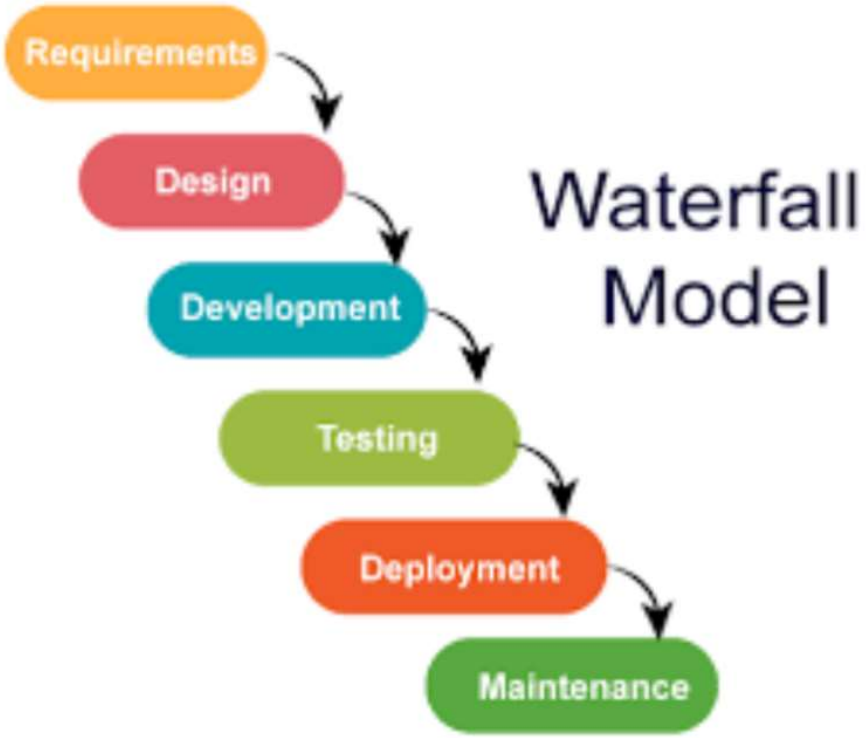
MENU ☰

SEPTEMBER 6, 2022

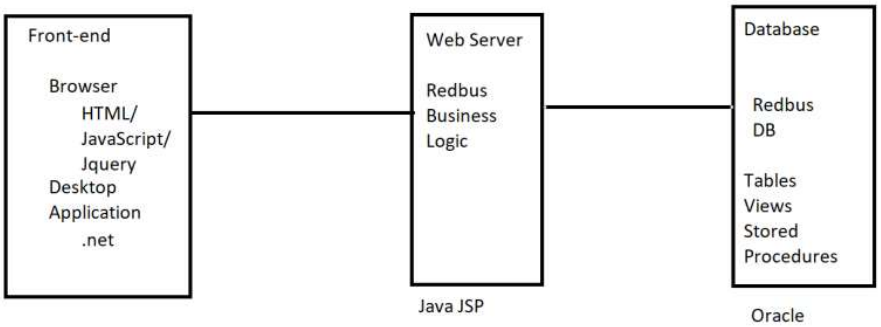
DevOps Classroomnotes 06/Sep/2022

Software Development Life Cycle (SDLC)

Waterfall Model:



* The Release cycle was in the range of 6 months – 2 years.
* Our goal: Is to build bus ticket booking system
* To build bus ticket booking system:
* The design of the application is as follows



* Assume the development has to be started on Aug 15 2004 and to be finished by Aug 15 2005
* During this time, testing team is preparing test cases based on Requirements documents
* Development is handled by 4 teams
* SQL Developers
* Java Developers
* UI Developers

* .net Developers
* Each of this team works in isolation.
* On Aug 1 2005, they decided to combine their work, this is referred as Big Bang Integration.
* This situation often leads to errors, now to meet deadlines we make things work somehow with incorrect fixes to your application.
* As a best practice, We do integration (Combining the work by all dev teams) frequently from day 1.
* All the integration failures will be known to the teams in the early stages of development.
* This way of integration is called Continuous Integration.
* Failures are Common, But be aware of failures rather than last minute surprises => Continuous Integration.
* For performing Continuous There are two important things
* We should integrate for almost every change done by any developer in any team pf the project => Day build
* We should also integrate for a stable changes done by all of your team memebers daily. This build will be given to your testing team next day for testing. => Night Build

- To Perform CI, CI Engines Like
    - Cruise Control Refer Here
    - Jenkins/Hudson Refer Here
- Next Step:
    - Understanding Version Control Systems

Stuff needed for Labs on CI/CD

- Cloud Account
    - Azure
    - AWS
- Windows 10/11
- Foundations:
    - Watch 1-7 videos Refer Here

Leave a Reply

Enter your comment here...

