below and change port number to 9090.

```
<Service name="Catalina">

  <!--The connectors can use a shared executor, you can define one or more named thread pools-->
  <!--
  <Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
      maxThreads="150" minSpareThreads="4"/>
  -->


  <!-- A "Connector" represents an endpoint by which requests are received
       and responses are returned. Documentation at :
       Java HTTP Connector: /docs/config/http.html
       Java AJP  Connector: /docs/config/ajp.html
       APR (HTTP/AJP) Connector: /docs/apr.html
       Define a non-SSL/TLS HTTP/1.1 Connector on port 8080
  -->
  <Connector port="9090" protocol="HTTP/1.1"
             connectionTimeout="20000"
             redirectPort="8443" />
  <!-- A "Connector" using the shared thread pool-->
  <!--
  <Connector executor="tomcatThreadPool"
             port="8080" protocol="HTTP/1.1"
             connectionTimeout="20000"
             redirectPort="8443" />
  -->
  <!-- Define an SSL/TLS HTTP/1.1 Connector on port 8443
       This connector uses the NIO implementation. The default
       SSLImplementation will depend on the presence of the APR/native
       library and the useOpenSSL attribute of the AprLifecycleListener.
       Either JSSE or OpenSSL style configuration may be used regardless of
       the SSLImplementation selected. JSSE style configuration is used below.
  -->
```

Save the file

Open this script

 # vim /usr/libexec/tomcat9/tomcat-locate-java.sh

Update as below

Updating the script this way adding the version 21 is solving the issue :

```
find_jdks()
{
for java_version in 21 17 11 10 9 8
do
```

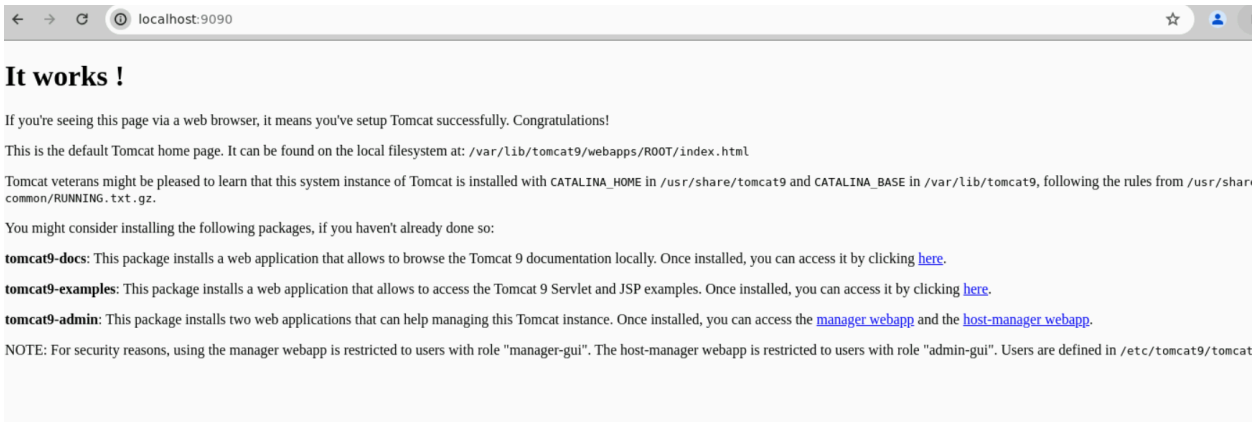
Save the file.

Restart tomcat

systemctl restart tomcat9

To check tomcat is up and running

Go to browser and type localhost:9090



==========

Now go to jenkins dashboard

Go to managejenkins→ plugins → Available plugins

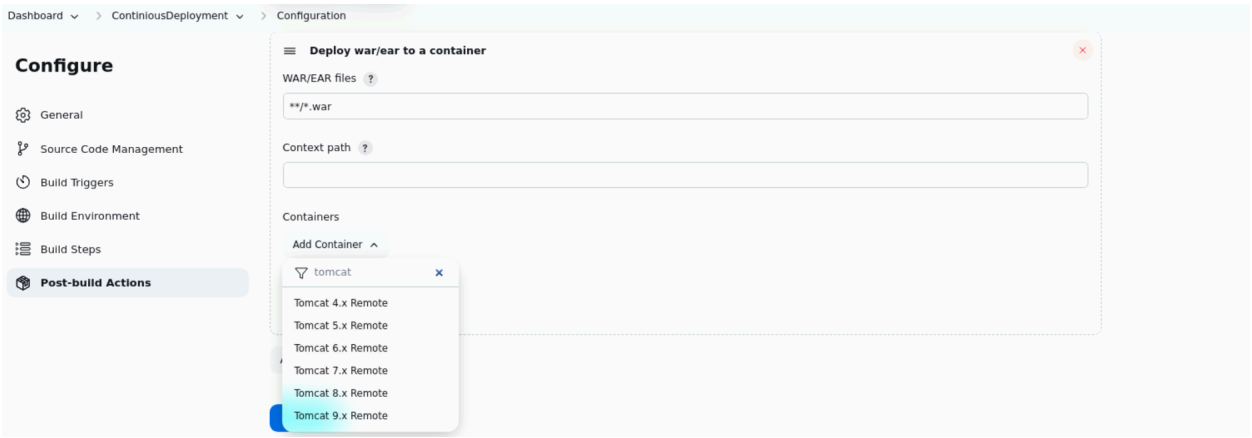
Search for deploy to container plugin

Select it and click on install button

Create a new freestyle Job→ Add the github repo in SCM https://github.com/Sonal0409/DevOpsCodeDemo.git

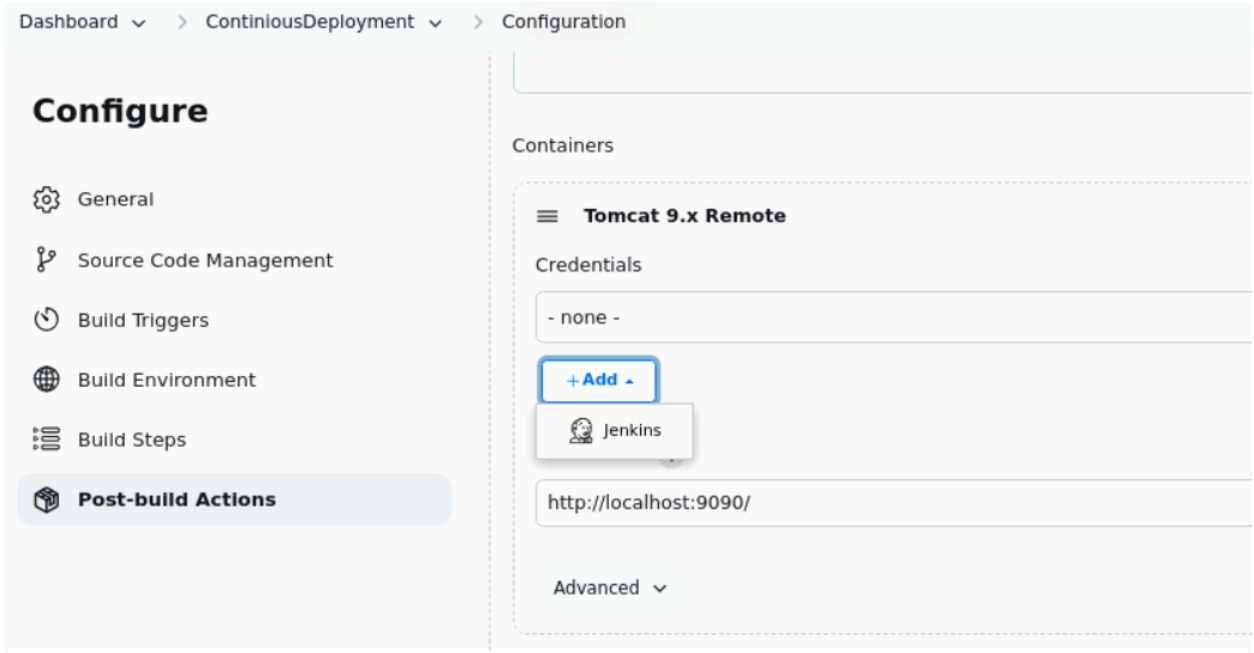
→ add build steps -> select invoke top level maven target → select maven version mymaven → give goal as package

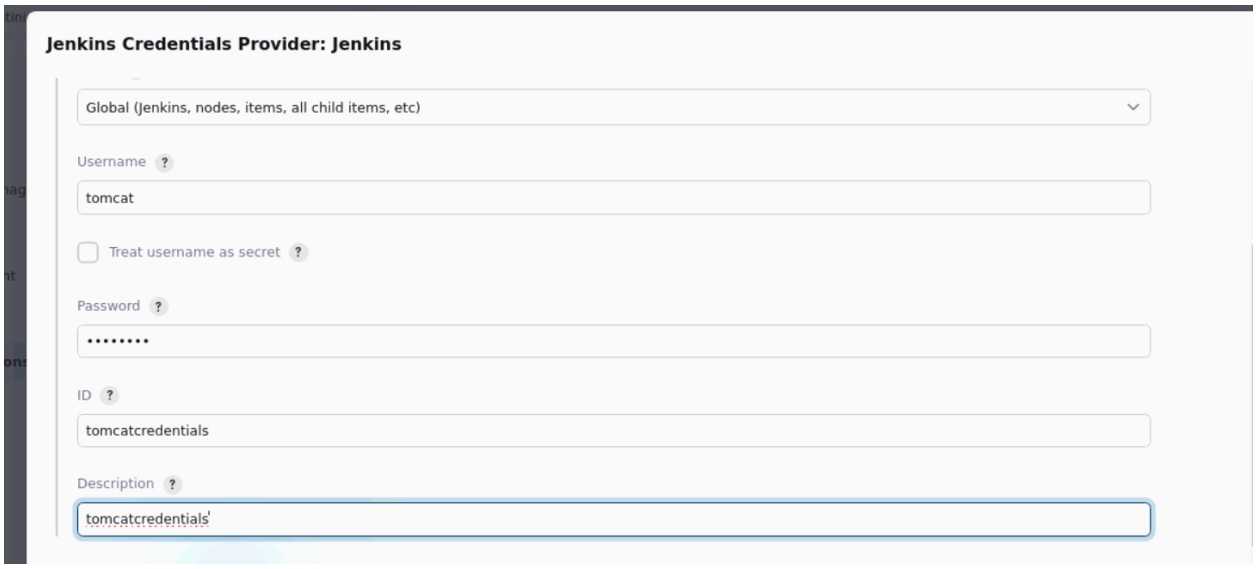
→ go to post build actions and select Deploy war/ear to a container



Add the container as show above

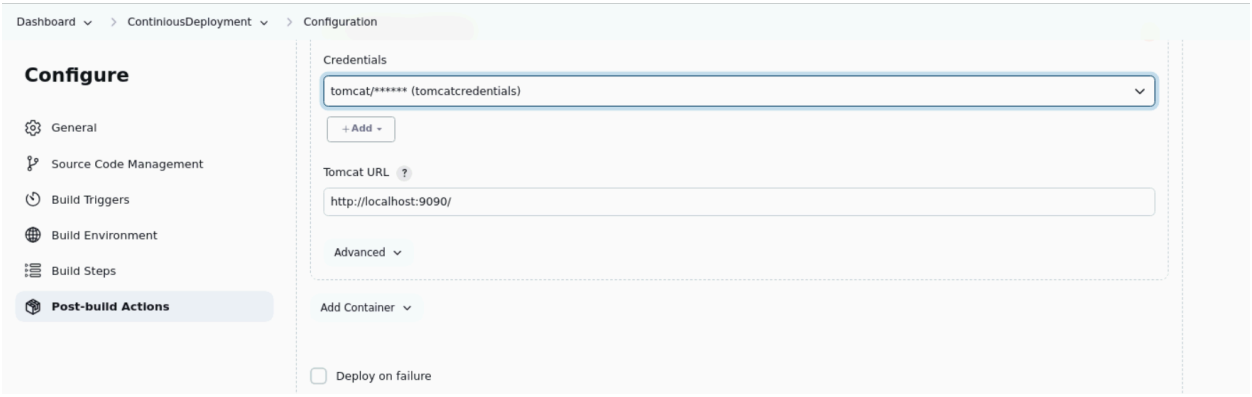
Now add credentials





Click on add button for credentials to be added

You will see the credential in the drop down



Add the URL as : http://localhost:9090/

Save and build the job

Check the deployed application on browser:

http://localhost:9090/addressbook/

==========
CICD Pipeline
==========

Create a new job → select pipeline project →press ok →
click on pipeline tab → give below code,

Note: In the below code for deployment, change the credentials id.. Give your credentials id- like tomcat-server

```
pipeline{

   agent any

   tools{
      maven 'mymaven'
   }

   stages{
      stage('checkout code')
      {
         steps{
            git
'https://github.com/Sonal0409/DevOpsCodeDemo.git'
         }
      }
```

```
        stage('Build and Deploy the Code')
        {
           steps{
              sh 'mvn  package'
           }

           post{

               success{
                  deploy adapters: [tomcat9(credentialsId: tomcat-server, path: '', url:
'http://localhost:9090/')], contextPath: null, war: '**/*.war'
               }

           }
        }

    }
    }
```

Save it and run it

Click on stages to see the execution.

==========

==========

Slack Integration with Jenkins

==========

Step1:

Connect to the slack tool and create a workspace on slack

Create a channel on Slack

1. **Create a slack workspace and a slack channel**

   Go to : https://app.slack.com/

   Sign in with google account → select the gmail account→ press continue

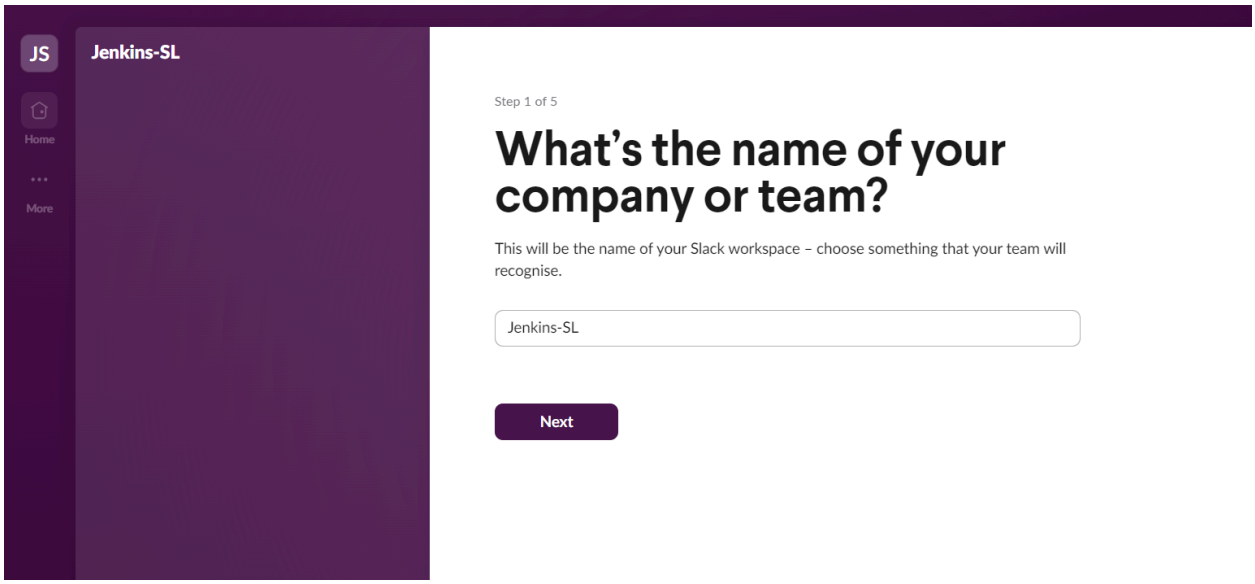
   Click on Create Workspace

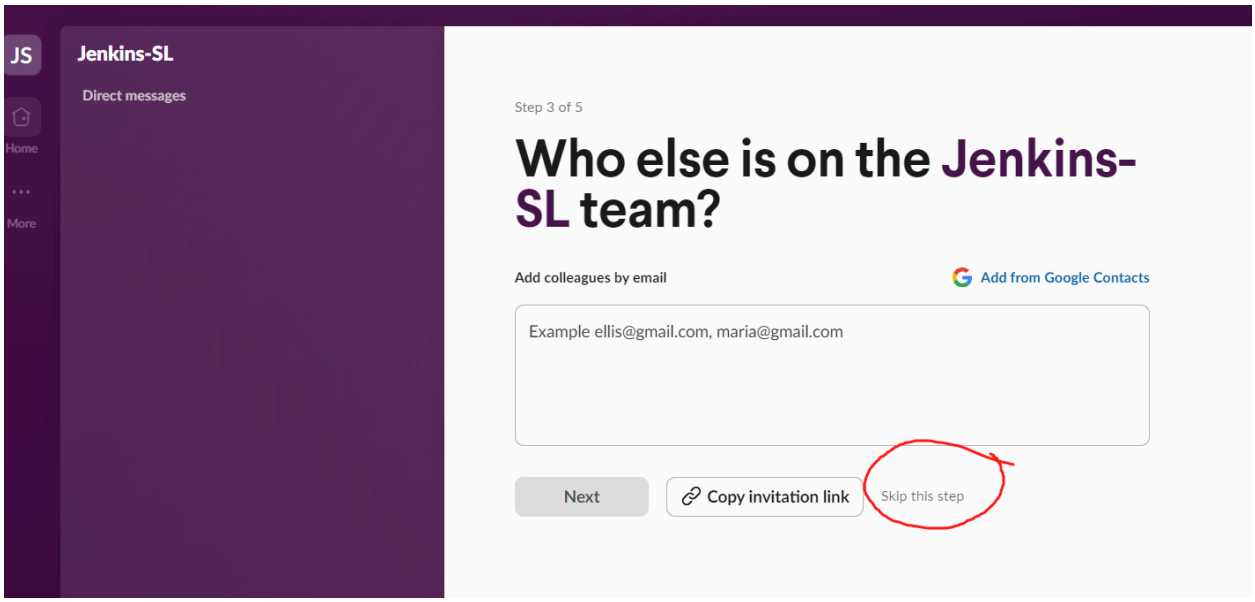
   What's the name of your company or team? : give as => slack-demo

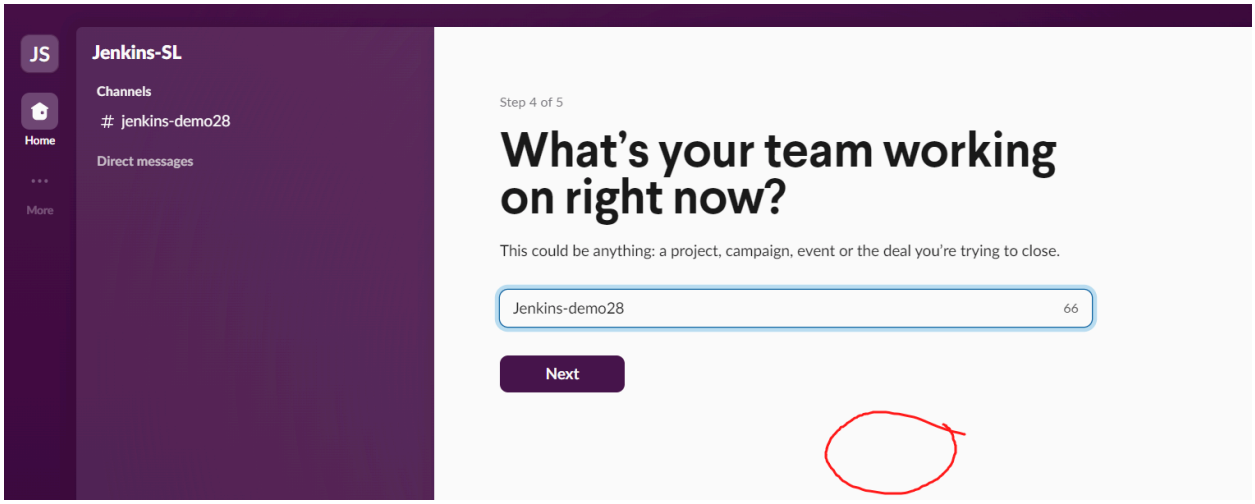
   This will your workspace name

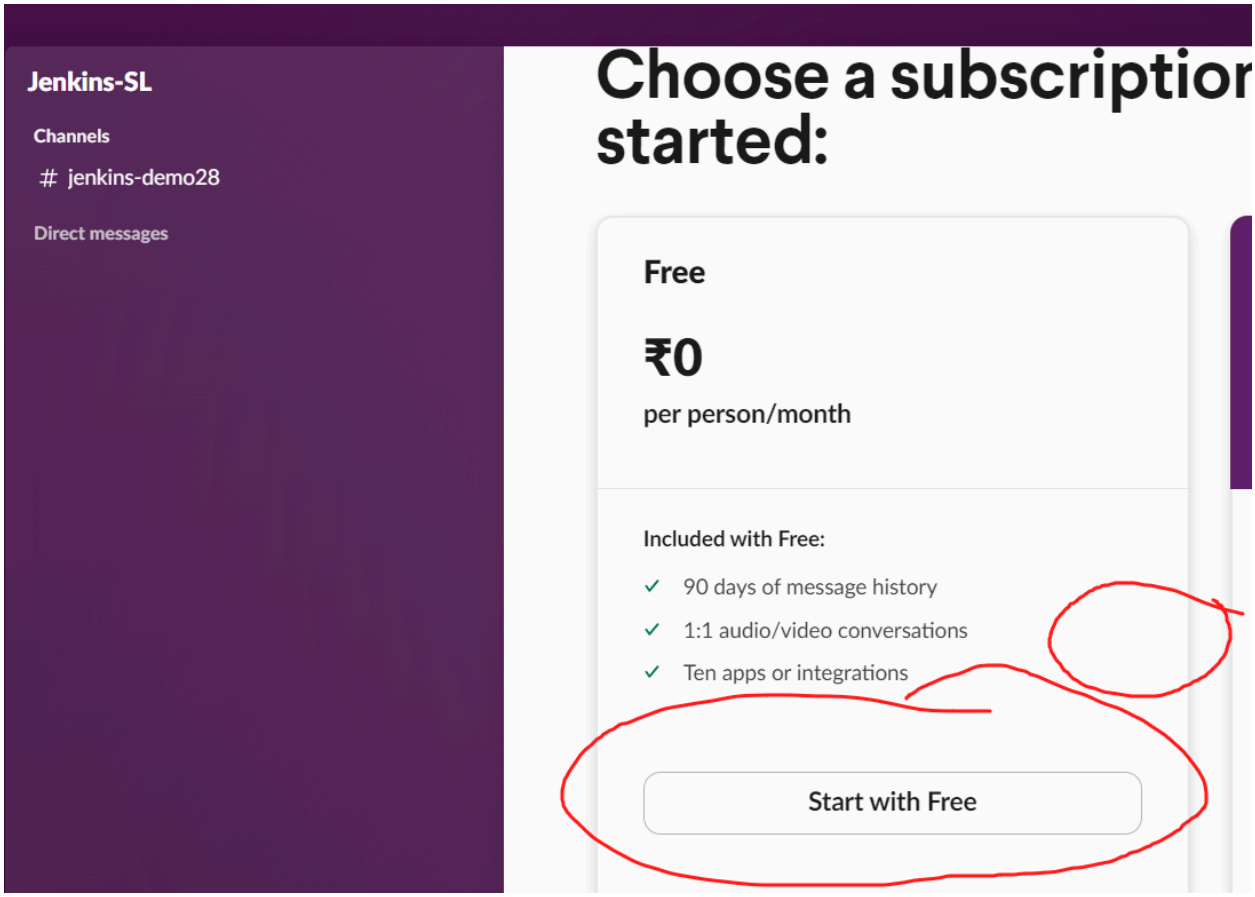
   Press next→ press next → and then skip the last step. -> click on skip this step