MENU ☰

SEPTEMBER 7, 2022

DevOps Classroomnotes 07/Sep/2022

Terms

Repository

- Repository is a Data storage solution where the history is maintained.
- Generally for this versioning is used and along with every version some meta data like reason for change is also maintained.
- Generally Repositories also have the option to view differences between versions.
- W.R.T CI/CD Pipelines we have two major repositories
    - Source Code Repository
    - Package Repository

Version Control System (VCS)

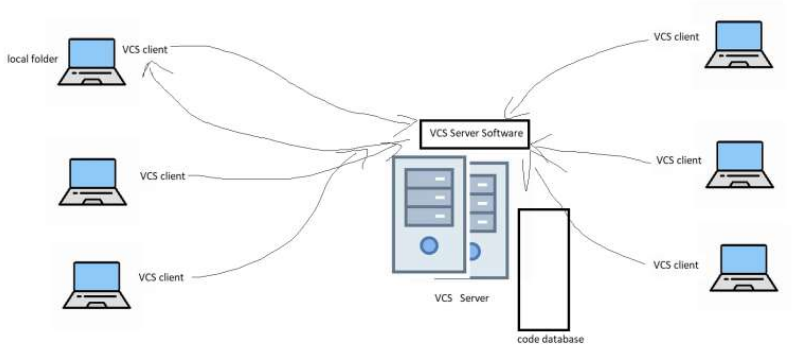
- VCS is used to store generally code (any files)
- We would require
    - Repository features
    - Multi-user
        - Synchronization
    - Multiple releases parallely

Evolution of VCS

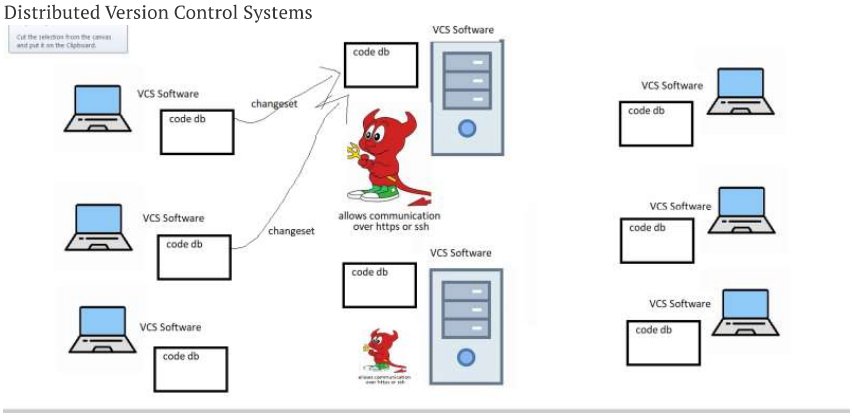
- Single User VCS => VSS
- Multi User VCS => Subversion (SVN)
- Distributed VCS => Bitkeeper, Git

Architectures of Version Control System

- Centralized Version Control Systems



- Distributed Version Control Systems



Leave a Reply

Enter your comment here...

This site uses Akismet to reduce spam. Learn how your comment data is processed.

About continuous learner

devops & cloud enthusiastic learner

VIEW ALL POSTS

MENU ☰

SEPTEMBER 8, 2022

DevOps Classroomnotes 08/Sep/2022

Git (A Distribute Version Control System)

- Git Supports working offline and then sync with Remotes (Servers with git & daemons) when required.

Lab Setup

- Windows:
  - Install Windows Terminal
  - Refer Here to download git for windows
  - Chocoloatey Refer Here: This is package manager
  - Visual Studio Code `choco install vscode -y`
  - AWS CLI `choco install awscli -y`
  - Azure CLI `choco install azure-cli -y`
- Mac:
  - Install Homebrew Refer Here
  - Git: `brew install git -y`
  - Visual Studio Code: `brew install --cask visual-studio-code`
  - AWS CLI `brew install awscli`
  - Azure CLI `brew install azure-cli`

Leave a Reply

MENU ☰

SEPTEMBER 9, 2022

DevOps Classroomnotes 09/Sep/2022

Git

- Watch the classroom video for some references in this document

Scenario -1: For CI/CD Pipelines

- Developer's View:
  - Created/used the repository
  - Added changes
  - pushed changes
- DevOps View:
  - Consider the change pushed at start creating
    - packages
    - sending packages to package repository
    - Create various test environments
    - Run automated tests
    - At any point if any thing fails notify the whole team.
- Refer Here

Scenario 2: Creating Reusable assets as DevOps Engineer

- Refer Here for the dummy repo created

First Steps into Git

- Installing Git: Refer previous classroom notes
- Verifying if the git is installed or not `git --version`

```
PS C:\Users\Dell> git --version
git version 2.35.1.windows.2
PS C:\Users\Dell>
```

- The first step in Git is to create a local repository.
  - Create empty folder
  - cd into it
  - execute init command `git init`

```
PS C:\Users\Dell> cd C:\temp\Understanding
PS C:\temp\Understanding> mkdir hello-git


    Directory: C:\temp\Understanding


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          9/9/2022   8:29 AM                hello-git


PS C:\temp\Understanding> cd .\hello-git\
PS C:\temp\Understanding\hello-git> git init
Initialized empty Git repository in C:/temp/Understanding/hello-git/.git/
PS C:\temp\Understanding\hello-git>
```

- Best approach:
    - Start using cheatsheets
    - Refer Here for the sample cheatsheet.