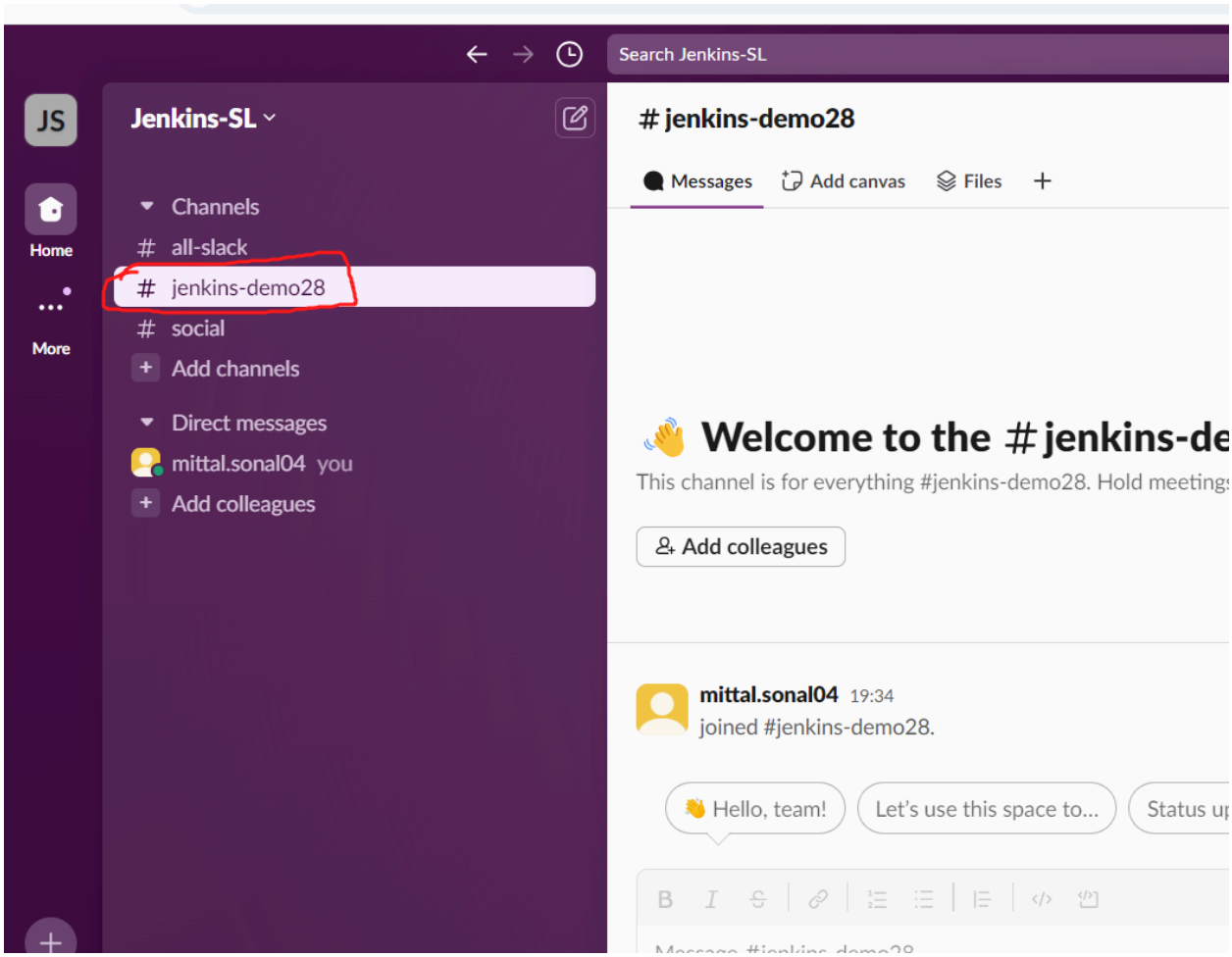
   What's your team working on right now? => give as Jenkins-demo ⇒ this will your channel name


JS
Jenkins-SL
Home
More
Step 1 of 5
What's the name of your company or team?
This will be the name of your Slack workspace – choose something that your team will recognise.
Jenkins-SL
Next


JS
Jenkins-SL
Direct messages
Home
More
Step 3 of 5
Who else is on the Jenkins-SL team?
Add colleagues by email
Add from Google Contacts
Example ellis@gmail.com, maria@gmail.com
Next
Copy invitation link
Skip this step

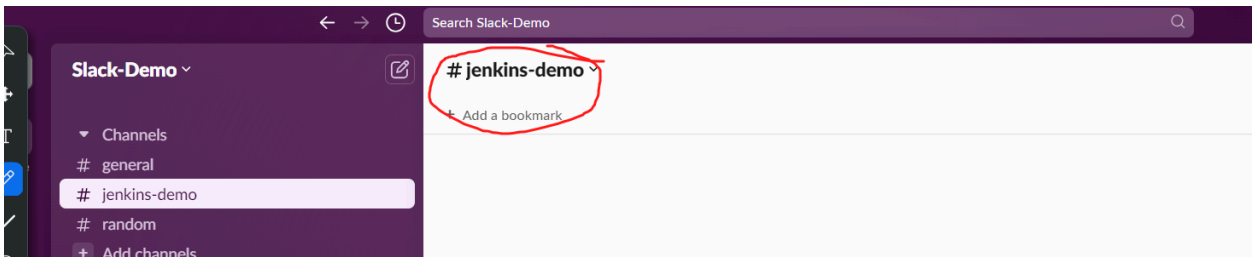

JS
Jenkins-SL
Channels
jenkins-demo28
Direct messages
Home
More
Step 4 of 5
What's your team working on right now?
This could be anything: a project, campaign, event or the deal you're trying to close.
Jenkins-demo28
66
Next


Jenkins-SL
Channels
jenkins-demo28
Direct messages
Choose a subscriptior
started:
Free
₹0
per person/month
Included with Free:
90 days of message history
1:1 audio/video conversations
Ten apps or integrations
Start with Free



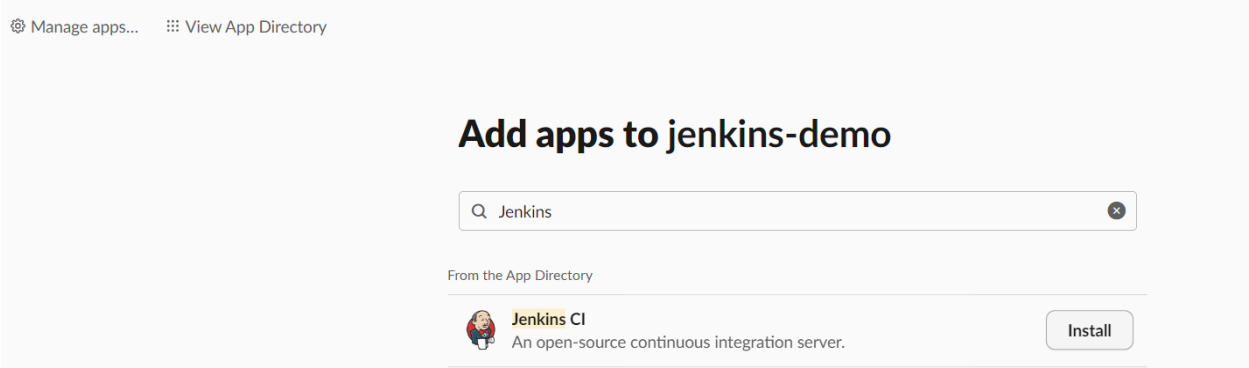
1. **Add jenkins CI tool to slack**

    - **Click on the channel drop down which is in the middle of the page**

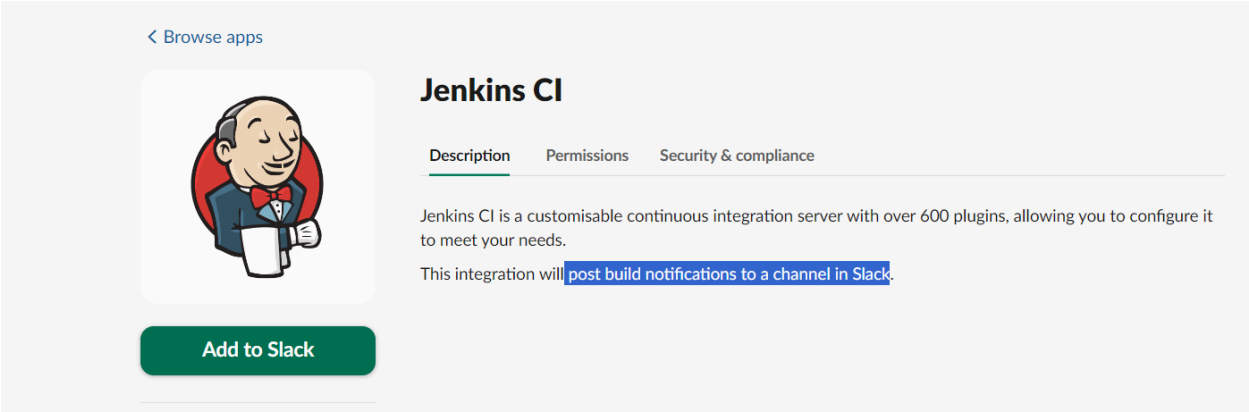


Click on Integrations → Click on add an ApP

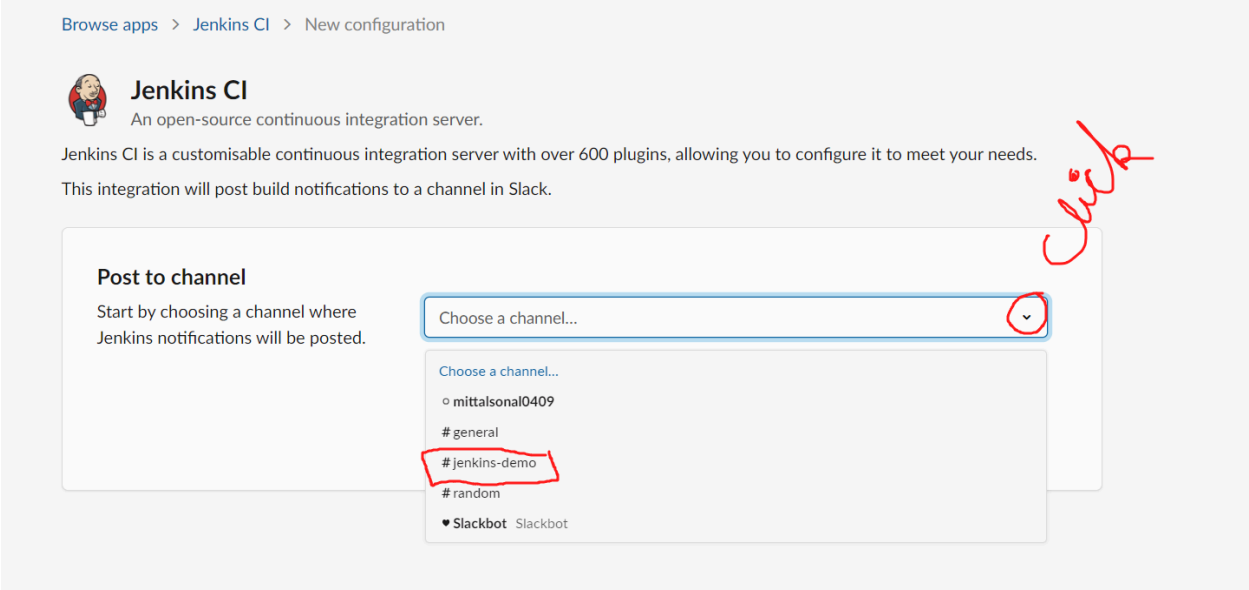
Search for Jenkins → Click on install



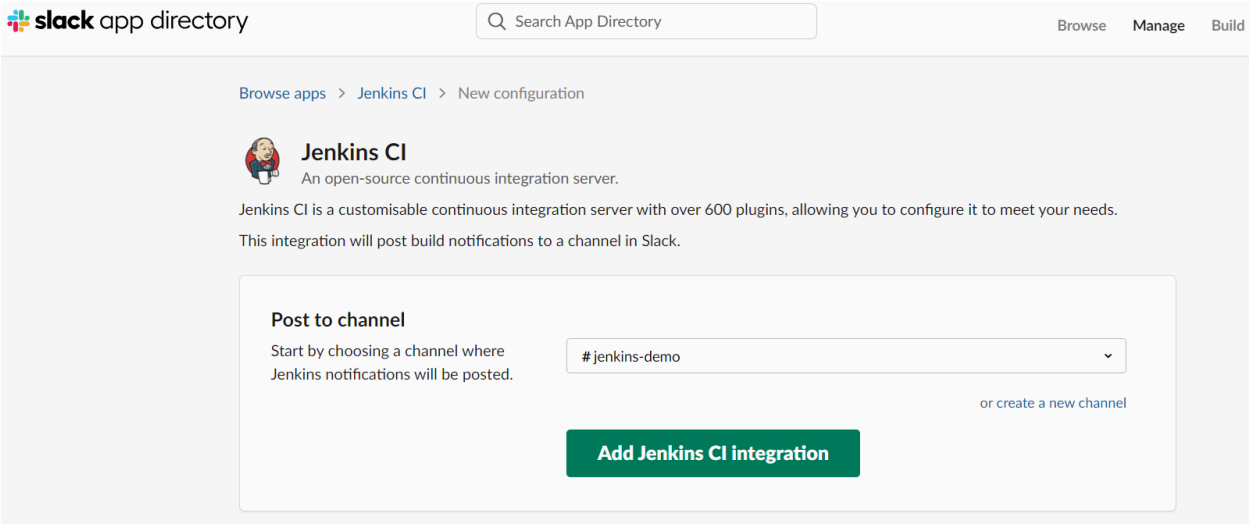
Click on Add to slack

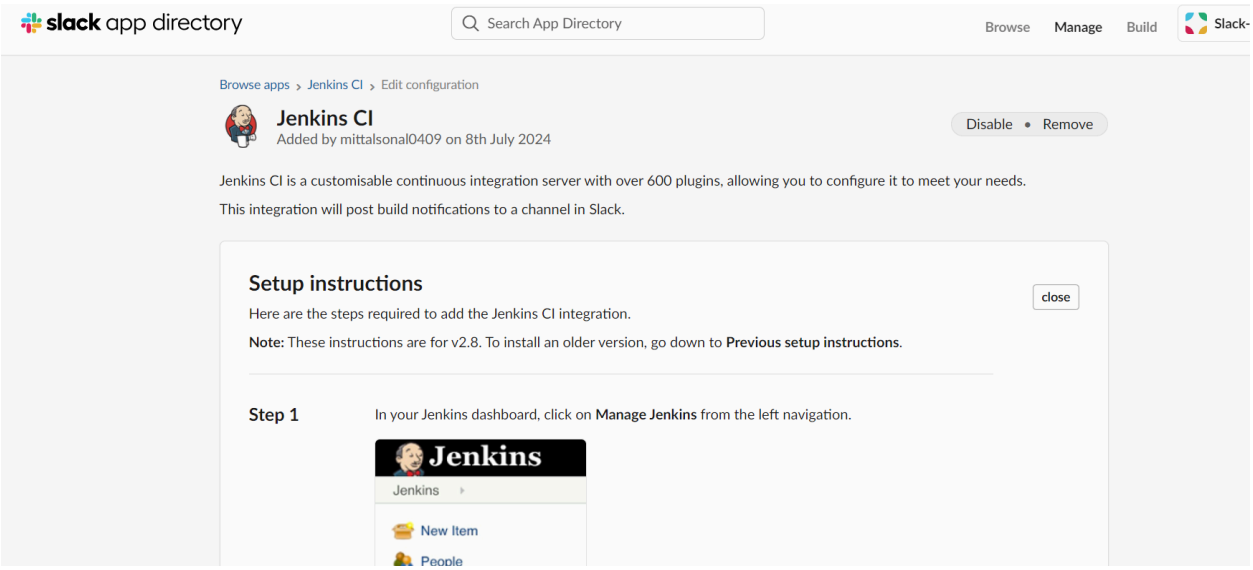


Select you channel name

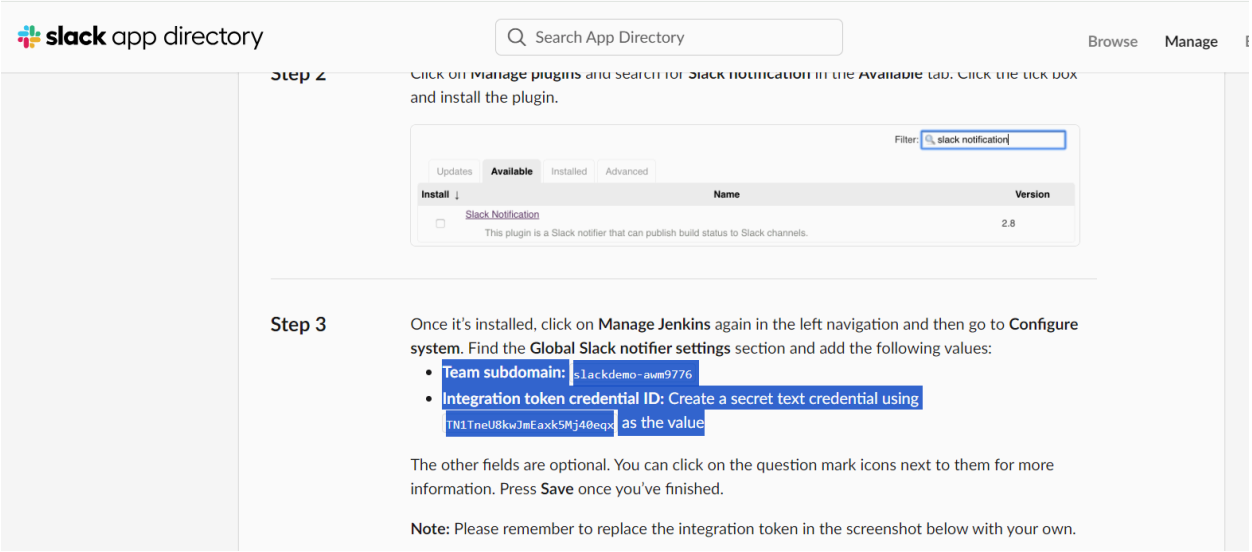


Click on add Jenkins CI integration



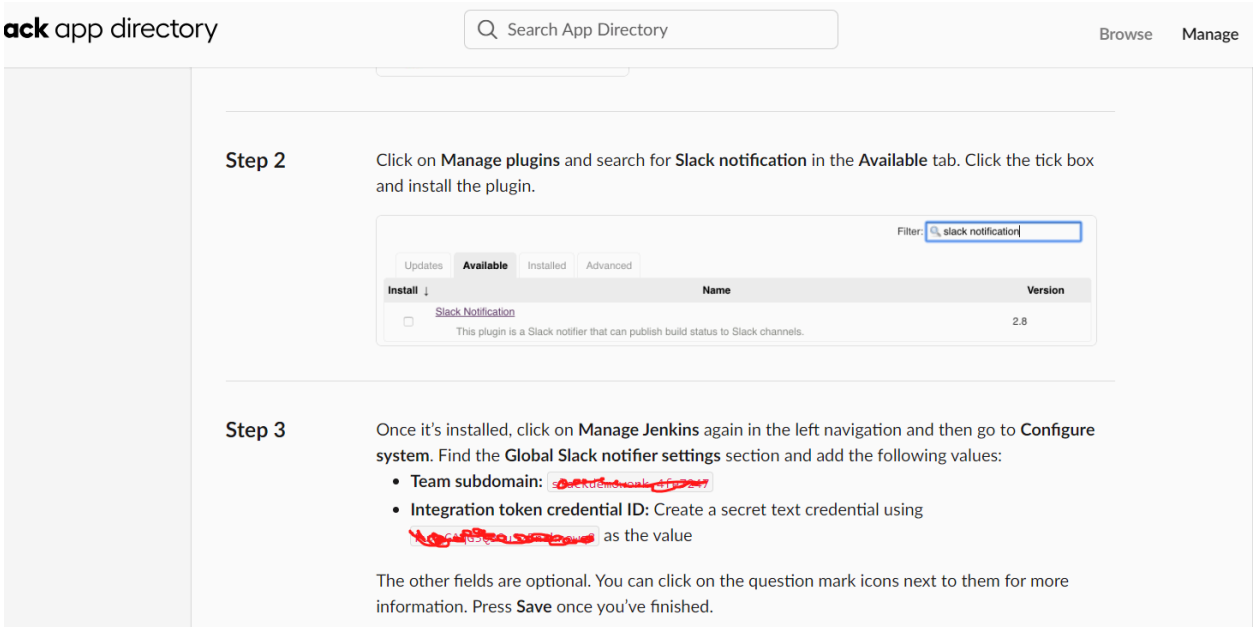


Copy the team domain name and token details



On the next page, scroll to step 3 and note down the below details:

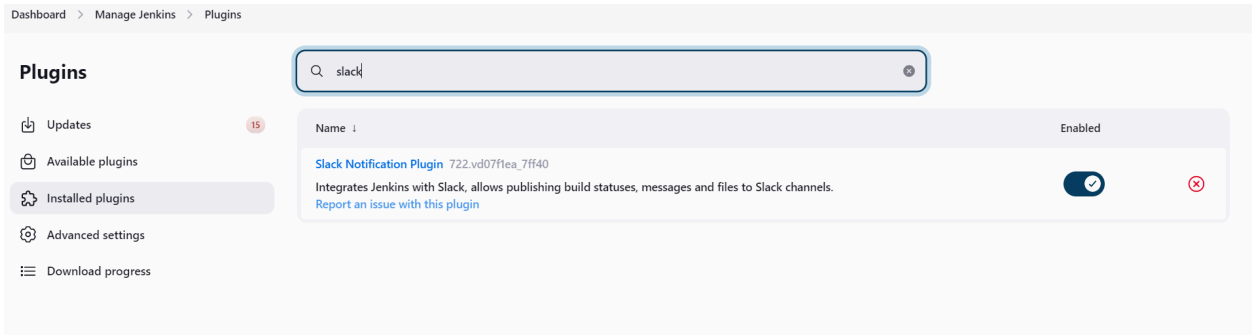
- **Team subdomain:**
- **Integration token credential ID:**



Install slack notification plugin on jenkins:

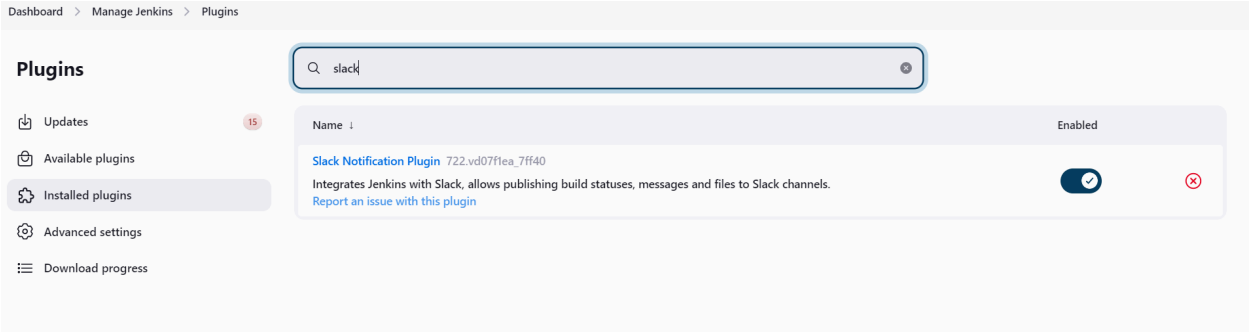
==========

Manage Jenkins → plugins → available plugins → slack notification → click on install button



Install slack notification plugin on jenkins:
==========

Manage Jenkins → plugins → available plugins → slack notification → click on install button



Setup slack on jenkins:
==========

Manage Jenkins → System -> Search for slack

In workspace add the subdomain name that we have copied from slack app directory(step3)

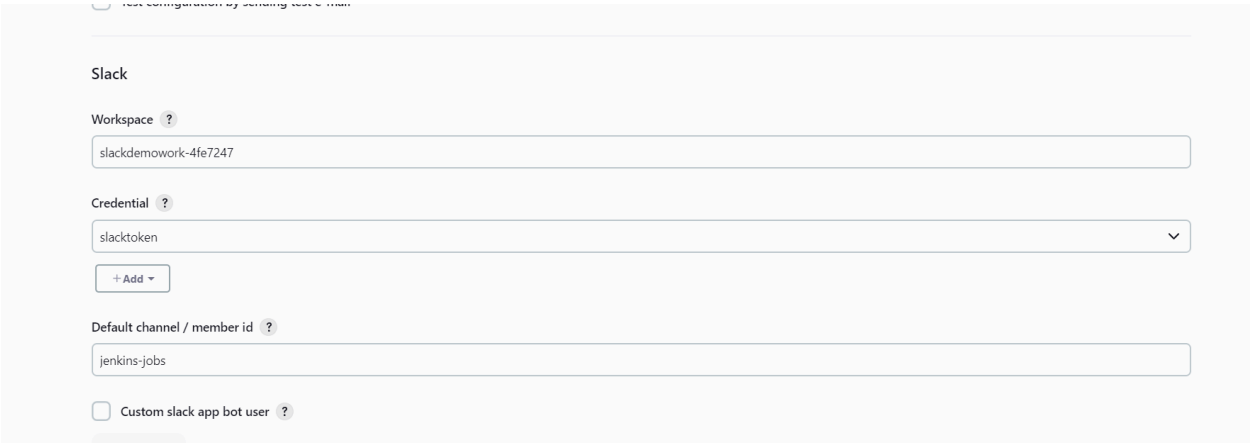
Create credentials to setup slack token

Credentisls kind = secret text


Jenkins Credentials Provider: Jenkins
Domain
Global credentials (unrestricted)
Kind
Username with password
Username with password
GitHub App
SSH Username with private key
Secret file
Secret text
X.509 Client Certificate
Certificate
Treat username as secret
Password


Jenkins Credentials Provider: Jenkins
Kind
Secret text
Scope
Global (Jenkins, nodes, items, all child items, etc)
Secret
ID
slacktoken
Description
slacktoken
Cancel
Add

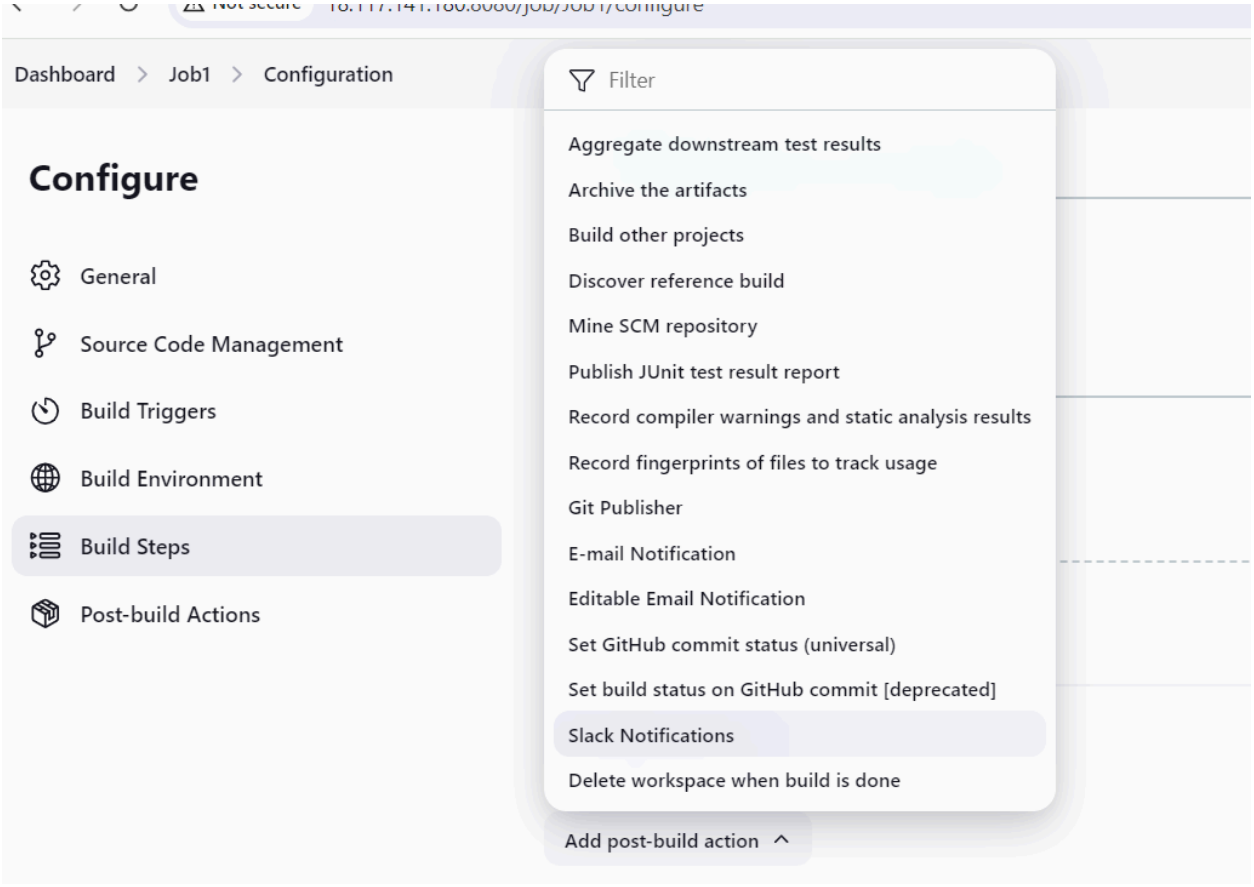
Enter the channel name which you have created on slack



Save the changes

Now take any freestyle job that is already existing

Clcik on the job name→ configure → go to post build actions



Save the job and check the notifications.

Now create pipeline with slack notification
New job → pipeline template→ add below code

```
pipeline{
agent any

stages{
  stage('Slack Notification'){
```

```
        steps{
            echo "Hello to Slack form pipeline"
        }
        post{
        success{
            slackSend channel: 'jenkins-demo28',
message: 'Pipeline Job Successfully Executed'
        }
        failure{
        slackSend channel: 'jenkins-demo28', message:
'Pipeline Job Failed'
        }
        }
    }
}
}
```

Save and run

==========
Day 7: 04 May 2025
==========
Agenda:
==========

- Continuous Integration with Github Actions
- Integrate github actions to trigger a Jenkins Job
- Master and worker node setup in jenkins

GITHUB ACTIONS:
==========

> It is a tool which is provided to us by GITHUB
> But it is not a version Control tool
> It is a very simple and an easy tool
> It is a tool that is in built on your github server
> We do not have to install it, we do not have to download or set it up.
> It is an open source tool, available for us on github
> We just have to log into github and start using it.
> The name of the tools is Actions
> No settings or no new credentials are required

What do we do in github actions:
==========

> It is a continuous integration tool
> on Actions tool we create workflows
> Workflows are set of tasks or actions that are executed on that github repository
> Github actions like jenkins can integrate with various devops tools to create CICD workflows
> GITHUB actions workflows = Jenkins pipeline

JENKINS	GITHUB ACTIONS
Has to be installed on windows, macc or linux machine	No need to install anywhere, it comes with github
Dependent on Java 17 or 21	No such dependency
We create pipelines, stages	we create workflows and we have jobs

Pipeline written based on groovy	Workflows are written using YAML
all the jobs are executed on jenkisn controller	GITHUb actions runs the workflow on its runners(VMs created automatically)
Jenkins is a global tool, it can work with any VC tool like github, bitbucket, gitlab	It is worked with github repository only
Can integrate with all devops tools	Can integrate with all devops tools
jenkins provides 1000+ plugins are available for integration	github actions provide same feature but using ready made actions

Key Elements of GITHUB ACTIONS:
==========

Workflow:

- Whatever tasks we have to automate for the repository we will write as a workflow.
- Single repository can have many workflows
- A workflow consist of jobs
- A workflow consist of event or trigger when the workflow has to be executed
- We can run a workflow manually or whenever an event occurs on the repository
- A workflow can have local variables, environment variables & secret variables

Jobs:

- These are the most essential elements of a workflow
- You can have multiple jobs in 1 workflow.
- Jobs consist of steps -> task/command/script to be executed
- In the workflow Jobs are executed parallelly, however we can set dependency for jobs to run in sequence.
- A jobs can use the variables

Runners:

- Just like we have agents in jenkins pipeline, here we have runners
- Readymade virtual machines of OS mac, windows and linux
- These runners are VMs which are provided by Azure cloud in the background
- Runners are VM where the workflow will be executed
- If you want you can create your own VM and ask github actions to run the workflow on your VM

Runners are the machines that execute jobs in a GitHub Actions workflow. For example, a runner can clone your repository locally, install testing software, and then run commands that evaluate your code.

GitHub provides runners that you can use to run your jobs, or you can host your own runners. Each GitHub-hosted runner is a new virtual machine (VM) hosted by GitHub with the runner application and other tools preinstalled, and is available with Ubuntu Linux, Windows, or macOS operating systems. When you use a GitHub-hosted runner, machine maintenance and upgrades are taken care of for you.

==========

Steps:

- The key component of jobs is steps
- Steps are nothing but command/script to be executed
- Steps will always run in sequence

Structure of github actions workflow:

==========

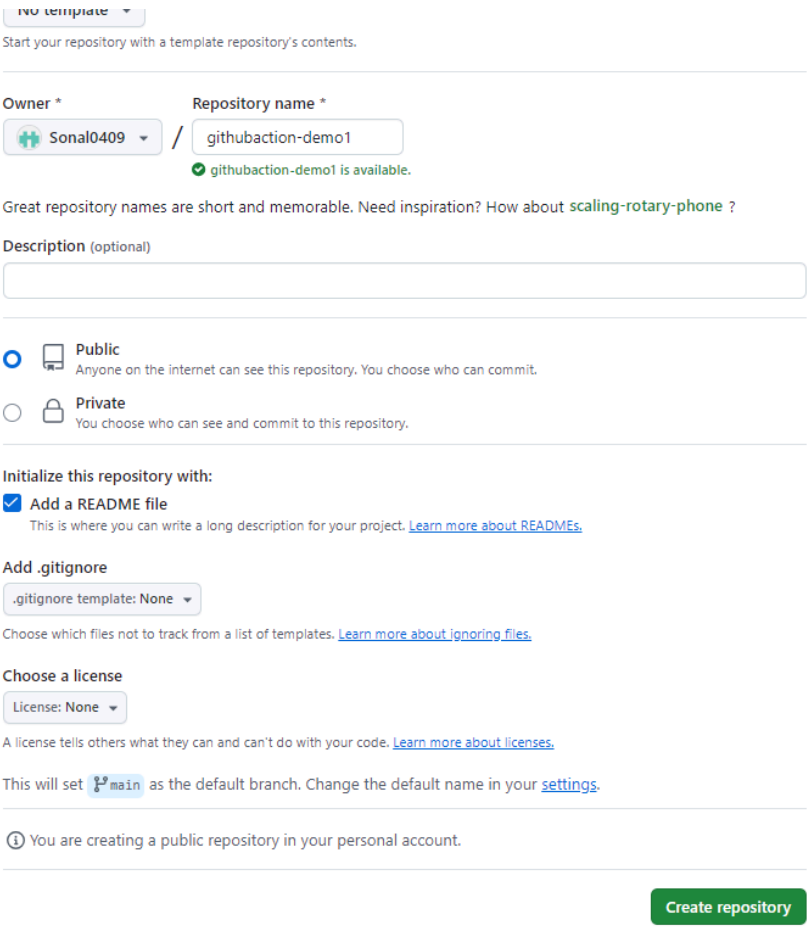
The github action code is written in YAML

```
Write YAML syntax:
====================
A file with YAML code will have extension as .yml or .yaml

key: value

A key can store a single value, list of values or a map(key: value)
In general, the tool provides the key and value is provided by user
values can be string, number, boolean

Example 1: Key storing a single value
=======================================
---
Company: Simplilearn
Trainer: Sonal
Training: Jenkins
Days: Weekend
Time: 7PM IST


Example 2: A key storing a list of values
=======================================
---

Company: Simplilearn
Trainers:
  - Sonal
  - Ravi
  - Jack
  - John
Trainings:
  - Jenkins
  - github
```

```
  - devops
  - aws
Days:
  - Weekend
  - Weekdays
Time:
  - 7PM IST
  - 9AM IST


Example 3: A key storing  again a key and value(map)

---
 # it is called as an OBJECT in YAML => set of key and value(maps)
---

Company: Simplilearn
Trainers:   # this is a key
  - name: Sonal  # this is a map
    phone: 2353546   # this is a map
    email: sonal@gmail.com   # this is a map
  - name: Ravi  # this is a map
    phone: 2353546   # this is a map
    email: ravi@gmail.com   # this is a map
  - name: Marc  # this is a map
    phone: 2353546   # this is a map
    email: marc@gmail.com   # this is a map
```

```
# Structure of GITHUB Actions workflow
# =====================================

# keys are provided to you by github actions
---
name: <name of the workflow> # you can give
on: # value given by github < on what even shoudl the workflow be trigger
- manually or on a commit>
jobs:
  <name_of_the_job>: # you can give
  runs-on: <give the runner OS> # value given by github
  env: <give env or sectret variables> # you can give, use github action
variables
  steps:
    - name: <give a name to the step > # you can give
      run: <run a command/script> # you can give
    - name: <give a name to the step > # you can give
      run: <run a command/script> # you can give
```

Create a github repo



Click on create repository

On the repository→ click on Actions tab → click on set up a workflow yourself

Add below code:

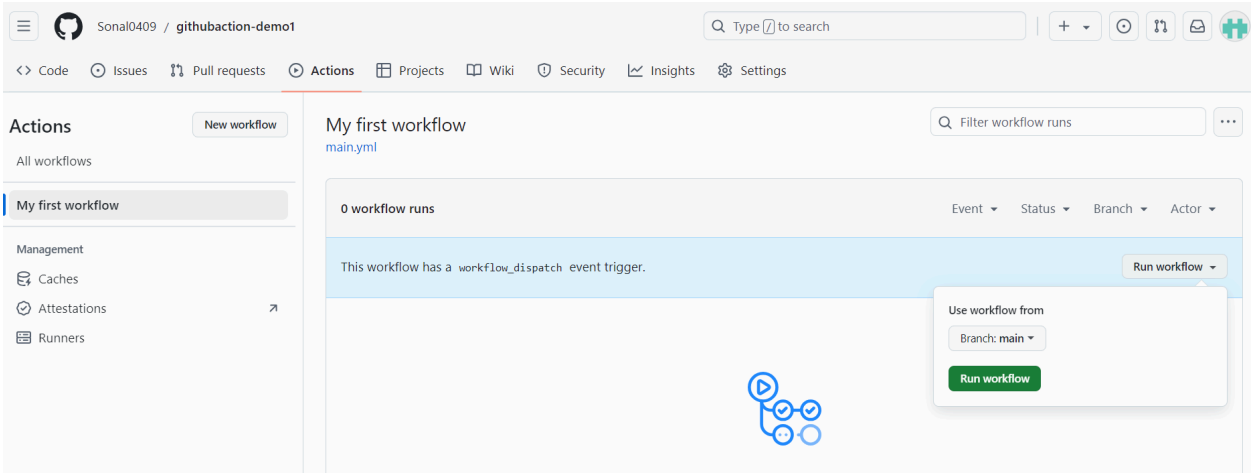
```
name: My first workflow
on: workflow_dispatch
jobs:
  firstjob:
```

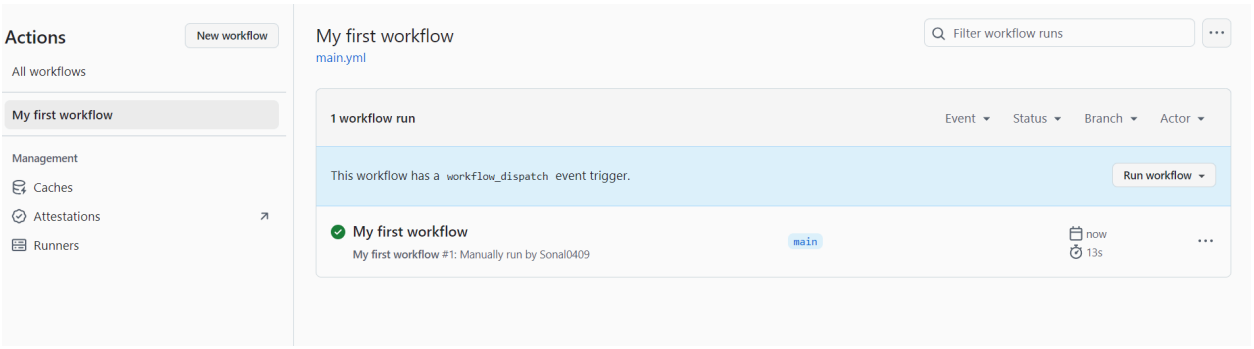
```
runs-on: ubuntu-latest
steps:
  - name: Print greetings!
    run: echo "Hello All"
  - name: Run multiple commands
    run: |
      echo "Hello Github"
      echo "This is job1"
```

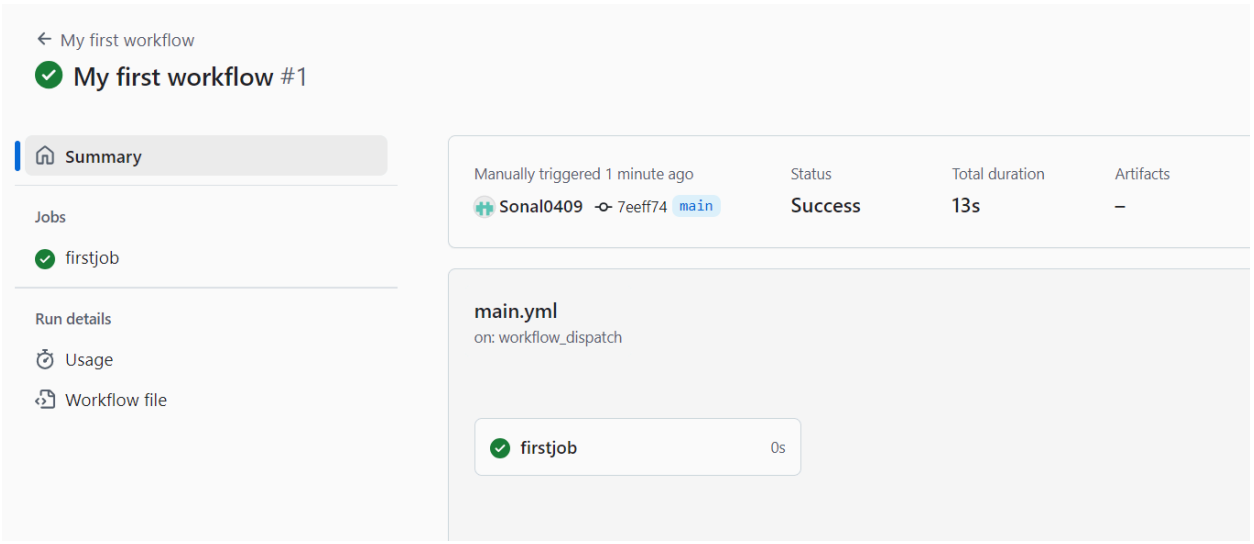
Save the file.