- Lets add a file to the repository

```
PS C:\temp\Understanding\hello-git> New-Item first.txt


    Directory: C:\temp\Understanding\hello-git


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----          9/9/2022   8:37 AM              0 first.txt


PS C:\temp\Understanding\hello-git> git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        first.txt

nothing added to commit but untracked files present (use "git add" to track)
PS C:\temp\Understanding\hello-git>
```

```
PS C:\temp\Understanding\hello-git> git add .\first.txt
PS C:\temp\Understanding\hello-git> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   first.txt

PS C:\temp\Understanding\hello-git> git log
fatal: your current branch 'master' does not have any commits yet
PS C:\temp\Understanding\hello-git> git commit -m "first commit"
[master (root-commit) ebddd5c] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 first.txt
PS C:\temp\Understanding\hello-git> git log
commit ebddd5c04fce4d999d31644f6db1c51bb2e4f617 (HEAD -> master)
Author: shaikkhajaibrahim <qtkhajacloud@gmail.com>
Date:   Fri Sep 9 08:38:37 2022 +0530

    first commit
PS C:\temp\Understanding\hello-git> git status
On branch master
nothing to commit, working tree clean
PS C:\temp\Understanding\hello-git>
```

Three Areas of Git

- In the local repo or local system which represents a developer's system, We have three areas
    - Working Tree: This is where we make changes
    - Staging area: This is intermediate area before sending changes to local repo
    - Local Repo: Once the change is in local repo it will have history

MENU ☰

SEPTEMBER 10, 2022

DevOps Classroomnotes 10/Sep/2022

Three Areas of Git

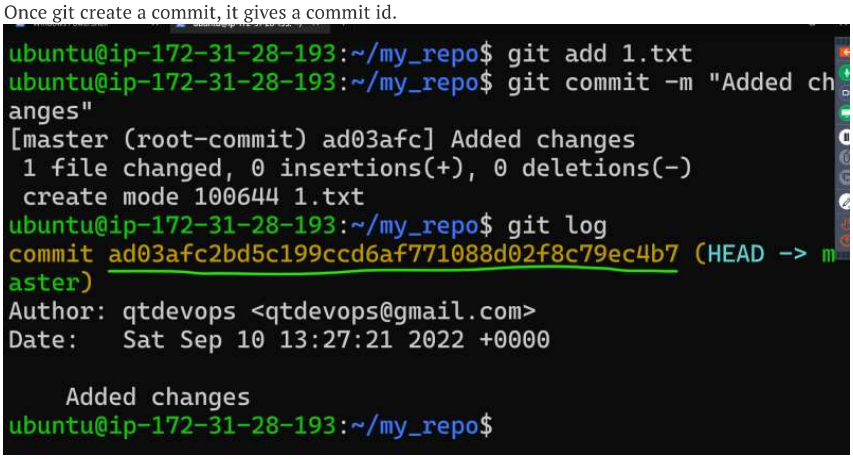
- Areas of Git

- Lets create a new folder and initialize it to be git repository
- Lets create a change
- Add the change to staging area and commit the change
- To commit a Change i.e. to maintain history, git requires
  - changes
  - time when commit is done
  - author username and email id

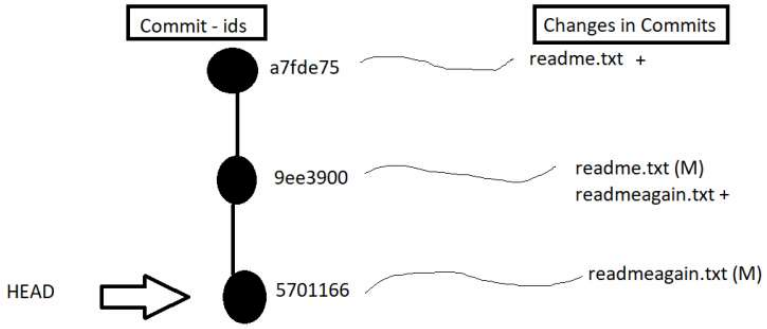
- To configure username and email id

```
git config --global user.name "qtdevops"
git config --global user.email "qtdevops@gmail.com"
```

- Once git create a commit, it gives a commit id.



- When we add new changes and commit the changes the latest commit will be child of previous commit or previous commit is parent of latest commit



Now Lets focus of on the other aspects of git

Empty Folders and Folders with files

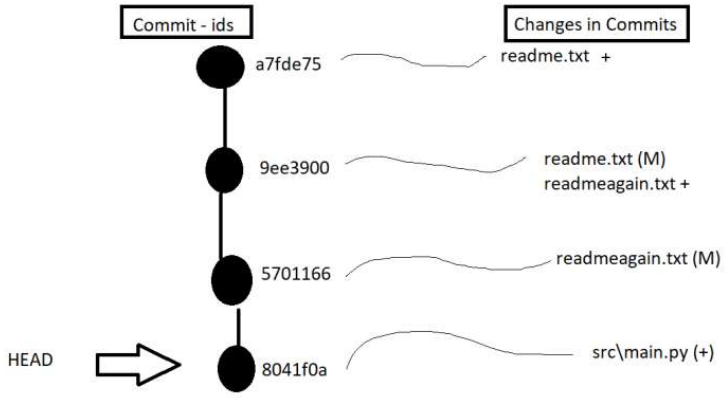
- Create a new folder and verify the status. git doesn't identify an empty folder as a change. only the folders with files are considered as changes

```
    Directory: C:\temp\Understanding\secondrepo


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         9/10/2022   7:26 PM                src
d-----         9/10/2022   7:26 PM                test
-a----         9/10/2022   7:14 PM              5 readme.txt
-a----         9/10/2022   7:14 PM             11 readmeagain.txt


PS C:\temp\Understanding\secondrepo> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        src/

nothing added to commit but untracked files present (use "git add" to track)
PS C:\temp\Understanding\secondrepo>
```

- In the above image src is a folder with a file and test is an empty, so git doesn't identify test as a change

```
PS C:\temp\Understanding\secondrepo> tree /F
Folder PATH listing
Volume serial number is 36E0-F780
C:.
|   readme.txt
|   readmeagain.txt
|
+---src
|       main.py
|
\---test
PS C:\temp\Understanding\secondrepo>
```

```
PS C:\temp\Understanding\secondrepo> git log --oneline
8041f0a (HEAD -> master) Added src
5701166 third commit
9ee3900 second commit
a7fde75 first commit
PS C:\temp\Understanding\secondrepo>
```



Deleting the files which had history (Deleting tracked files)

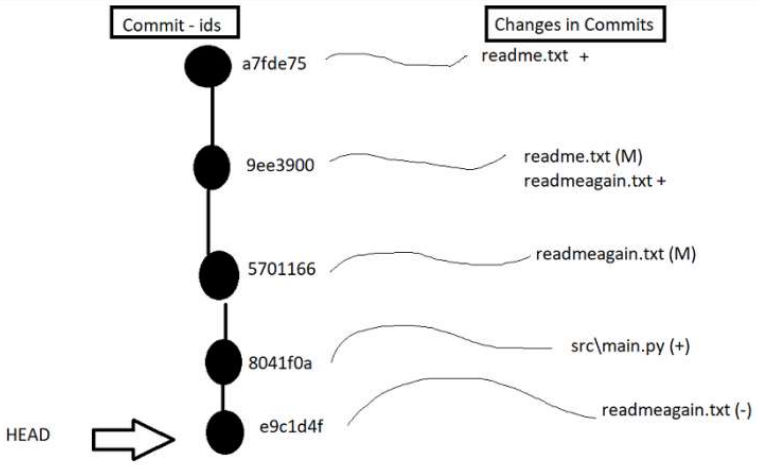
- Deleting a tracked file is considered as change.

- Now we can add the change to staging area and then commit it

```
PS C:\temp\Understanding\secondrepo> git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    readmeagain.txt

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\temp\Understanding\secondrepo> git add -A
PS C:\temp\Understanding\secondrepo>
PS C:\temp\Understanding\secondrepo> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    readmeagain.txt
```

```
PS C:\temp\Understanding\secondrepo> git log --oneline
e9c1d4f (HEAD -> master) Deleted unnecessary stuff
8041f0a Added src
5701166 third commit
9ee3900 second commit
a7fde75 first commit
```

- The history would be as shown below



Making changes in Working tree and removing the changes

- Make some changes which impacts history (tracked files)

```
PS C:\temp\Understanding\secondrepo> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.txt
        modified:   src/main.py

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\temp\Understanding\secondrepo>
```

- Create some untracked files

```
PS C:\temp\Understanding\secondrepo> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.txt
        modified:   src/main.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        New Text Document.txt

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\temp\Understanding\secondrepo>
```

- To clean all the changes.
- Git has different approach to clean
    - tracked files:

- we can use restore commands

```
PS C:\temp\Understanding\secondrepo> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.txt
        modified:   src/main.py

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\temp\Understanding\secondrepo> git restore .\src\main.py
PS C:\temp\Understanding\secondrepo> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.txt
```

```
PS C:\temp\Understanding\secondrepo> git restore .\readme.txt
PS C:\temp\Understanding\secondrepo>
PS C:\temp\Understanding\secondrepo> git status
On branch master
nothing to commit, working tree clean
```

- Execute `git restore --help`
- untracked files `git clean -fd .`

```
PS C:\temp\Understanding\secondrepo> git clean -fd .
Removing New Text Document.txt
Removing test/
PS C:\temp\Understanding\secondrepo> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.txt
        modified:   src/main.py

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\temp\Understanding\secondrepo>
```

Removing Changes from staging area to Working Tree

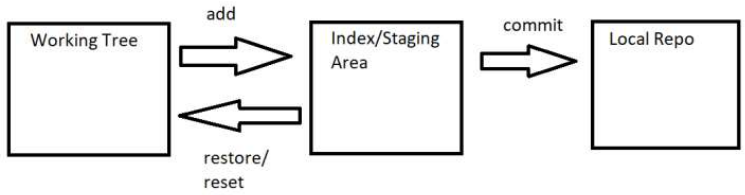
- Lets add changes to staging area. Now i would want to remove the changes

```
PS C:\temp\Understanding\secondrepo> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   readme.txt
        modified:   src/main.py

PS C:\temp\Understanding\secondrepo>
```

```
PS C:\temp\Understanding\secondrepo> git restore --staged .\readme.txt
PS C:\temp\Understanding\secondrepo> git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   src/main.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.txt

PS C:\temp\Understanding\secondrepo> git restore --staged .\src\main.py
PS C:\temp\Understanding\secondrepo>
```

```
PS C:\temp\Understanding\secondrepo> git restore --staged .\src\main.py
PS C:\temp\Understanding\secondrepo>
PS C:\temp\Understanding\secondrepo> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   readme.txt
        modified:   src/main.py

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\temp\Understanding\secondrepo> git restore .
PS C:\temp\Understanding\secondrepo> git status
On branch master
nothing to commit, working tree clean
PS C:\temp\Understanding\secondrepo>
```

- Areas of Git updated



Exercises

- Create a new folder anywhere in your system and make it a git repo
- Create 3 folders and check for status
- now add a file in first folder and then execute git status
- add this change to staging area
- create one more file in second folder and add this change also to staging area
- Now commit the change
- Draw a tree like what we have done in the class `git log --oneline`
- Now create two files in third folder, add them to staging area and now remove one file from staging area and create 2 commit.
- Update your tree
- now add the file which you moved from staging area to working tree back to staging ared and then create a third commit
- Now create two files each in three folders add all of them to staging area
- Clean the above changes as if you have not done any changes after 3 commit.
- Terms:
    - Untracked file: Untracked files are not part of git history, they are new files which are present in working tree

Note

- Creating a VM on AWS Refer Here and Azure Refer Here

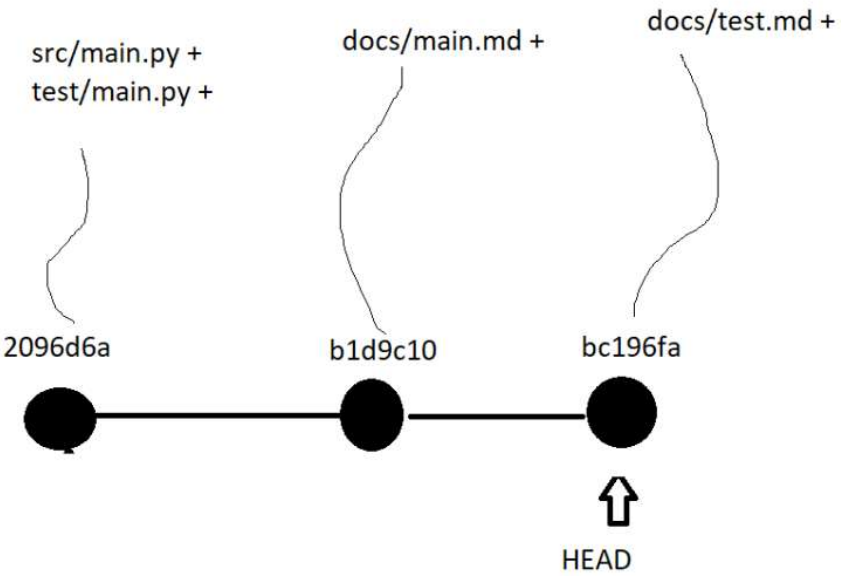
MENU ☰

SEPTEMBER 11, 2022

DevOps Classroomnotes 11/Sep/2022

Exercises

- Create a new folder anywhere in your system and make it a git repo
- Create 3 folders and check for status
- now add a file in first folder and then execute git status
- add this change to staging area
- create one more file in second folder and add this change also to staging area
- Now commit the change
- Draw a tree like what we have done in the class `git log --oneline`
- Now create two files in third folder, add them to staging area and now remove one file from staging area and create 2 commit.
- Update your tree
- now add the file which you moved from staging area to working tree back to staging ared and then create a third commit
- Now create two files each in three folders add all of them to staging area
- Clean the above changes as if you have not done any changes after 3 commit.