Now click on Actions TAB → **on the left side of the page you will see your workflow name**→ On the right side you will see Run workflow button→ click on it -> click on run workflow

Refresh the page to see the workflow







Demo 2:

==========

```
name: My Second workflow
on: workflow_dispatch
jobs:
  firstjob:
    runs-on: ubuntu-latest
    steps:
      - name: Print a message
        run: echo "Hello All"
```

```
  secondjob:
    runs-on: ubuntu-22.04
    steps:
      - name: Run multiple commands
        run: |
          echo "Hello from Actions"
          echo "Executed Job2 of the workflow"
```

Demo 3:

==========

Fork this repository:

https://github.com/demo1orgtoday/MavenBuild-SL.git

Go to .github/workflows → click on the main.yml file → click on edit button → remove all the code → add the below code

```
name: CICD using Maven and tomcat
on:
 push:  # run when there is commit to repo
 workflow_dispatch:  # run manually
jobs:
 CICDjob:
   runs-on: ubuntu-latest
   steps:
```

```
- name: Clone the repo on ubuntu server
  uses: actions/checkout@v4
- name: Install Java and maven on ubuntu server
  uses: actions/setup-java@v4
  with:
   distribution: 'temurin'
   java-version: '11'
   cache: 'maven'
- name: Build the code
  run: mvn package
```

Save the file and click on actions

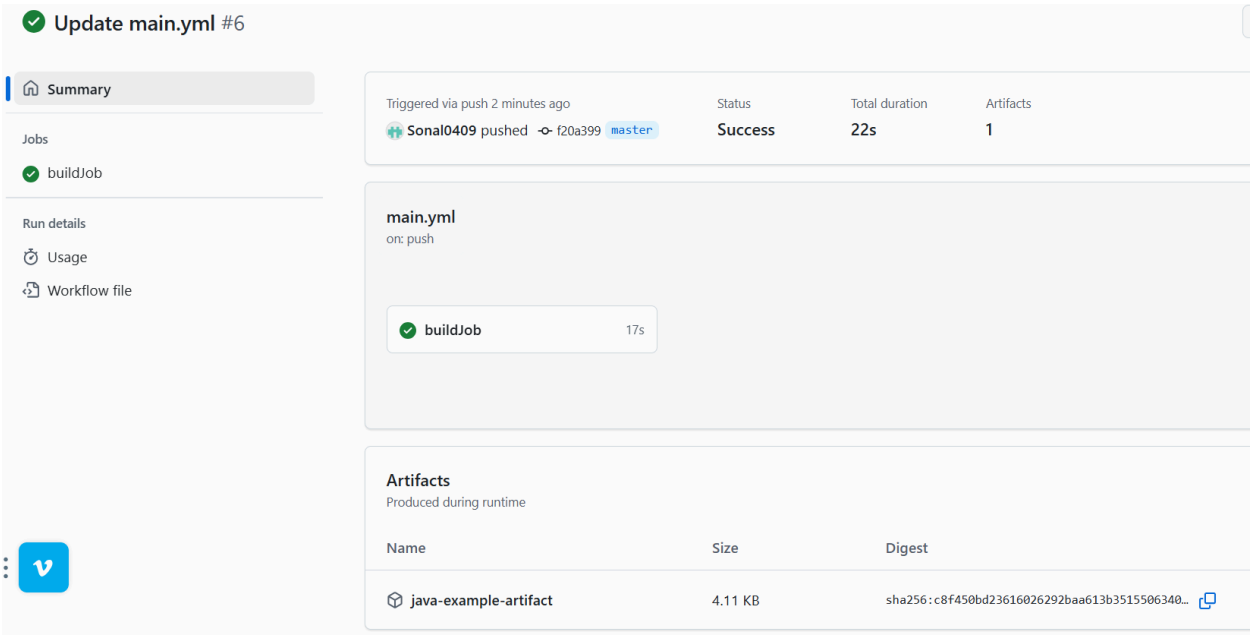
Build and upload the artifact to github from VM

==========

```yaml
name: CI using maven and Jenkins
on:
 push:  # run the workflow on every commit
 workflow_dispatch: # run the workflow maually
jobs:
  buildJob:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code on the ubuntu server
        uses: actions/checkout@v4
      - name: Install Java and Maven on the ubuntu server
        uses: actions/setup-java@v4
        with:
         distribution: 'temurin'
         java-version: '11'
```

```
          cache: 'maven'
      - name: Test the code
        run: mvn test
      - name: Build the code
        run: mvn package
      - name: Get the artifact and upload on github repo
        uses: actions/upload-artifact@v4
        with:
          name: java-example-artifact
          path:
/home/runner/work/MavenBuild-SL/MavenBuild-SL/target/java-example.war
      # this is directory path on VM
```

Save and check the workflow execution



Demo 5:

==========

This also explains and completed your Course end project- 1

```yaml
name: CI using maven and Jenkins
on:
 push:  # run the workflow on every commit
 workflow_dispatch: # run the workflow maually
jobs:
  buildJob:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code on the ubuntu server
        uses: actions/checkout@v4
      - name: Install Java and Maven on the ubuntu server
        uses: actions/setup-java@v4
        with:
         distribution: 'temurin'
         java-version: '11'
         cache: 'maven'
      - name: Test the code
        run: mvn test
      - name: Build the code
        run: mvn package
      - name: Get the artifact and upload on github repo
        uses: actions/upload-artifact@v4
        with:
         name: java-example-artifact
         path:
/home/runner/work/MavenBuild-SL/MavenBuild-SL/target/java-example.war
      - name: Connect to LAB and run curl command to trigger a Job
        uses: cross-the-world/ssh-scp-ssh-pipelines@latest
        with:
          host: '65.2.111.206' # change to your lab server public IP
          user: 'labuser'
          pass: 'Nuvelabs123$'
          port: 22
          connect_timeout: 10s
          first_ssh: |
            sudo curl -l -u admin:Root123$
http://localhost:8080/job/Build-Job/build?token=token1
          scp: |
           './target/*.war' => /tmp
          last_ssh:
```

```
ls -al /tmp
```

==========

Agenda: DevSecOps integration with Jenkins

==========

What is DevSecOps?

==========

Security testing:

==========

- This is a non functional testing
- Type of testing where we check if the application is vulnerable to cyber attacks or not.
- Checks the impact of malicious or unexpected inputs on the application.
- Will provide evidence that the system is safe or not.

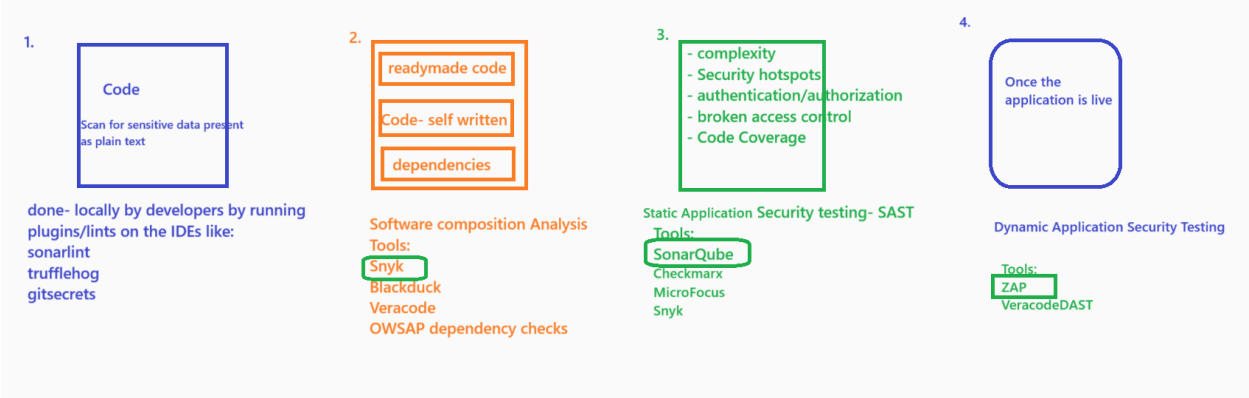
The main focus is:

==========

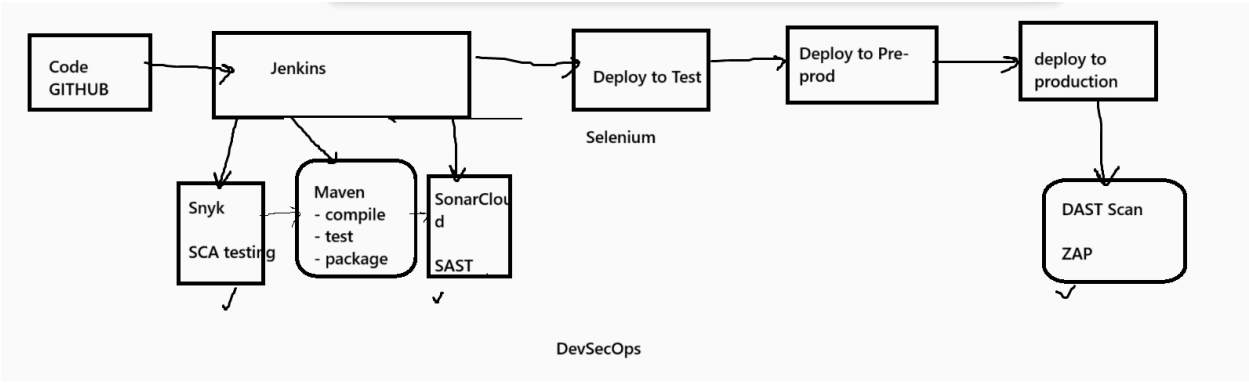
- To perform activities just like a cyber attack and observe how the application responds
- Identify the risks involved due to a vulnerability or threat
- When the application is delivered there should be less risk of security breach.

Types of Security testing:

==========



DevSecOps Pipeline



Setup jenkins on the Lab

==========

Go to browser of the Lab
Give URL as : localhost:8080

Username: admin
Password: Root123$

You will be on jenkins dashboard

For the setup to be complete, we have to update the jenkins plugins

Go to manage jenkins → Scroll down to Plugins

Click on updates:

Select the checkbox for updates and click on update button

Give few mins for updates to complete

Now reset jenkins

Scroll down and click on checkbox to restart jenkins server

==========

Demo: SoanrQube Integration with Jenkins
==========

It's an open source tool

Sonar’s static application security testing (SAST) engine detects security vulnerabilities in your code and guides you through resolution before you build and test your application. With SAST, you can achieve robust application security and compliance for complex projects.

SonarQube Server includes a powerful secrets detection tool, one of the most comprehensive solutions for detecting and removing secrets in code. Together with SonarQube for IDE, it prevents secrets from leaking out and becoming a serious security breach.

SonarQube Server helps you comply with common code security standards, such as the NIST SSDF, OWASP, CWE, STIG, and CASA. Your code is automatically checked for vulnerabilities and provides reports on how your code stands against these standards.

Your code is an asset. SonarQube helps you realize the complete value of your development efforts. By analyzing your codebase, finding real issues, and providing guidance on resolving them quickly, you can transform your code investments in to business outcomes.

Features:

- Integrated with GitHub Actions, GitLab CI/CD, Azure Pipelines, Bitbucket Pipelines, and Jenkins to automate code reviews and show code health status where you work at every step.
- Use Sonarqube by downloading it, as a container or as Sonar cloud
- Industry-leading accuracy maximizes signal and minimizes noise while reducing time-draining work. Receive actionable code health metrics in minutes instead of hours.
- Sonarqube can be used to perform code review and code coverage for more than 26 programming language
- Automate code vulnerabilty reviews for all code: open source, developer-written, and AI-generated. Unrivaled security detection uncovers deeply hidden security issues.
- Sonarqube IDE: Find and remediate issues in real-time as you code with SonarQube for IDE. Follow your coding policies in the IDE when in connected mode with SonarQube Server.
- Perform automated code reviews as required by every compliance standard. SonarQube's detailed reports help you comply with common standards such as OWASP.

- Prevent code from reaching production that doesn't meet your policies with SonarQube quality gates. Eliminate issues in human-written and AI code, cutting late remediation costs.

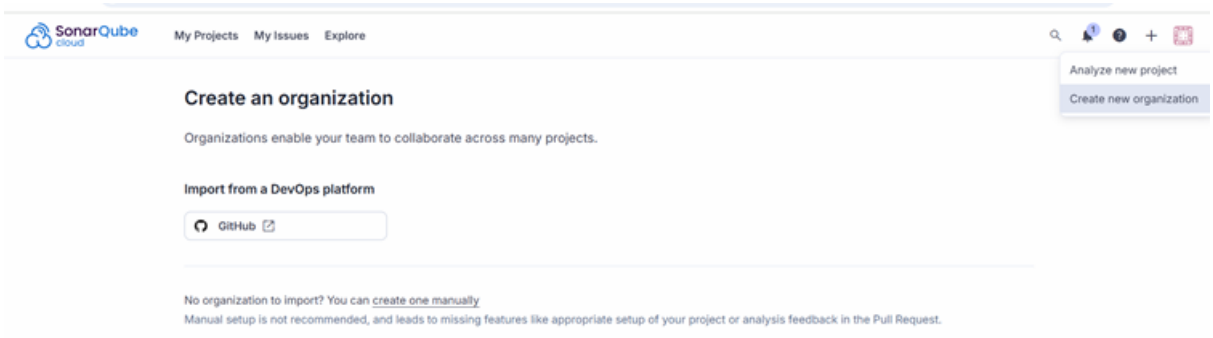
==========

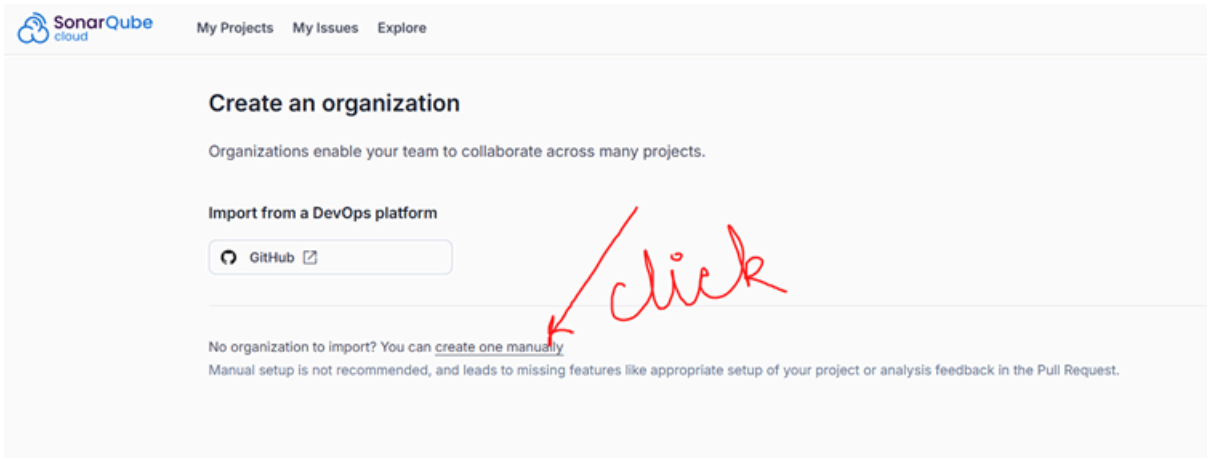
Demo:

Loginto sonarcloud with github

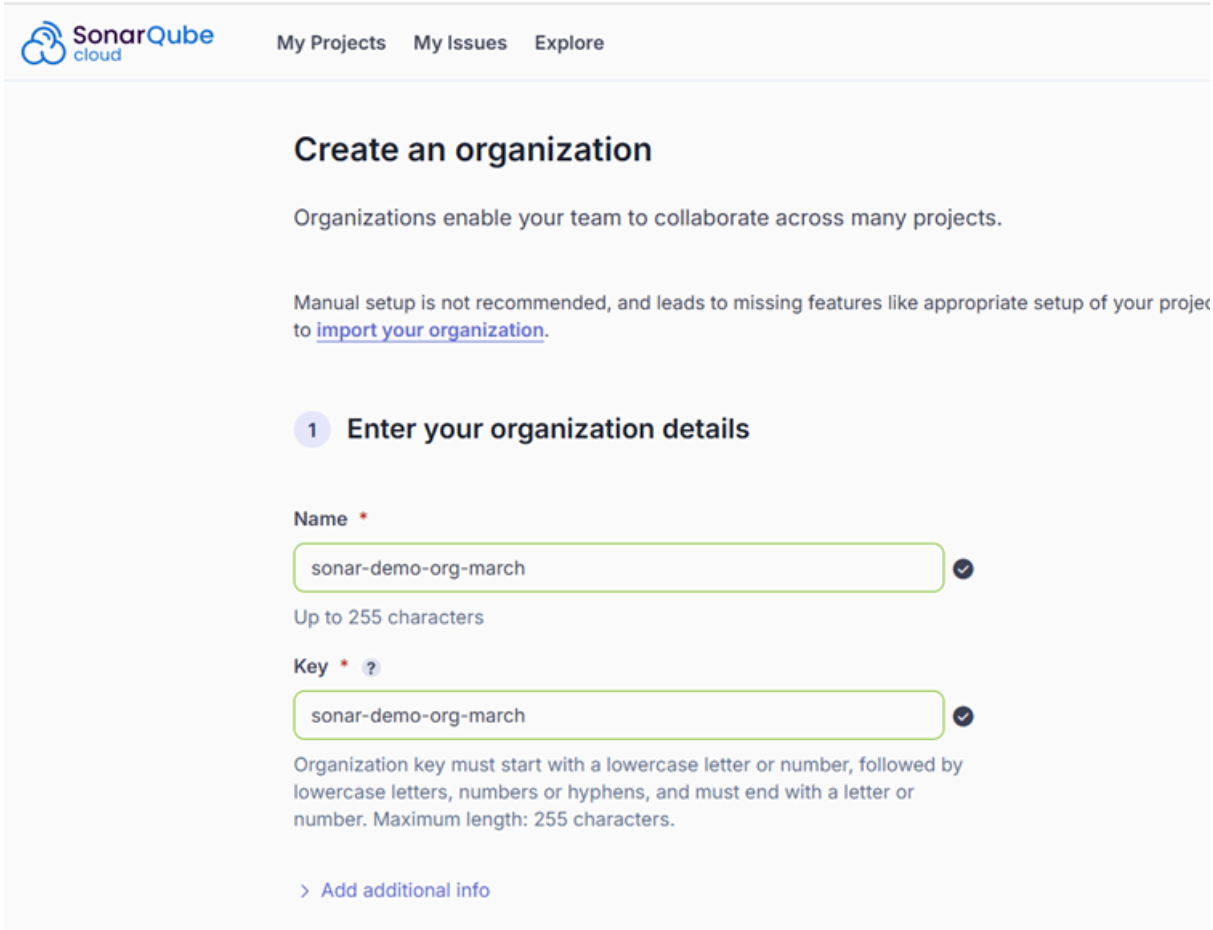
You will be on sonarcloud dashboard

Create an organization in sonar cloud

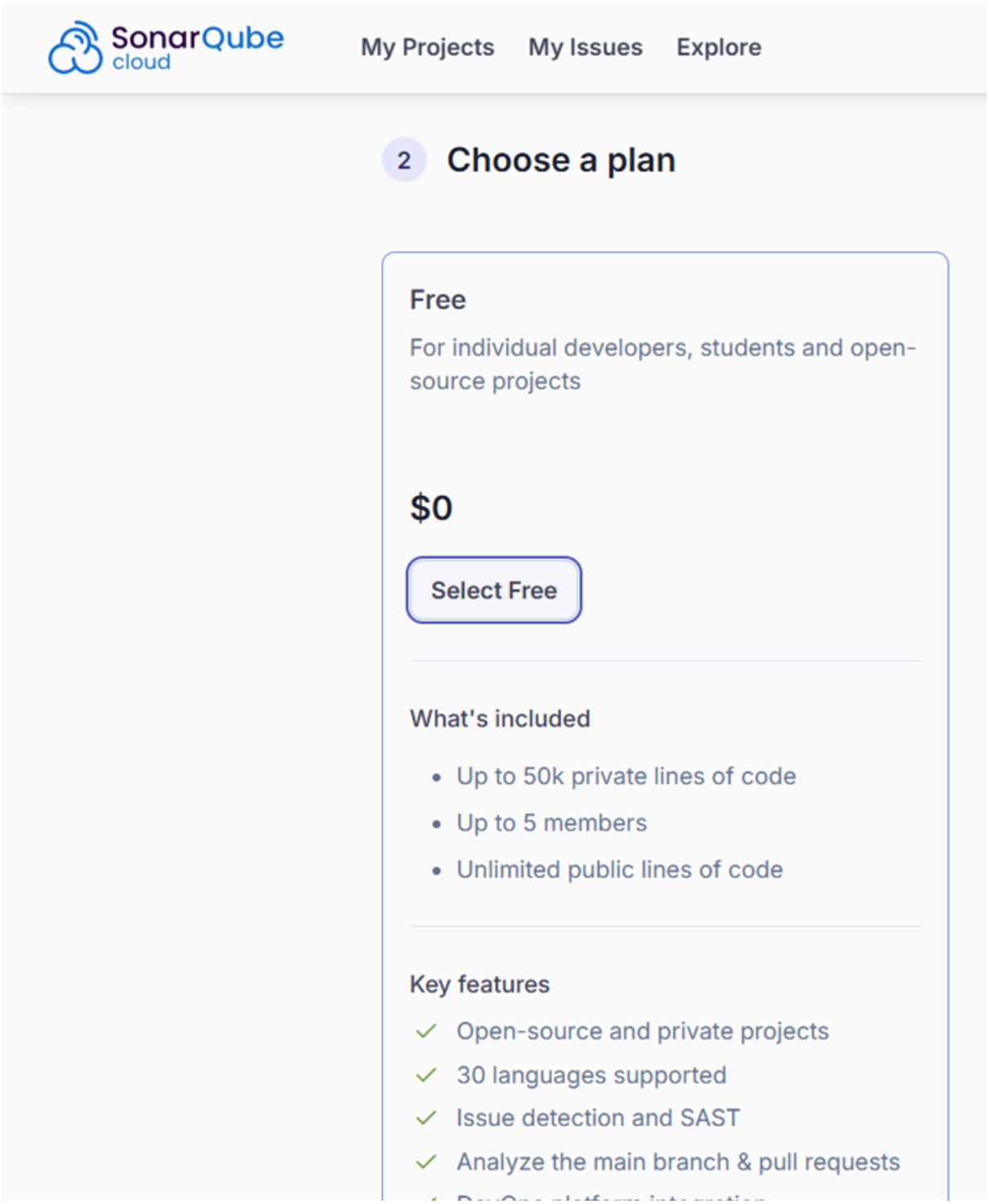




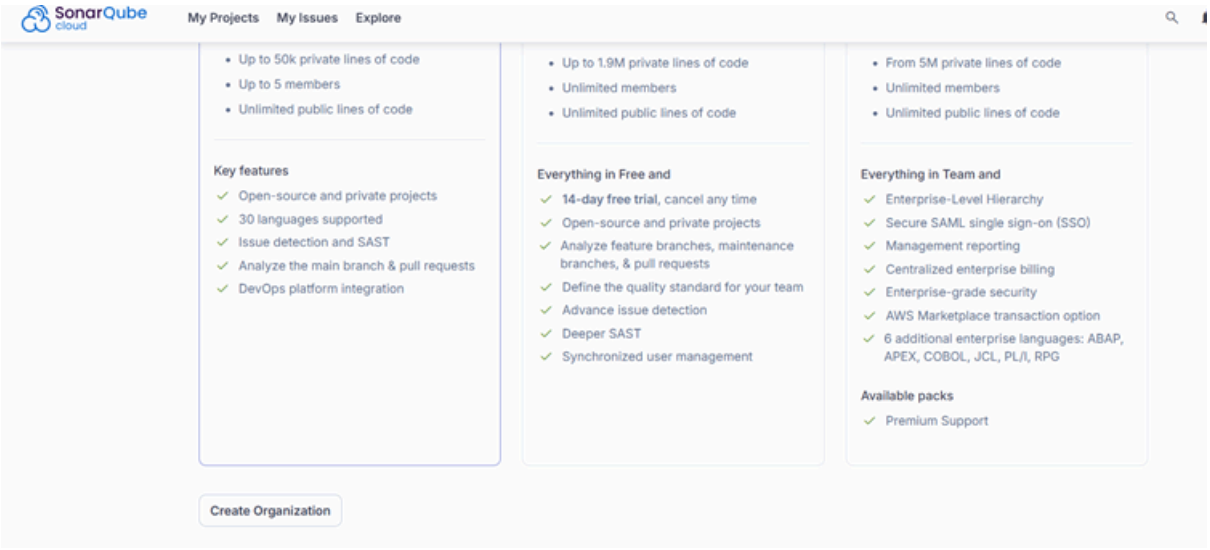
Give a unique org name



Select the free plan

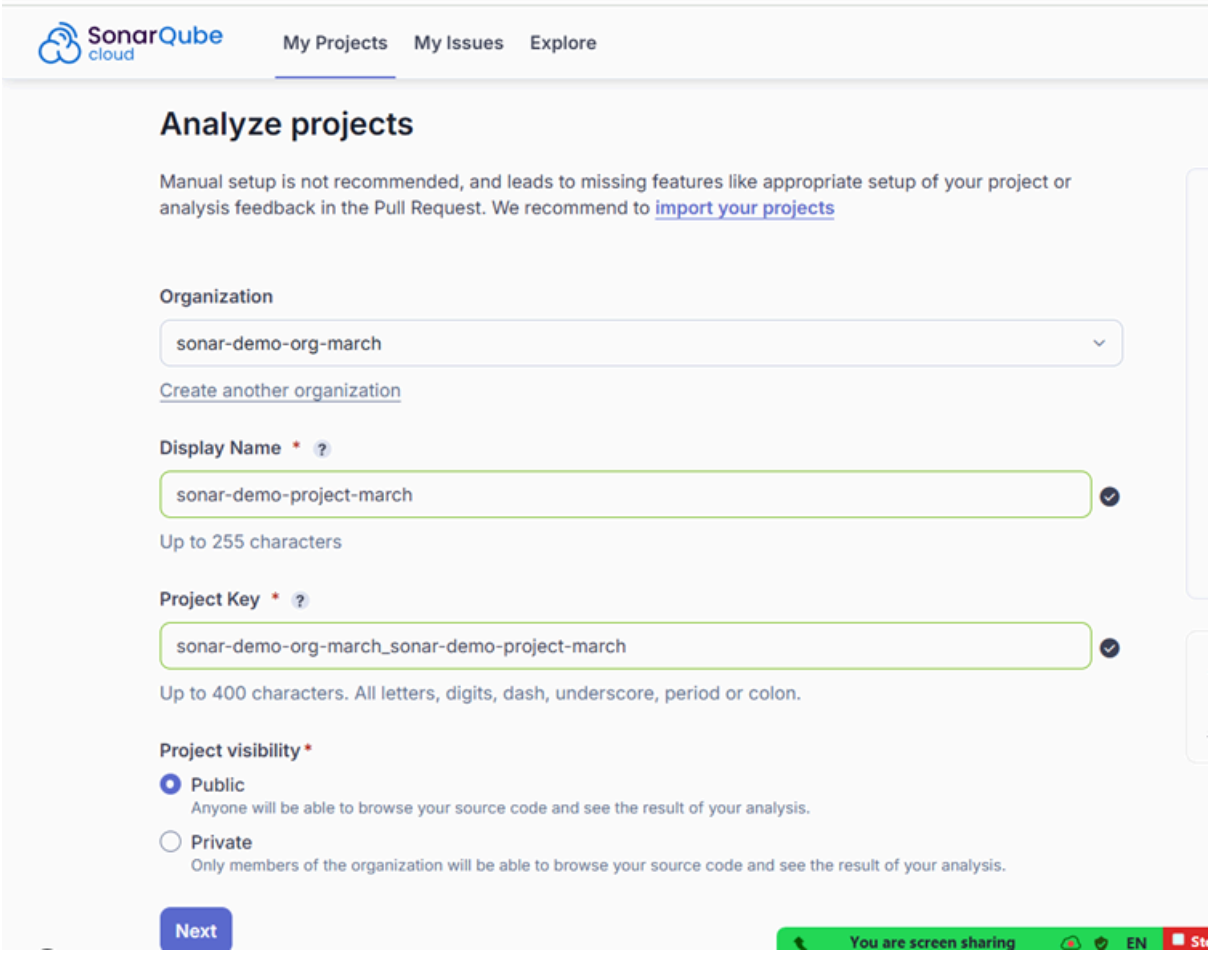


Scroll down and click on create organization

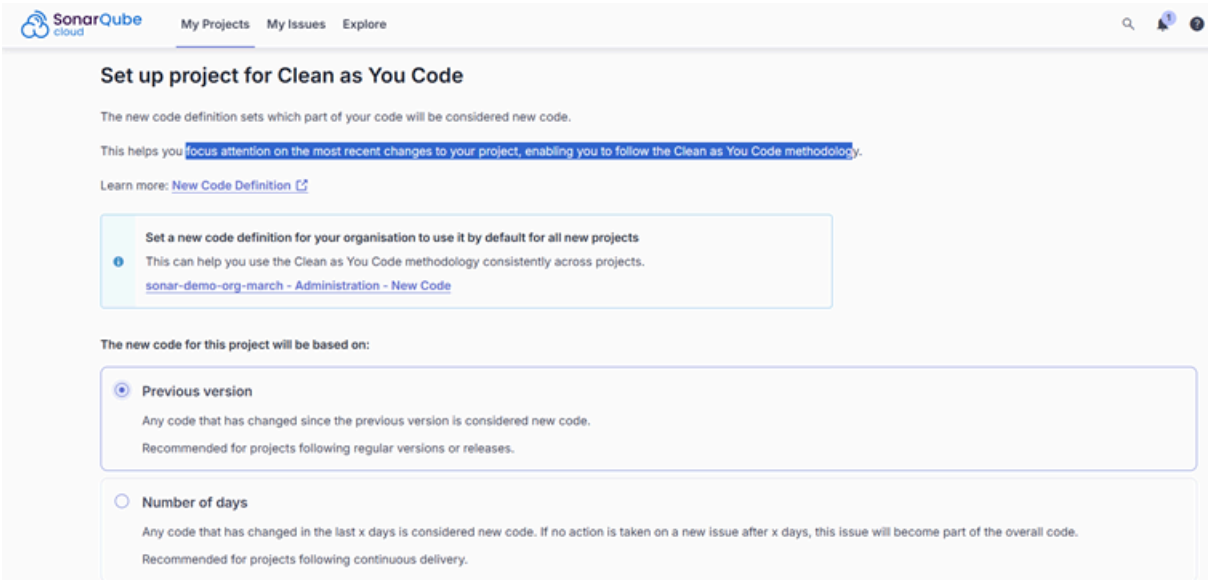


Step 3: Create a project in the organization

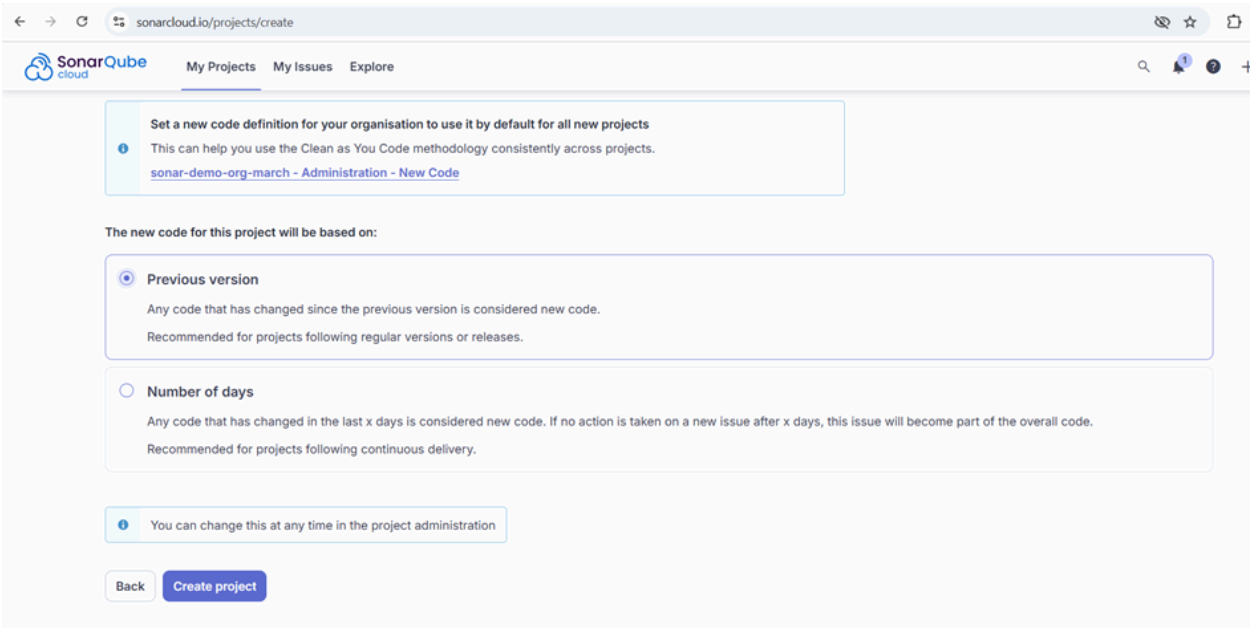
Click on analyze project button and enter below details



Press next and select previous version



Click on create project



Now to go LAB → go Jenkins dashboard

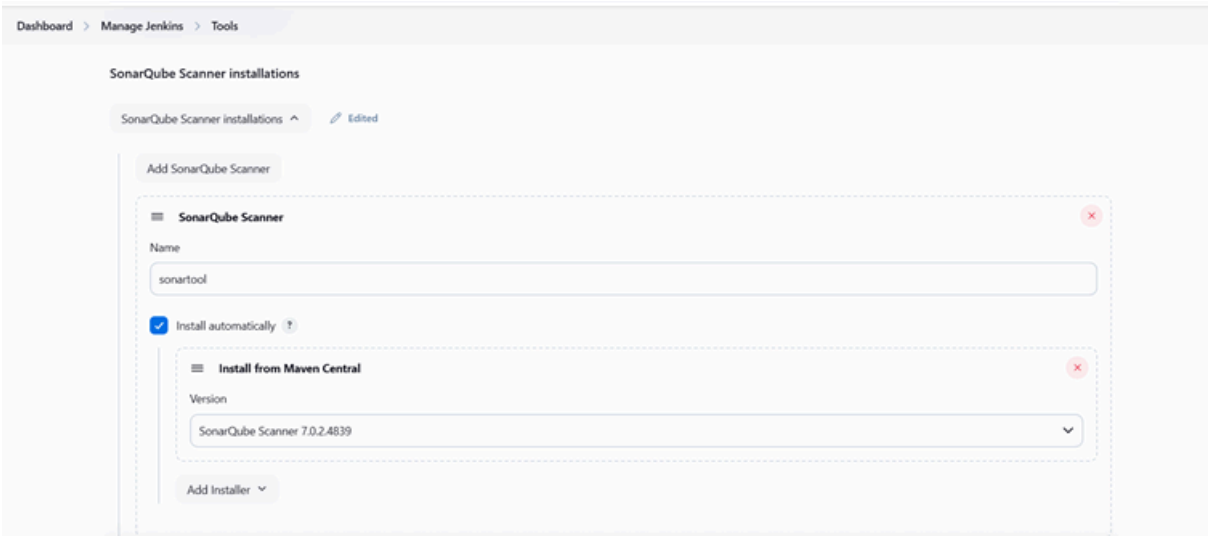
Download the sonar cloud scanner plugin

Go to manage Jenkins à go to Plugins à available plugins à search for SonarQube Scanner à select it and click on install

After installation

Setup Sonar tool on Jenkins

Manage Jenkins à tools à scroll down to **SonarQube Scanner installations** -> click on Add SonarQube scanner -> add below details as same. -> save the page

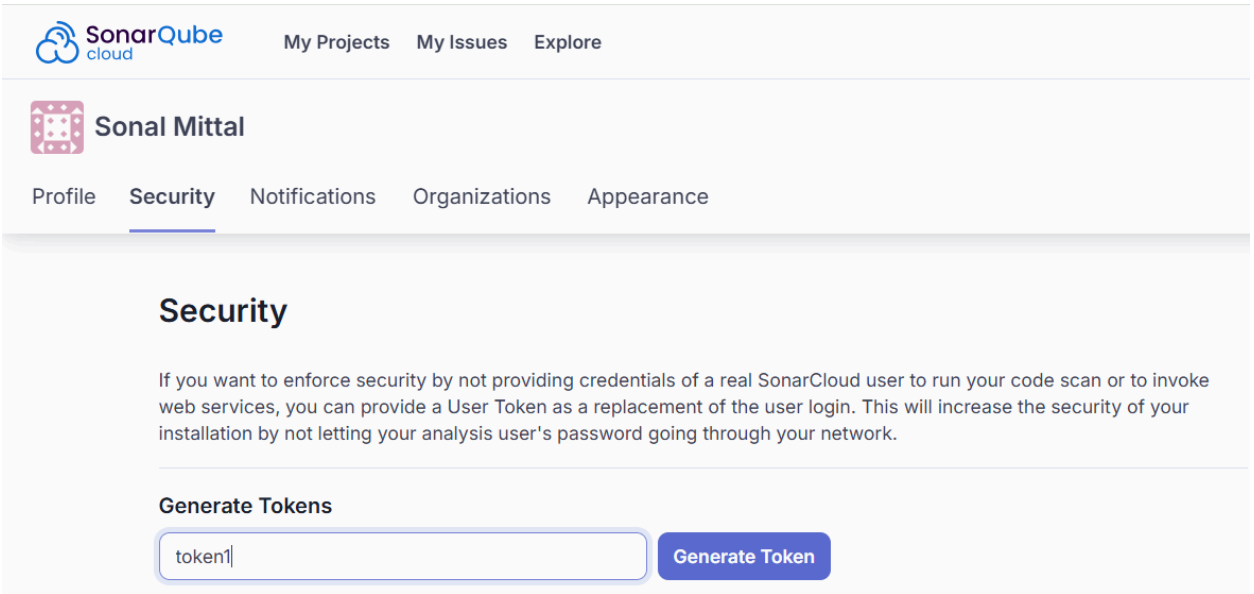


Save this page

Next steps:

Go to : https://sonarcloud.io/account/security

Enter a token name → click on generate token



Copy the token :

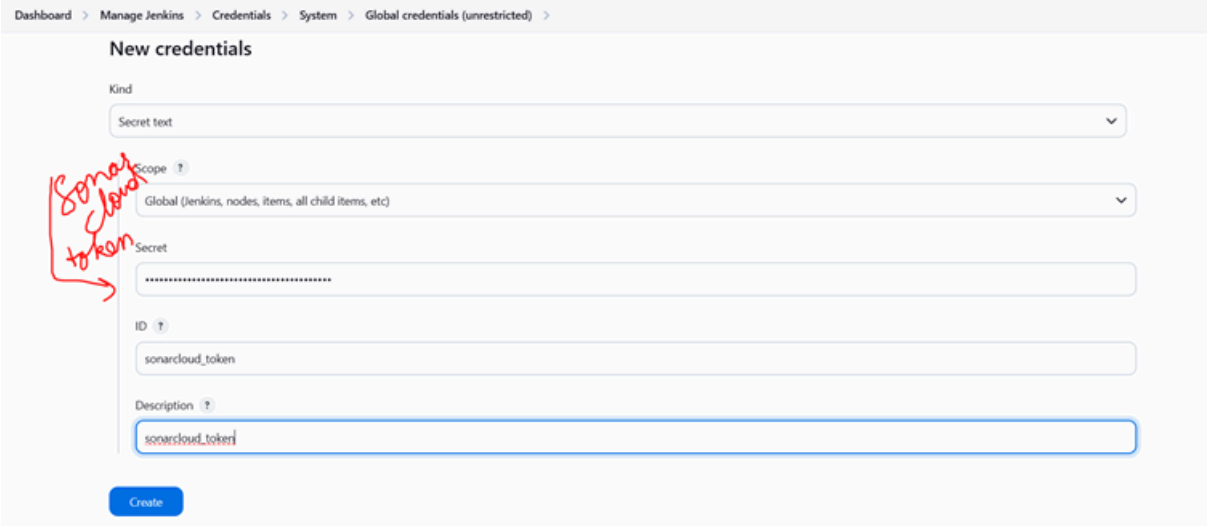
7edf3ce6bf08f0b22c0f16e8ef264ca029060f47

Go back to jenkins:

==========

Go to Jenkins server and Create a credentials to store the sonar token

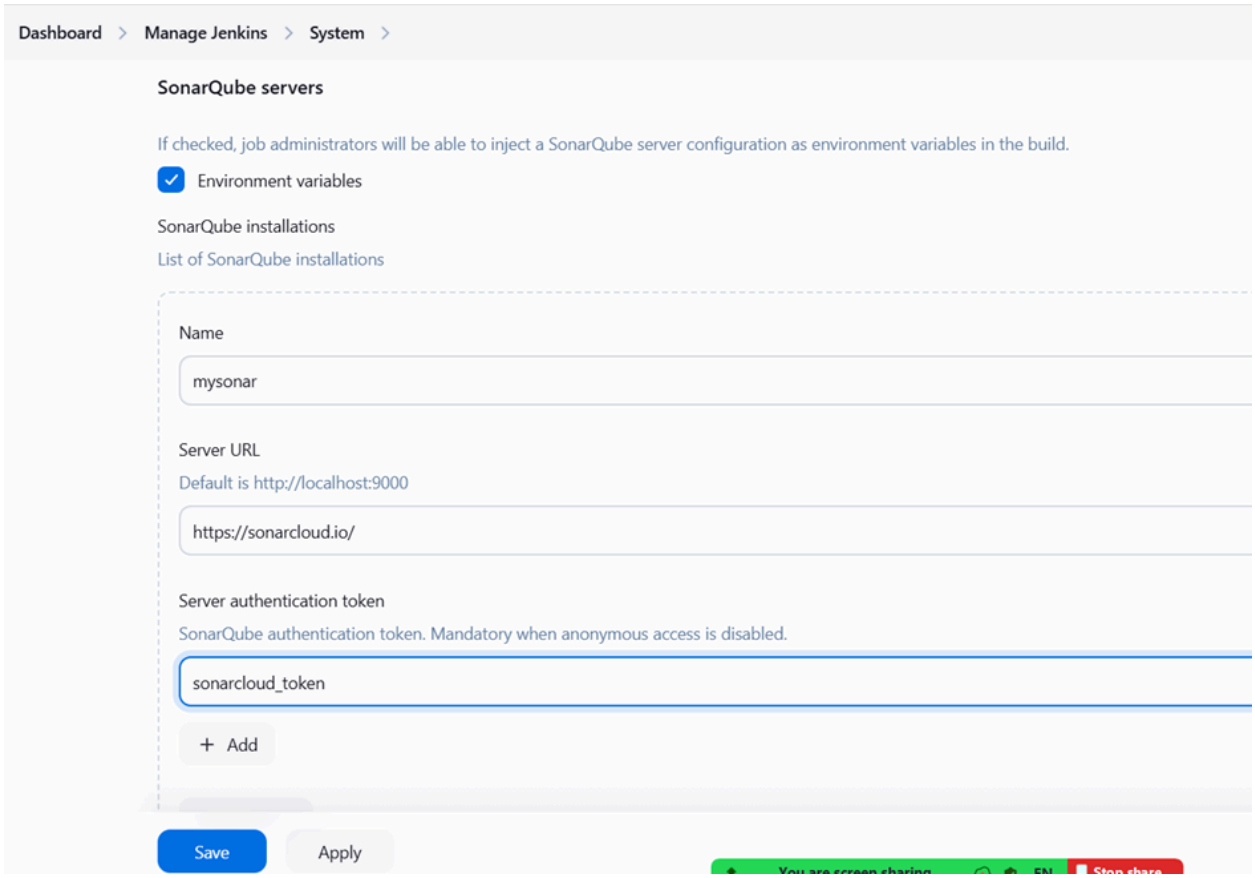
Go to manage Jenkins à go to credentials à click on global -> click on Add crednetials -> add below details à click on create button



Next step:

Tell jenkins where is our sonar cloud -> sonar server

Manage Jenkins → System → Scroll down and search for SonarQube servers → Enter the below details:



Create a pipeline job and add the below code:

==========

```
pipeline{


agent any


tools{
```

```
    maven 'mymaven'

}



stages{



    stage('Checkout Code'){

        steps{

            git
'https://github.com/demo1orgtoday/java-reachability-playground.git'

        }

    }

    stage('Build Code'){

        steps{

            sh 'mvn compile'

        }

    }

    stage('Sonar Scan')

    {

        environment{

            scannerHome=tool 'sonartool'  // sonar-client

        }

        steps{

            //connect to sonar server

            withSonarQubeEnv('mysonar'){
```

```
        sh '''${scannerHome}/bin/sonar-scanner \

-Dsonar.projectKey=sonar-demo-org-may-sonal_sonar-demo-project-may \

                -Dsonar.java.binaries=target/classes \

                -Dsonar.organization=sonar-demo-org-may-sonal

            '''

             }

        }

    }

}

}
```

==========

Sonarcloud integration with github action

==========

Fork this repository :
https://github.com/sonal0409ORG/SonarQubeCoverageJava.git

Go to Settings of repo → go to secrets and environments→ go to Actions → go to secret variables

Create a variable as SONAR_TOKEN

Add the sonar cloud token and save the variable.

And add the below github actions workflow

==========

```
name: Run SonarQube with Maven
on: push
jobs:
  sonarscan:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with:
        java-version: '17'
        distribution: 'adopt'
        cache: maven
    - name: Build along with Maven cloud
      run:  mvn -B verify sonar:sonar
-Dsonar.projectKey=sonar-demo-org-may-sonal_sonar-demo-project-
```

may -Dsonar.organization=sonar-demo-org-may-sonal
-Dsonar.host.url=https://sonarcloud.io -Dsonar.token=${{
secrets.SONAR_TOKEN }}

env:

GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}

SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}

Commit the file and click on Actions to check the execution.

==========

Agenda: 11-May-2025

==========

- SCA testing using Snyk
- DAST testing using ZAP
- Security and user authorization in Jenkins

DEMO : Snyk Integration with Jenkins
==========

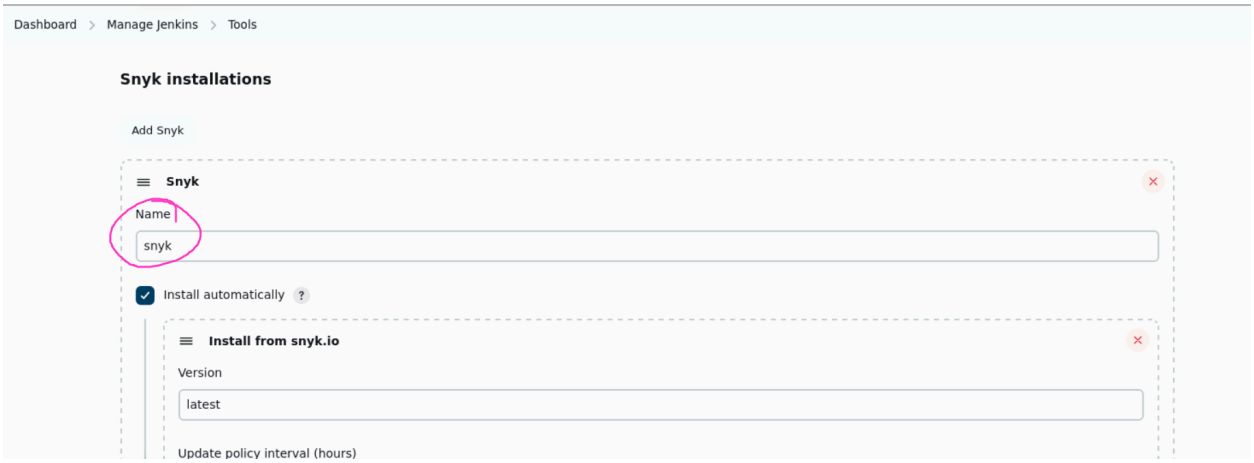
Download Snyk plugin in Jenkins

Download the synk security plugin

Manage jenkins → plugins→ available plugins→ search for snyk security → Click on install button

Configure Snyk tool

Manage Jenkins → Tools → scroll down to snyk installations -> Add snyk -> give name as snyk -> save the page.



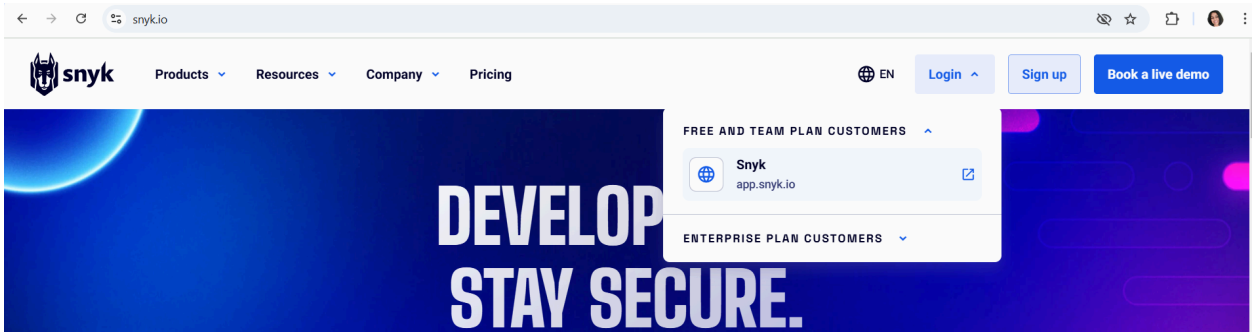
Lets us now connect to snyk application:

Go to : https://app.snyk.io/login

Demo 2: Setup snyk tool and generate token to authenticate with Jenkins

Go to the URL : https://snyk.io/ and open it on your laptop browser.