Some popular but correct opinions

- Git is a Stupid Content Tracker
- Linus Torvalds has a built a new file system which can track contents
- Lets understand how git works
- When we initialize git repository
    - .git folder gets created
    - HEAD pointer points to a default reference which is master
    - Reference will point to a specific commit id

- Create a first commit on some folder after initalization.

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/fourth_repo
 (master)
$ git log --oneline
946d428 (HEAD -> master) commit 1

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/fourth_repo
 (master)
$
```

- Now lets use a plumbing command `git cat-file`
  - `git cat-file -t` shows the type
  - `git cat-file -p` shows the contents

- Git uses SHA-1 Hash to store objects.
- Git has following object types
  - tree => folder
  - blob => file
  - commit

- Lets find out the type and contents of commit id

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/fourth_repo (master)
$ git cat-file -t 946d428
commit

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/fourth_repo (master)
$ git cat-file -p 946d428
tree d2bd908b5d11f6a1bf7574de0f723f00e9d69272
author shaikkhajaibrahim <qtkhajacloud@gmail.com> 1662903956 +0530
committer shaikkhajaibrahim <qtkhajacloud@gmail.com> 1662903956 +0530

commit 1

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/fourth_repo (master)
$
```

- Now lets find the contents of tree

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/fourth_repo (master
$ git cat-file -p d2bd908b5d11f6a1bf7574de0f723f00e9d69272
100644 blob e69de29bb2d1d6434b8b29ae775ad8c2e48c5391    main.py
```

- Lets find the contents of file i.e. blob main.py

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/fourth_repo (master)
$ git cat-file -p e69de29bb2d1d6434b8b29ae775ad8c2e48c5391

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/fourth_repo (master)
$
```

- main.py is empty
- Now lets some content and create a second commit
- please refer video for the rest of images on how git works.

Git Branches

- Git by default create a branch which is called as `master`
- Branch will point to a latest commit done in it.
- To create a new branch from existing branch `git branch <branch-name>`

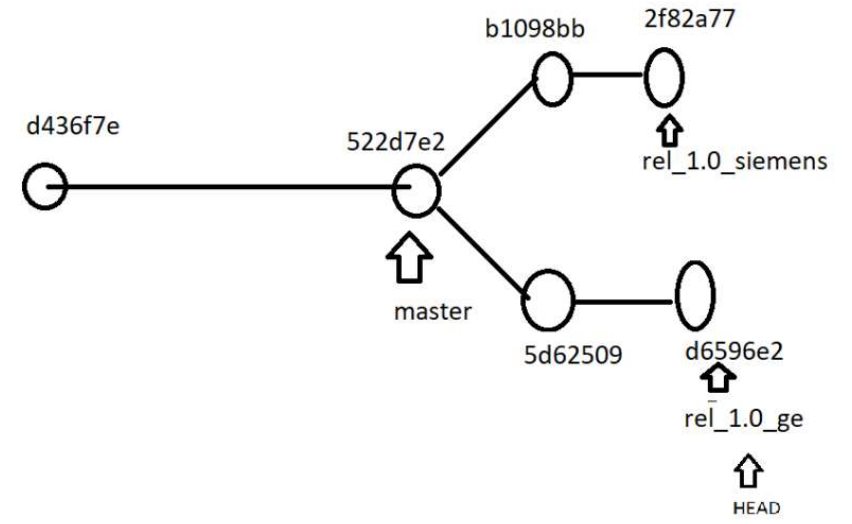
```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_mrs (master)
$ git log --oneline
522d7e2 (HEAD -> master, rel_1.0_siemens) Added test code
d436f7e Added common code

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_mrs (master)
$ git branch rel_1.0_ge

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_mrs (master)
$ git log --oneline
522d7e2 (HEAD -> master, rel_1.0_siemens, rel_1.0_ge) Added test code
d436f7e Added common code

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_mrs (master)
$
```

- Now lets make some changes in `rel_1.0_siemens`, to do this HEAD should be pointing towards `rel_1.0_siemens` for this we use checkout

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_mrs (master)
$ git checkout rel_1.0_siemens
Switched to branch 'rel_1.0_siemens'

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_mrs (rel_1.0_siem
ns)
$ git branch
  master
  rel_1.0_ge
* rel_1.0_siemens

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_mrs (rel_1.0_sieme
ns)
$
```

- Now lets add some changes as discussed in the class

```
$ git log --oneline --graph --all
* d6596e2 (HEAD -> rel_1.0_ge) Added changes
* 5d62509 Added ge
| * 2f82a77 (rel_1.0_siemens) Added test case
| * b1098bb Added company and version
|/
* 522d7e2 (master) Added test code
* d436f7e Added common code
```



- Next Steps:
  - A change done in the master branch should be made available to both siemens and ge branch
  - A commit in ge branch should also be applied to siemens branch
  - Adding multiple users i.e.Remote Repository
    - pull
    - fetch
    - push
    - remote branches
    - remote repos
    - origin
  - All the changes done for siemens and ge to be merged to common branch
  - The above activities can be address
    - merge
    - rebase
    - cherry pick
    - merge – conflicts

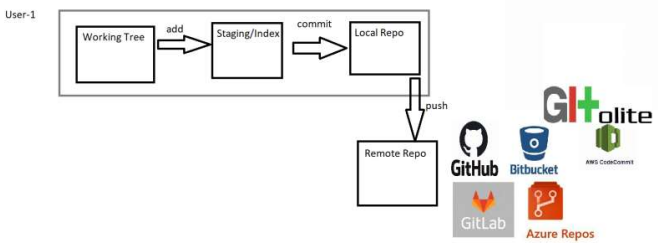
MENU ☰

SEPTEMBER 13, 2022

DevOps Classroomnotes 13/Sep/2022

Fourth Area of Git

- The fourth area of git is Remote Repository
- Remote Repository is a Git with some server features (Connectiviy, User managment)
- Remote Repository can be
    - Self hosted (Where we install on one of our servers)
        - Gitolite
        - Gitlab
        - BitBucket
    - Cloud Hosted (Where the git with server featuers is preinstalled and you just need to configure users, repositories etc)
        - GitHub
        - GitLab
        - BitBucket
        - Azure Source Repos
        - AWS Code Commit



Scenario – 1

- You have started working locally and created some commits
- Now you want this code to be shared with your team members
- We create a Remote Repository & then push the changes from your local repo to remote Repo
- Local Repository with two changes

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_hospital_app (main)
$ git log --oneline
0b7b1d8 (HEAD -> main) Added boilerplate for test and docs
d0ccecc Added boilerplate code

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_hospital_app (main)
$
```

GitHub

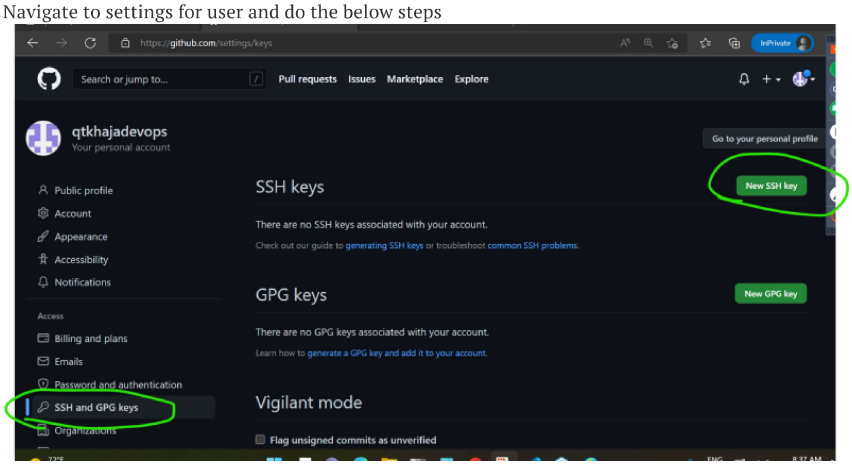
- Create a GitHub Account
- Lets use SSH authentication
- Lets create SSH key pair in your system

```
PS C:\Users\Dell> ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (C:\Users\Dell/.ssh/id_rsa):
```

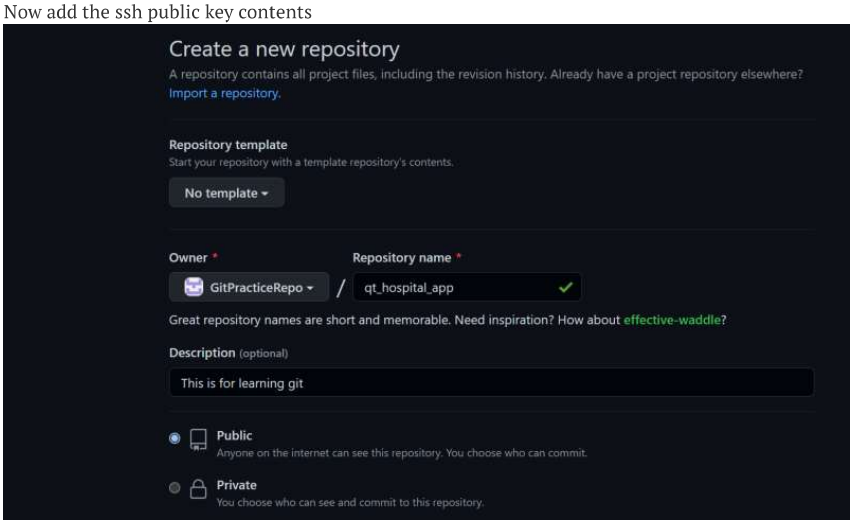
- Press Enter for the next inputs which ssh-keygen asks
- In your home directory (`c:\users\<username>` in windows `/Users/<username>` in mac and `/home/<username>` in linux) a .ssh folder will be created with two files