Click on login and then click on Snyk

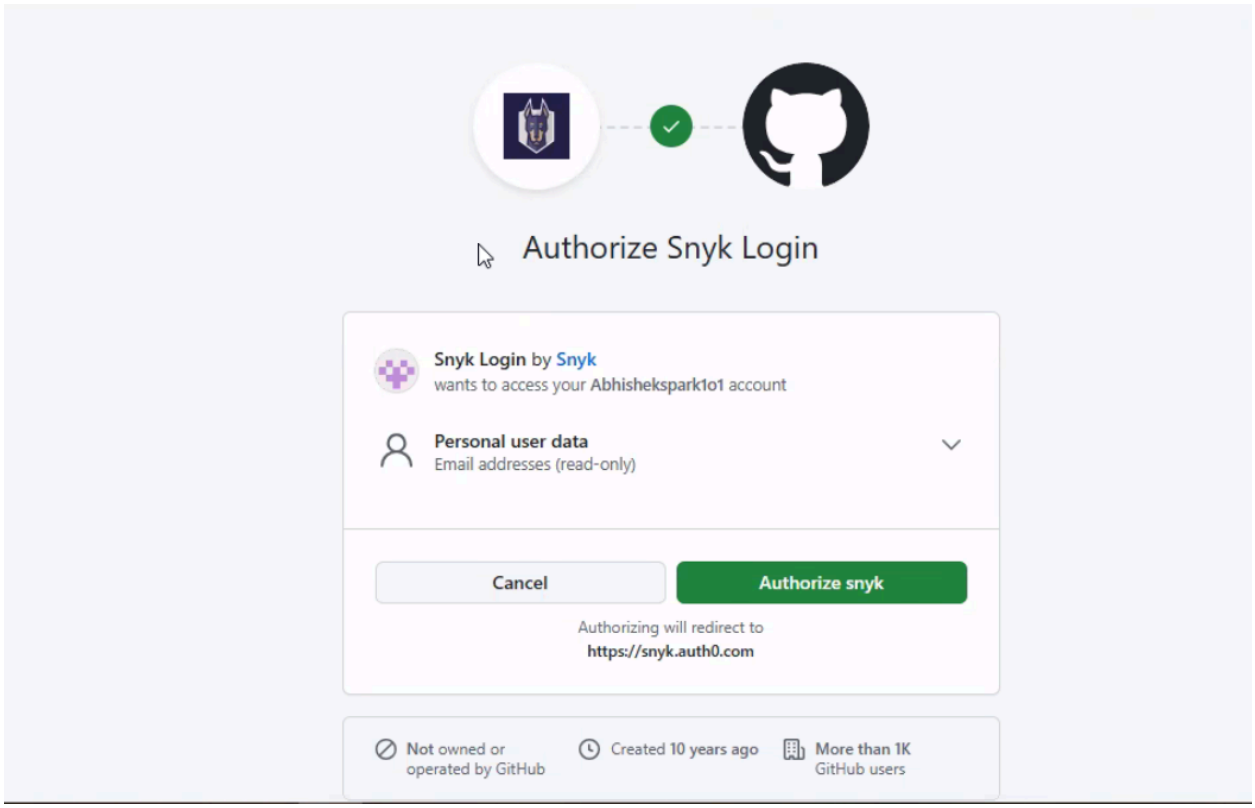


OR open below URL on your browser

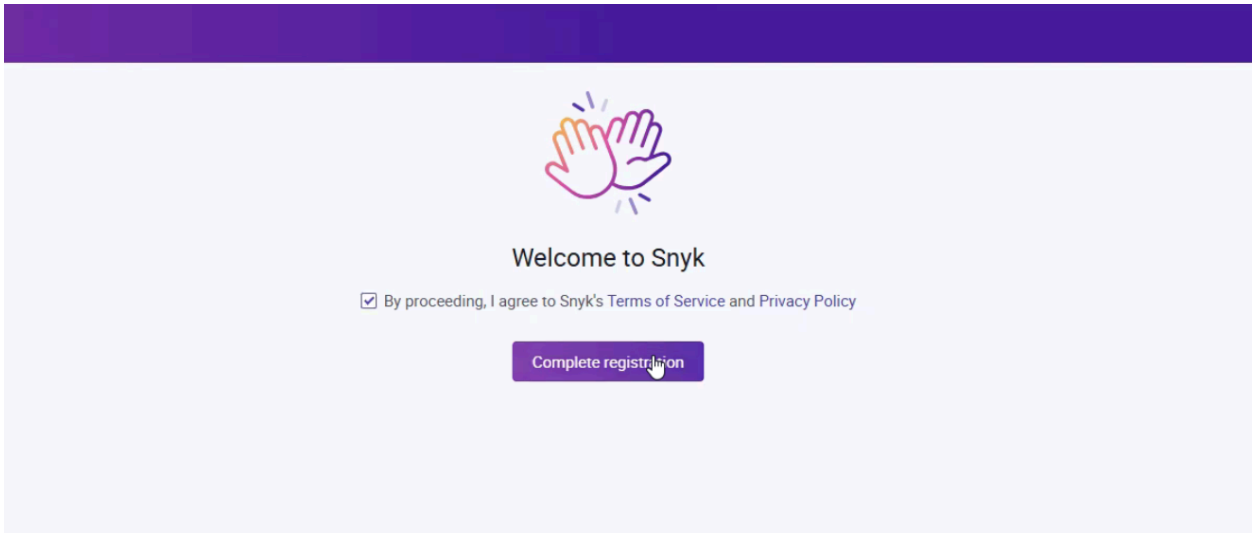
https://app.snyk.io/login

Click on GITHUB

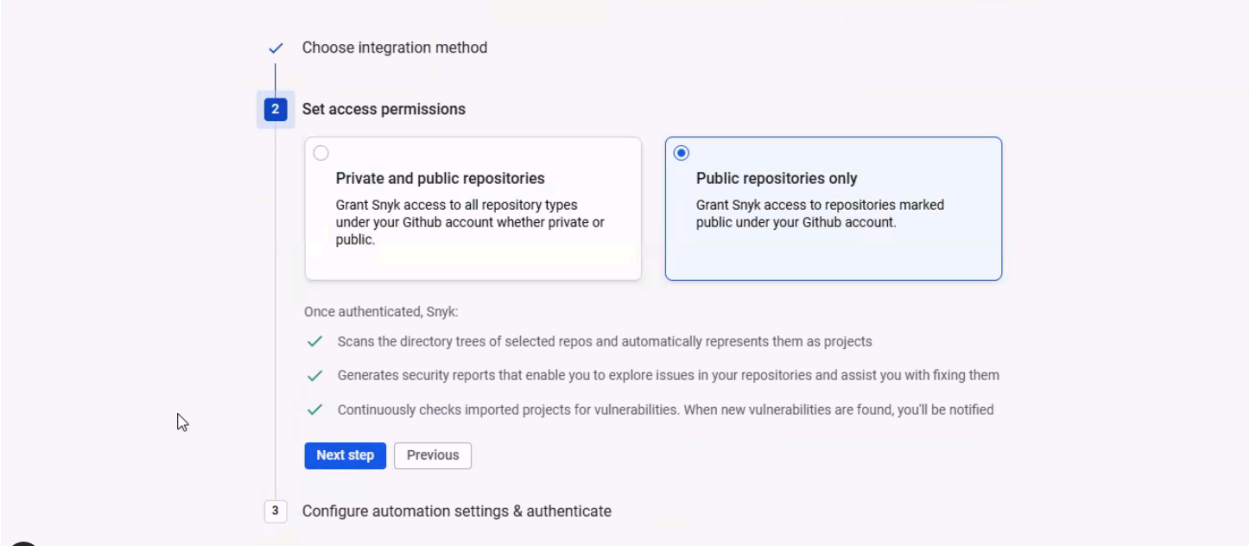
Click on Authorize snyk



Click on checkbox and Complete registration

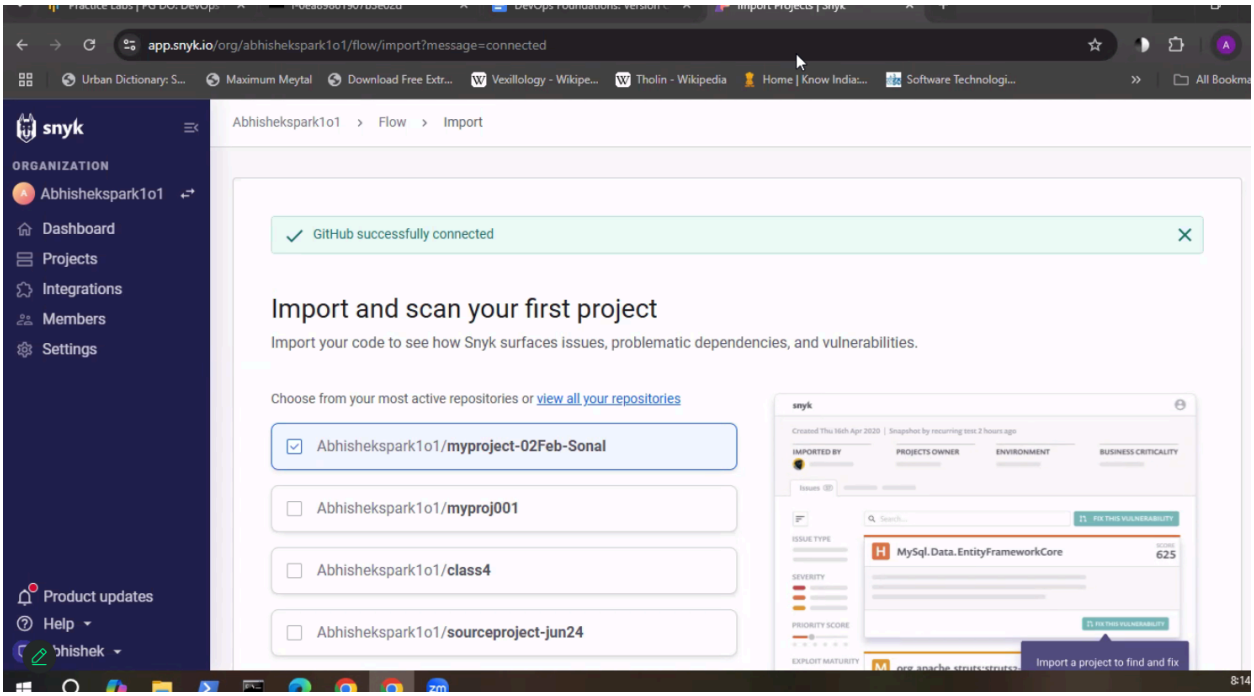


Here select public repositories and click on next step

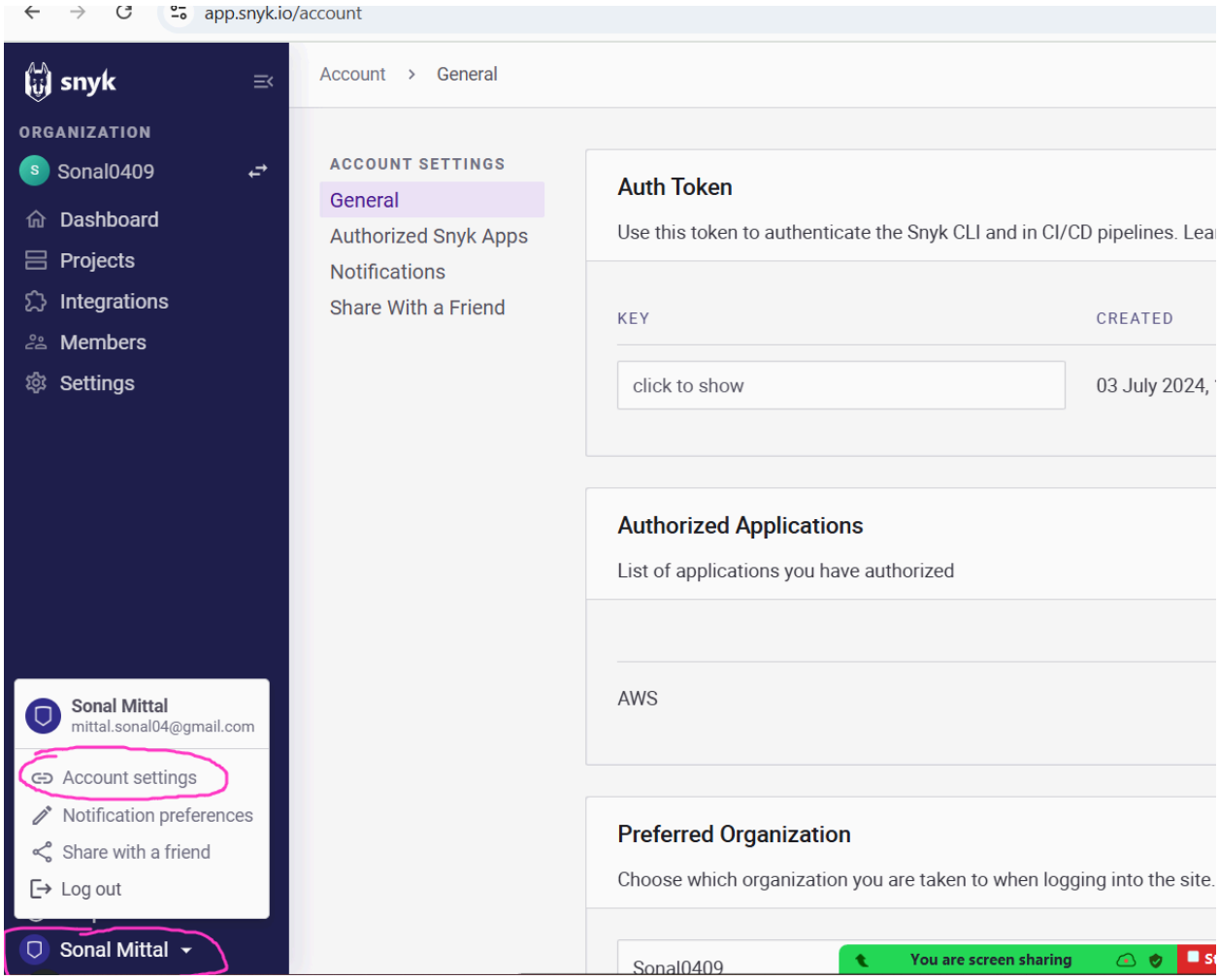


Click on Authenticate github and complete the 3rd setup

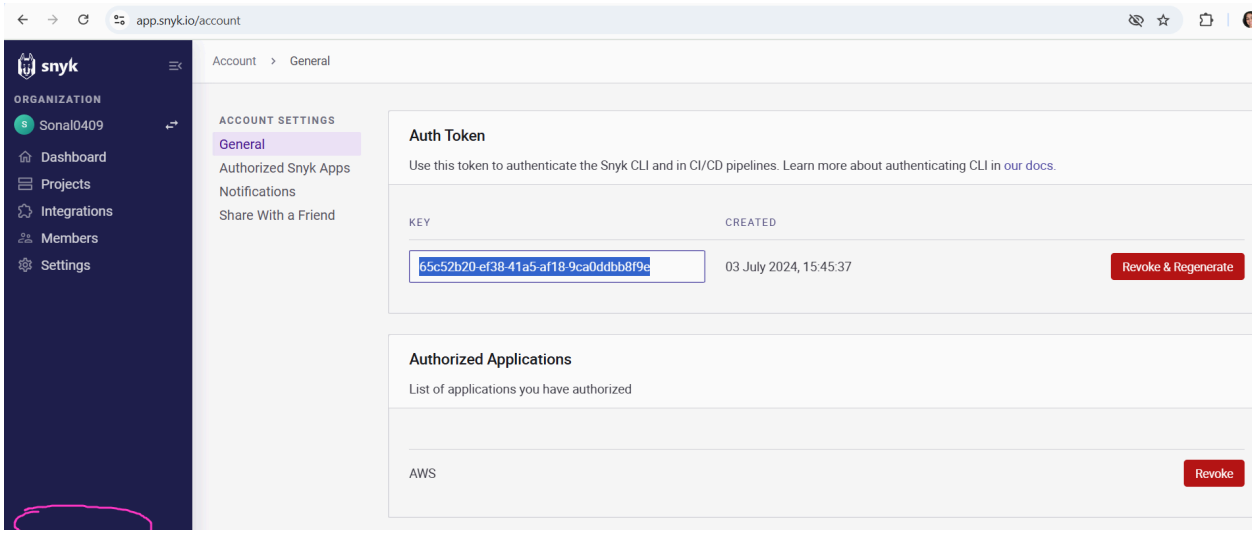
You will on the dashboard



Click on your name at the bottom → click on account settings

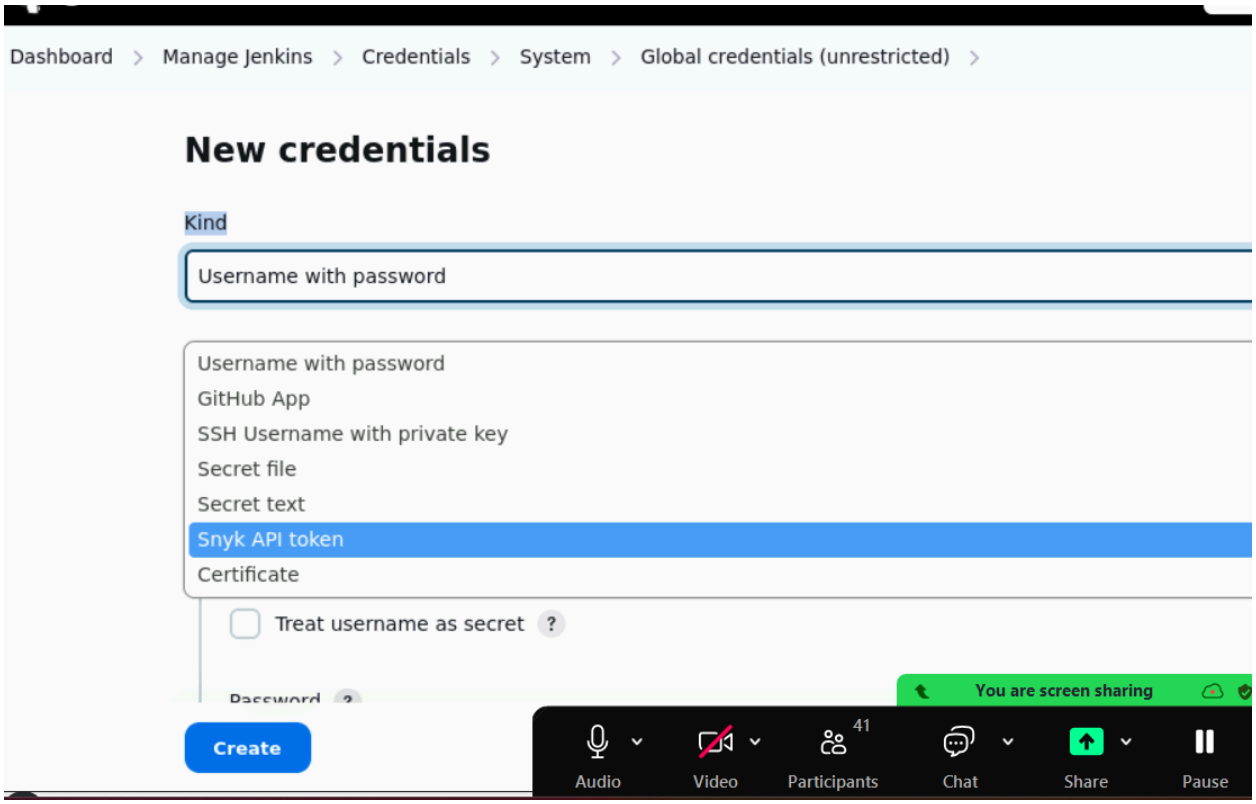


In the general section → auth token → click on Key input box → copy the key



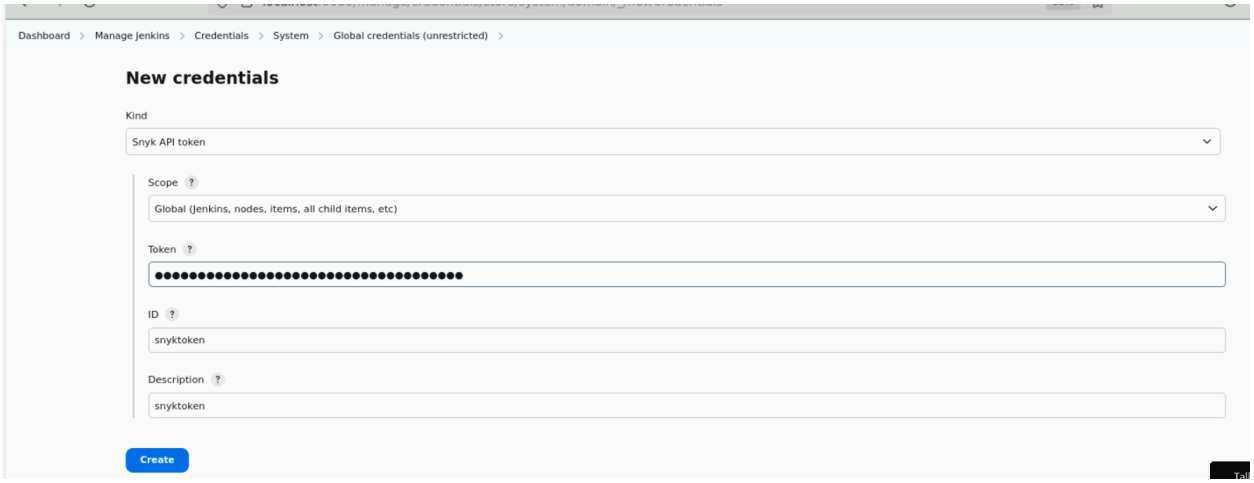
Go to Lab:

Manage Jenkins --> scroll to credentials --> click on it --> click on global--> click on add credentials button → In kind dropdown you will see snyk api token



Ge the Auth token from your Snyk tool which we have generated earlier

Paste the token
Give ID as snyktoken
Give description snyktoken
Click on create



If maven installation has not been done→ pls complete that also

Manage Jenkins --> tools --> scroll down
maven Installations --> add maven
give name as mymaven
and save the page

==========

Install Snyk using the lab terminal

```
sudo apt update
sudo apt install -y nodejs npm
sudo npm install -g snyk
```

Run the below pipeline

```
pipeline{
    agent any

    options{
        timestamps()
    }
    tools{
        maven 'mymaven'
    }

    stages{
        stage("checkout"){
            steps{
                git
'https://github.com/niiloy/SonarQubeCoverageJava.git'
            }
```

```
        }
        stage("snyk server"){
            steps{
                sh "snyk --version"
                //snykSecurity snykInstallation: 'snyk',
snykTokenId: 'snyktoken'
                withCredentials([string(credentialsId:
'snyk_token', variable: 'SNYK_TOKEN')]) {
                    sh '''
                        snyk auth $SNYK_TOKEN
                        snyk monitor
--project-name=niiloy/SonarQubeCoverageJava
                    '''
                }

            }
        }
    }

}
```

==========

GITHUB ACTIONS WITH SYNK

```
name: Run SCA scan with Snyk
on: push
jobs:
  Snykscan:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Set up JDK 17
      uses: actions/setup-java@v3
      with:
        java-version: '17'
        distribution: 'temurin'
        cache: 'maven'
    - name: Build the code
      run: |
       echo "JAVA_HOME=$JAVA_HOME"
       mvn clean install -DskipTests
    - name: Run Snyk scan
      uses: snyk/actions/maven@master
      continue-on-error: true
      env:
        SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
```

with:
 args: test

==========

DAST -> ZAP tool

==========

ZAP is an open source security testing tools which is used to find issues on the web application after it has been deployed.

Using ZAP we will perform passive or baseline scan in which we will observe the communication between the web browser and the application without actively iterating with the application

Using ZAP we will perform active scans which are more aggressive to the target application
In this scan the ZAP tool will send different types of requests to different endpoints of the web application

Docker tools and its integration with jenkins

==========

Docker is a deployment tool, that will help you to deploy your build or application on a container

What is a container:

==========

A container is process which holds you application and its required s.w or frameworks needed for the app to run

App.war + tomcat+java+os lib

It is an isolated process where your application is running, a container has an IP address, has volume, has network, has file storage

A container is running → application is running

A container stop→ application stops

How to create a container

==========

A container is always launched from an Image
Image is just a binary file or an executable

When we run an Image a container is launched and container will be running with the application

Image is a file which has a list of libraries that are needed for our application to run on a container.

Docker → IMAGE → Container [App is running]

Types of images:
==========

Custom Image : Images created by users for their application code

Base Image
==========

Docker as tool comes with list of ready made Images for open source:

- **Databases**
- **OS**
- **Tools**
- **Programming language**

SO today we will use that base image of docker to install zap tool

In today's class we will use docker base image for ZAP

We will run the image → we will get a container in which ZAP tool is running

We will then provide a webpage URL as input to the ZAP container so that zap tool can scan the webpages of the application and generate DAST report.

Create a pipeline job and add below code:
==========

Provide permission to jenkins user to run docker commands:

GO to lab terminal and run below commands:

 # sudo su -

 # chmod 777 /var/run/docker.sock

==========
Create a new pipeline job and add below code

==========

```
def scan_type
def target
pipeline{

agent any

parameters {
       choice  choices: ['Baseline', 'APIS', 'Full'],
              description: 'Type of scan that is going to perform inside the container',
              name: 'SCAN_TYPE'

       string defaultValue: 'https://medium.com/',  // URL to scan
              description: 'Target URL to scan',
              name: 'TARGET'

}


stages{
    stage('Setting up of the OWASP ZAP container'){

     steps{
        echo "pulling Image of ZAP ---> Start"
        sh 'docker pull ghcr.io/zaproxy/zaproxy:stable'
        echo "pulling of Image completed ---> End"
        echo "Running Image ---> Starting Container"
        sh 'docker run -dt --name owasp-$BUILD_NUMBER ghcr.io/zaproxy/zaproxy:stable'

     }
     }

    stage('Creating a working Directory in the container'){

      steps{
              echo "Connect to container and execute command to create a directory"
              sh 'docker exec owasp-$BUILD_NUMBER mkdir /zap/wrk'
      }
    }
```

```
    stage('Scan target on owasp container') {

steps {
            script {
                                                scan_type = "${params.SCAN_TYPE}"
               echo "----> scan_type: $scan_type"
               target = "${params.TARGET}"


if (scan_type == 'Baseline') {
                  sh """
                     docker exec owasp-$BUILD_NUMBER \
                     zap-baseline.py \
                     -t $target \
                     -r report.html \
                     -I
                   """
                }
else if (scan_type == 'APIS') {
                  sh """
                     docker exec owasp-$BUILD_NUMBER \
                     zap-api-scan.py \
                     -t $target \
                     -r report.html \
                     -I
                   """
                }
                else if (scan_type == 'Full') {
                  sh """
                     docker exec owasp-$BUILD_NUMBER \
                     zap-full-scan.py \
                     -t $target \
                     -r report.html \
                     -I
                   """
                }
                else {
                  echo 'Something went wrong...'
                }
    }
        }
     }
```

```
        stage('Copy Report to Workspace') {
            steps {
                script {
                    sh '''
                        docker cp owasp-$BUILD_NUMBER:/zap/wrk/report.html
${WORKSPACE}/report.html
                    '''
                }
            }
        }
    }


post {
        always {
            echo 'Removing container'
            sh '''
                    docker stop owasp-$BUILD_NUMBER
                    docker rm owasp-$BUILD_NUMBER
                '''

 }


}


}
```

Save the job and run

==========

These are complete steps for Project 1 using githubActions and tomcat

https://github.com/demo1orgtoday/MavenBuild-SL.git

1. Create a code repository on GitHub

2. Create a GitHub Actions pipeline to perform continuous integration

```
name: CICD using Maven and tomcat
on:
 push:  # run when there is commit to repo
 workflow_dispatch:  # run manually
jobs:
 CICDjob:
   runs-on: ubuntu-latest
   steps:
    - name: Clone the repo on ubuntu server
      uses: actions/checkout@v4
    - name: Install Java and maven on ubuntu server
      uses: actions/setup-java@v4
      with:
       distribution: 'temurin'
       java-version: '11'
       cache: 'maven'
    - name: Build the code
      run: mvn package
```

3. Configure Tomcat Apache for automated code deployment

4. Integrate the GitHub Actions pipeline to invoke the Jenkins pipeline

go to lab install tomcat9 and tomcat9-admin

setup user.xml and server.xml, restart tomcat

```
 - name: connect to LAb and deploy code on tomcat9
      uses: cross-the-world/ssh-scp-ssh-pipelines@latest
      with:
       host: '13.233.135.136'
       user: 'labuser'
       pass: 'Nuvelabs123$'
       port: 22
       connect_timeout: 10s
       first_ssh: |
         sudo chmod 777 /var/lib/tomcat9/webapps
       scp: |
         './target/*war' => /var/lib/tomcat9/webapps
       last_ssh: |
         sudo systemctl restart tomcat9
```

5. Invoke pipeline to validate automated deployment

Steps to run a pipeline job form GitHub actions:

Go to jenkisn JOB--> go to triggers --> Trigger builds remotely (e.g., from scripts) --> give any token name (token1)

Construct the remote URL using this syntax:
JENKINS_URL/job/JOB-NAME/build?token=TOKEN_NAME

http://localhost:8080/job/Githubaction_trigger/build?token=token1

Github actions will go to terminal and run the job --construct the url

curl -l -u admin:Root123$ http://localhost:8080/job/Githubaction_trigger/build?token=token1

Now go to GitHub action main.yml file add the following code

```
- name: Connect to Lab machine and trigger a Jenkins job
      uses: cross-the-world/ssh-scp-ssh-pipelines@latest
      with:
        host: '43.205.118.149'
        user: 'labuser'
        pass: 'Nuvelabs123$'
        port: 22
        connect_timeout: 10s
        first_ssh: |
          curl -l -u admin:Root123$ http://localhost:8080/job/Githubaction_trigger/build?token=token1
```

The complete workflow

==========

```
name: CICD using Maven and tomcat
on:
 push:  # run when there is commit to repo
 workflow_dispatch:  # run manually
jobs:
 CICDjob:
   runs-on: ubuntu-latest
   steps:
    - name: Clone the repo on ubuntu server
      uses: actions/checkout@v4
    - name: Install Java and maven on ubuntu server
      uses: actions/setup-java@v4
      with:
       distribution: 'temurin'
       java-version: '11'
       cache: 'maven'
    - name: Build the code
      run: mvn package
    - name: connect to LAb and deploy code on tomcat9
      uses: cross-the-world/ssh-scp-ssh-pipelines@latest
      with:
       host: '13.233.135.136'
       user: 'labuser'
       pass: 'Nuvelabs123$'
       port: 22
       connect_timeout: 10s
       first_ssh: |
         sudo chmod 777 /var/lib/tomcat9/webapps
       scp: |
```

```
        './target/*war' => /var/lib/tomcat9/webapps
      last_ssh: |
        sudo systemctl restart tomcat9
    - name: Connect to Lab machine and trigger a Jenkins job
      uses: cross-the-world/ssh-scp-ssh-pipelines@latest
      with:
        host: '43.205.118.149'
        user: 'labuser'
        pass: 'Nuvelabs123$'
        port: 22
        connect_timeout: 10s
        first_ssh: |
          curl -l -u admin:Root123$
http://localhost:8080/job/Githubaction_trigger/build?token=to
ken1
```

==========

Project 2:

Setup slack workspace and slack channel
Integrate Jenkins application with slack
We have already done jenkins sending notification to slack

As part of project -> slack should trigger jenkins job

Step1:
==========

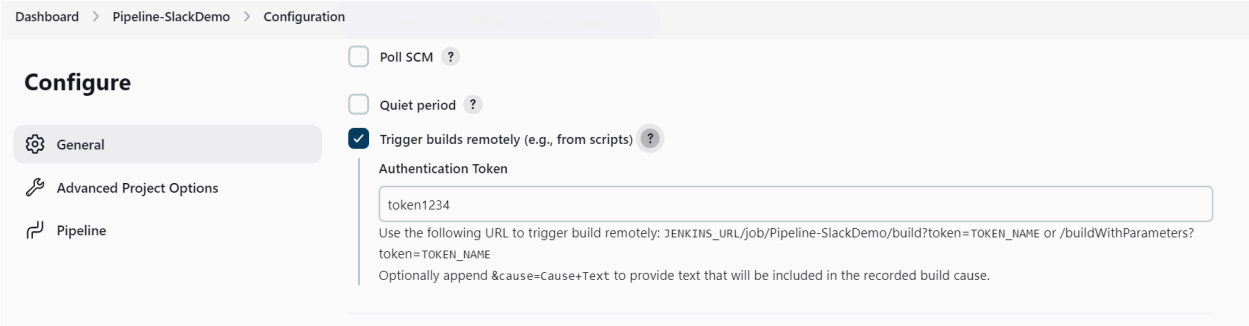
Setup a trigger in Jenkins job so that it can be executed from slack

Select a Jenkins job and create a trigger to run a Jenkins job from a remote server.

Jenkins Job→ configure→ Build triggers -> Select the trigger ->

Trigger builds remotely (e.g., from scripts)

Enter the Authentication token -> give any name to the token → token123

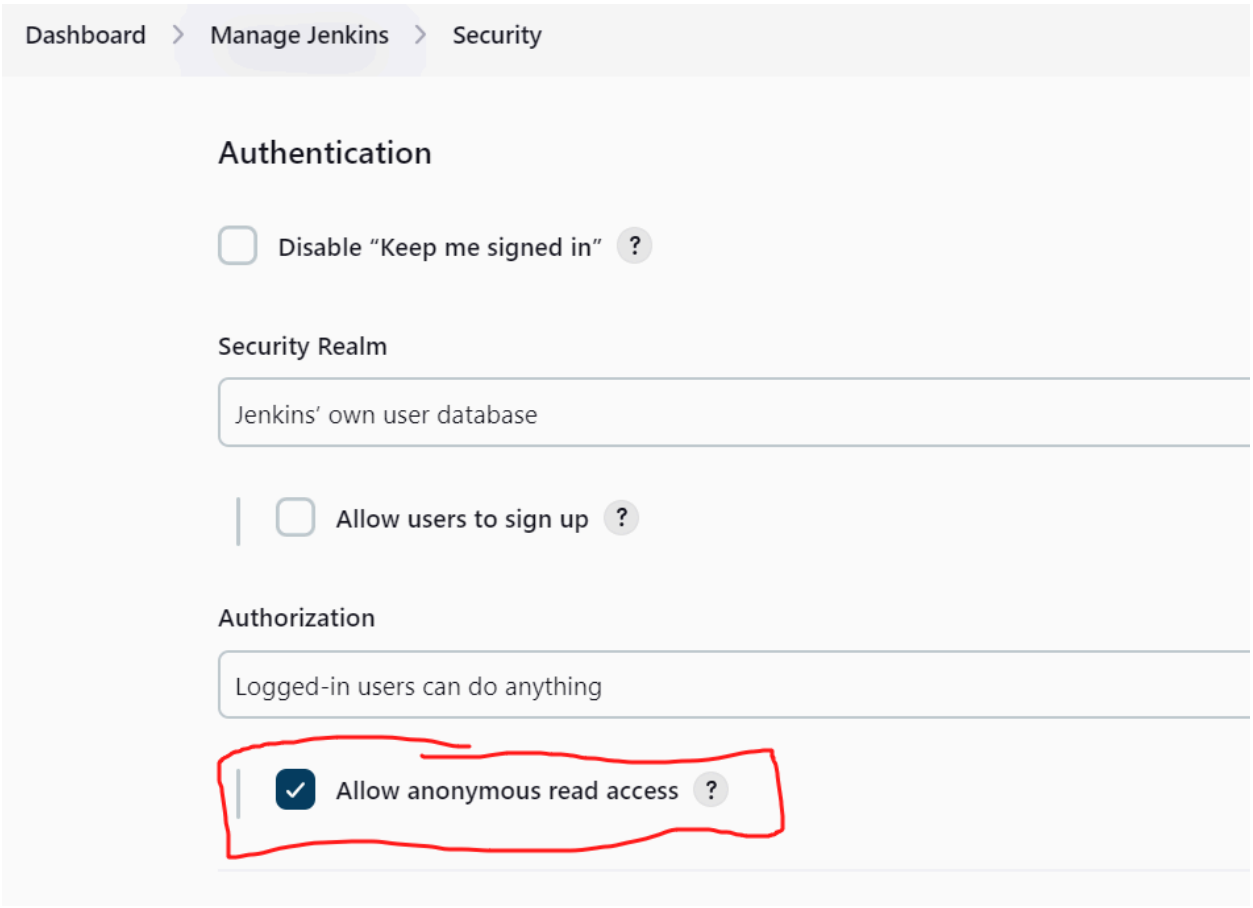


Construct the URL based on the token given

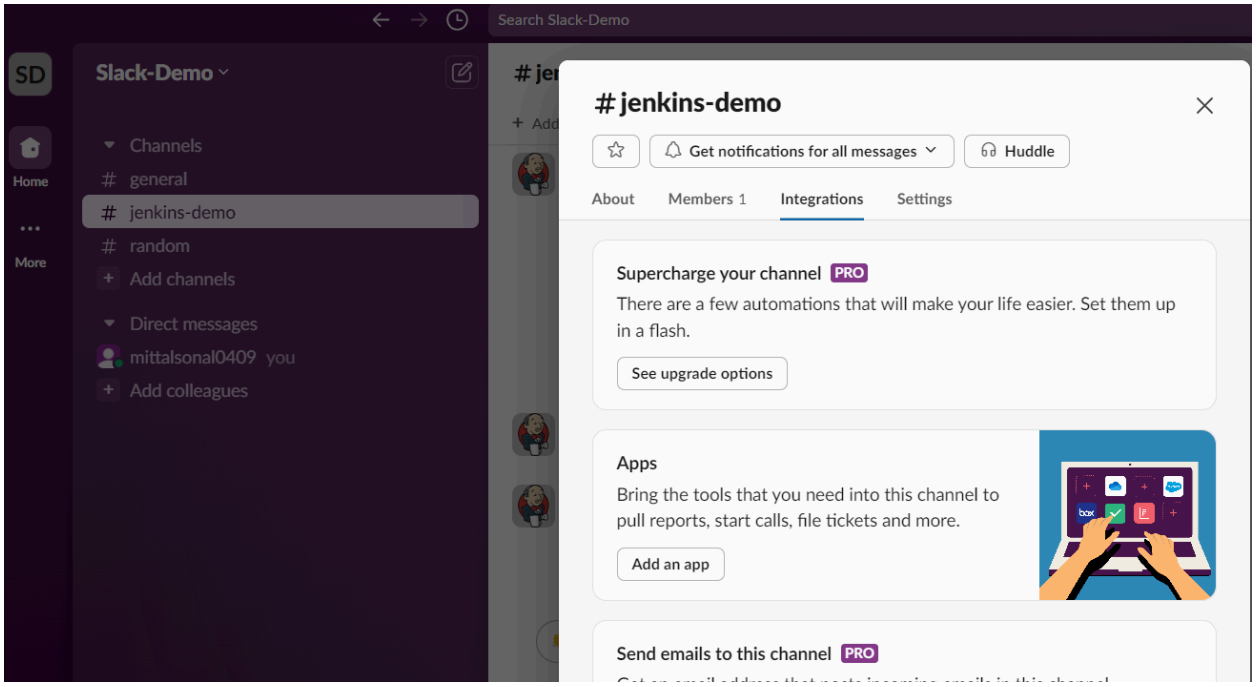
JenkinsURL/job/Pipeline-SlackDemo/build?token=TOKEN_NAME

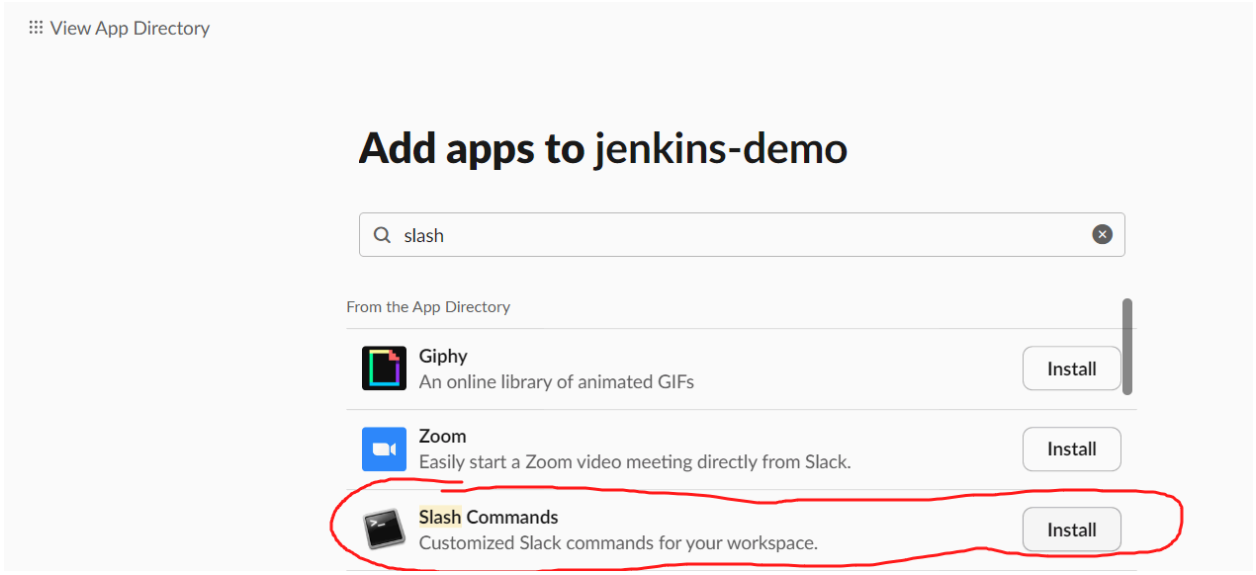
http://localhost:8080/job/SlackPipeline/build?token=token123

Manage Jenkins → Security-> Select Allow anonymous read access



Go to Slack Channel and add an APP

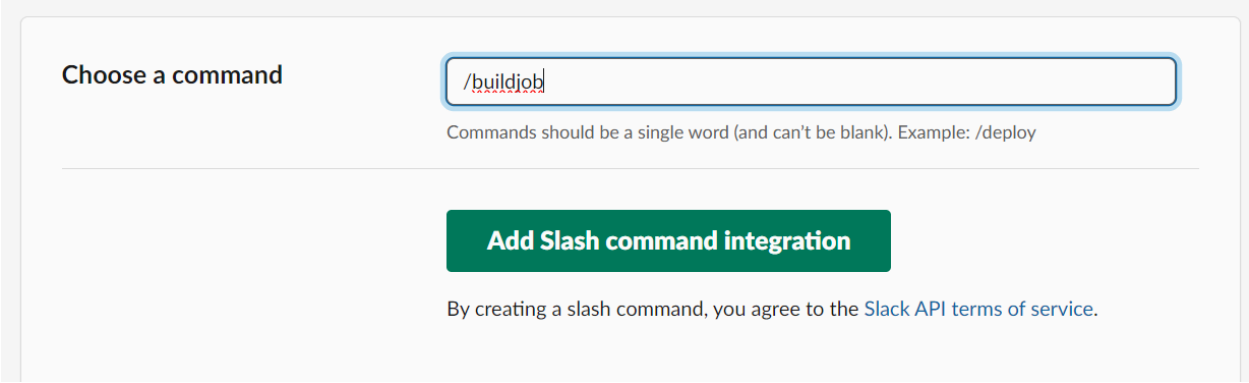




Click on Add to Slack

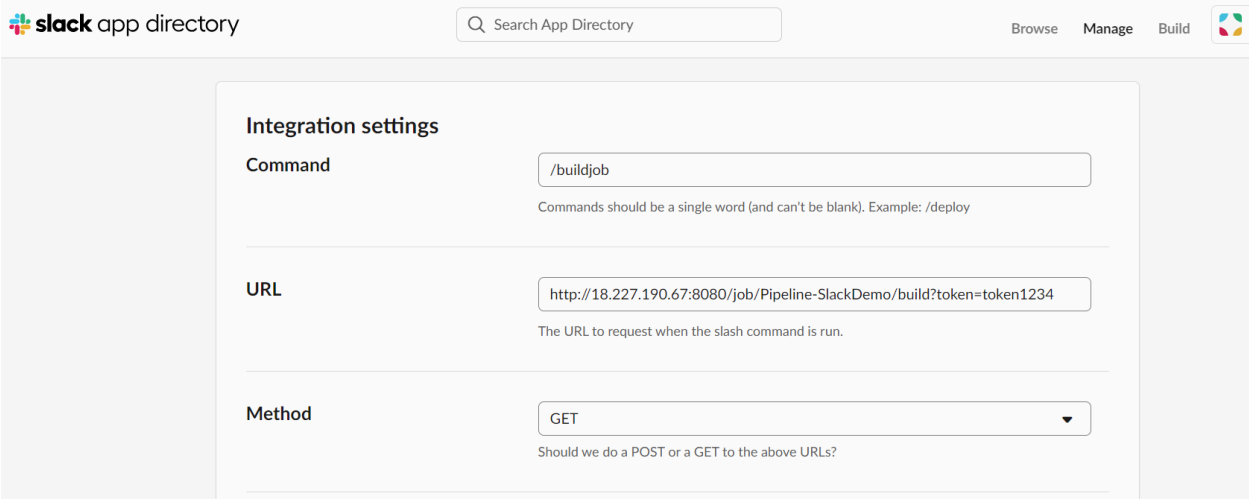
Now choose a command

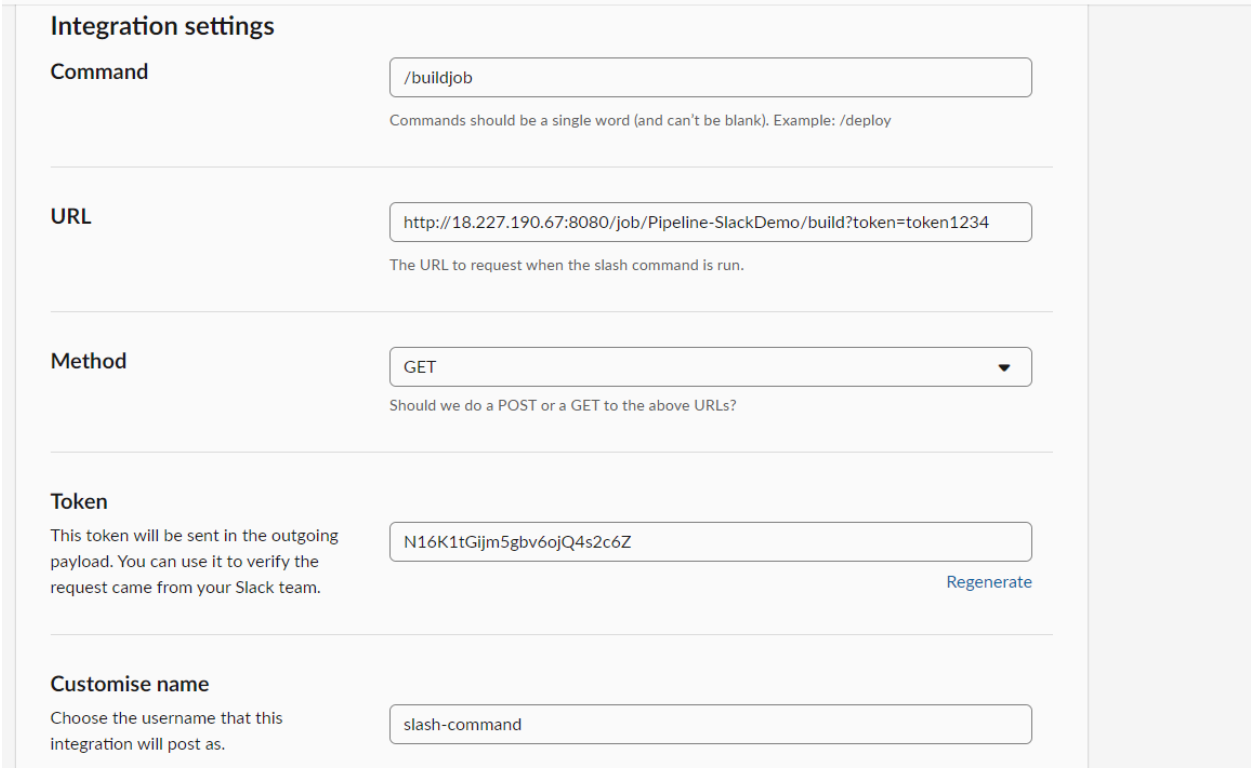
Note: command starts with /
Command will not have a space

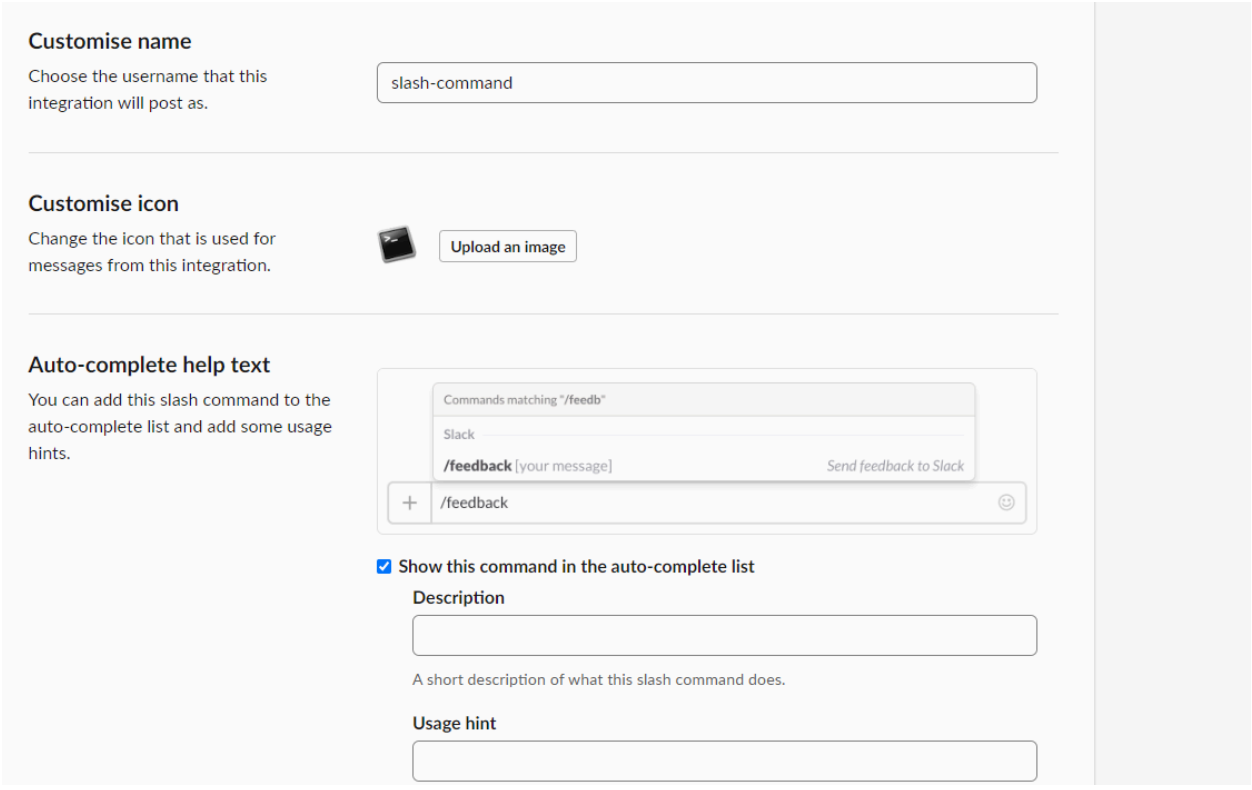


On the next page we have to add Jenkins URL details

Scroll down and enter Integration settings

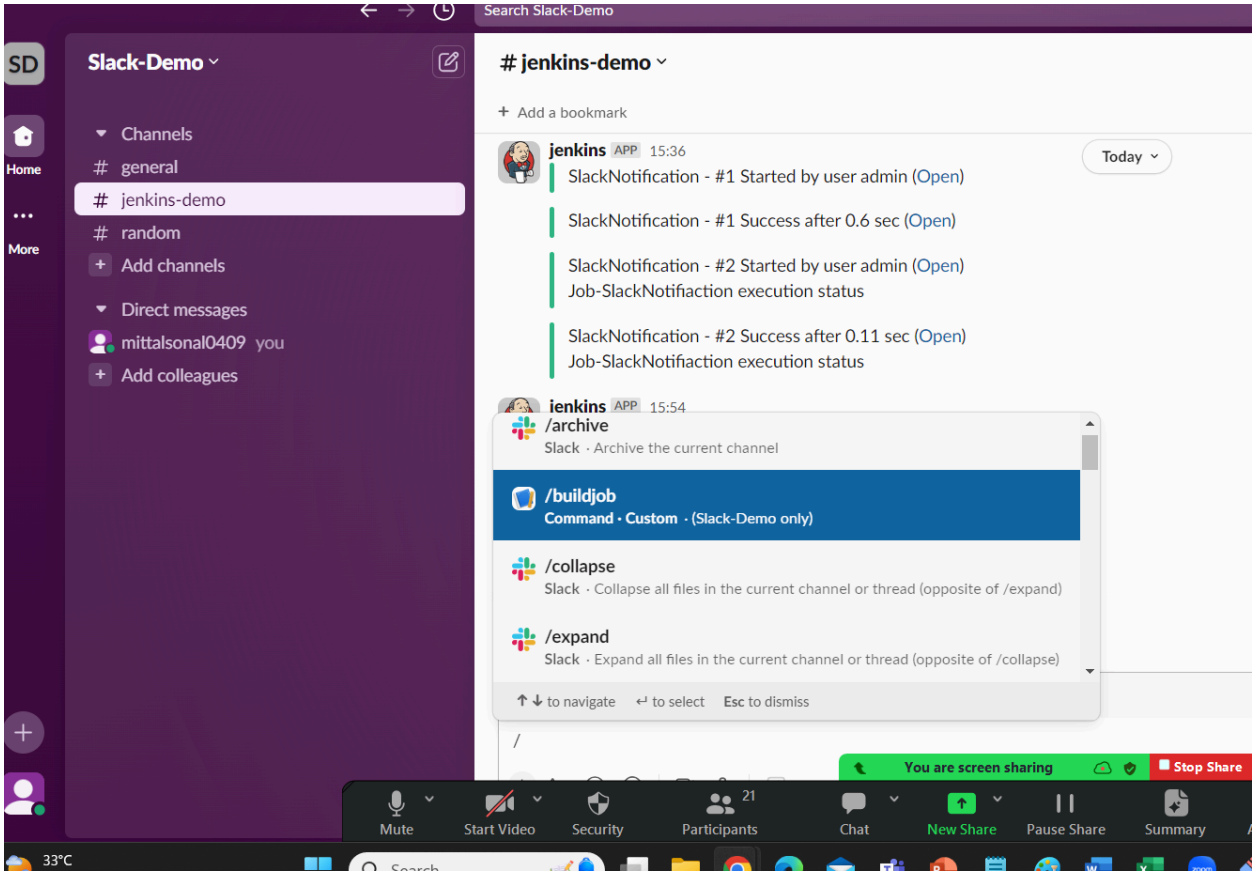






Save the integrations

GO to slack dashboard, and use slack to trigger jenkins job



The project is completed.