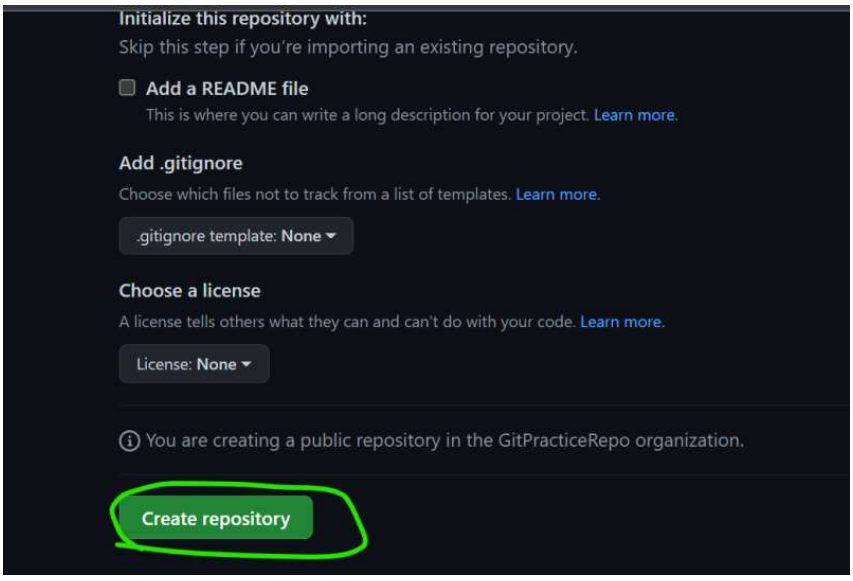
- id_rsa => private key => to keep secure locally
- id_rsa.pub => public key => Git remote repository

- Lets add this ssh key to the GitHub
- Navigate to settings for user and do the below steps

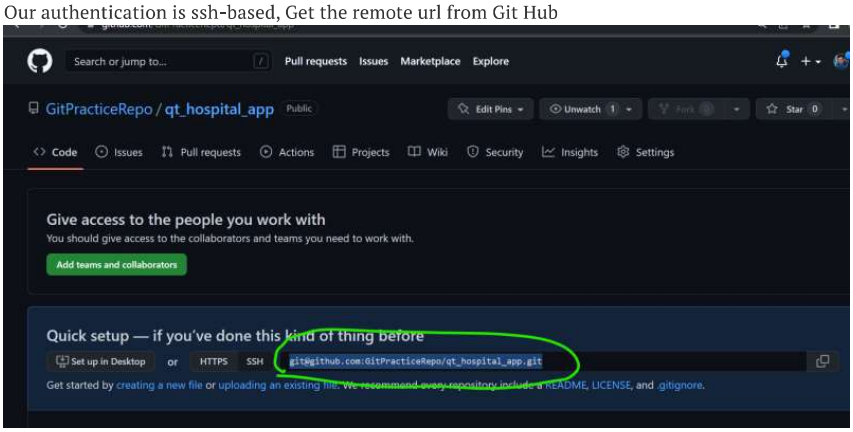


- Now add the ssh public key contents





- Our authentication is ssh-based, Get the remote url from Git Hub

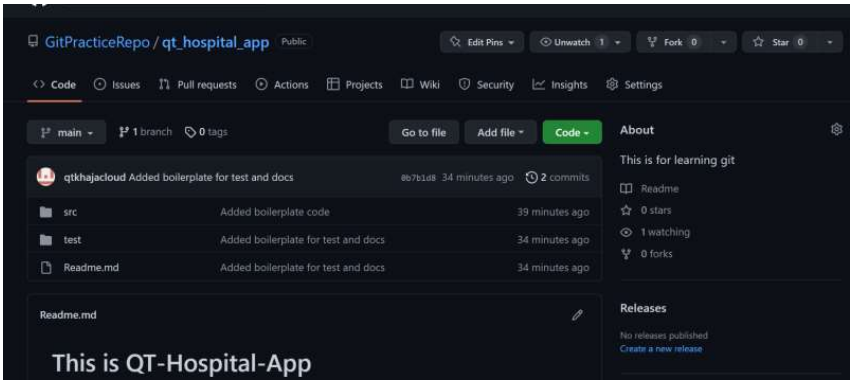


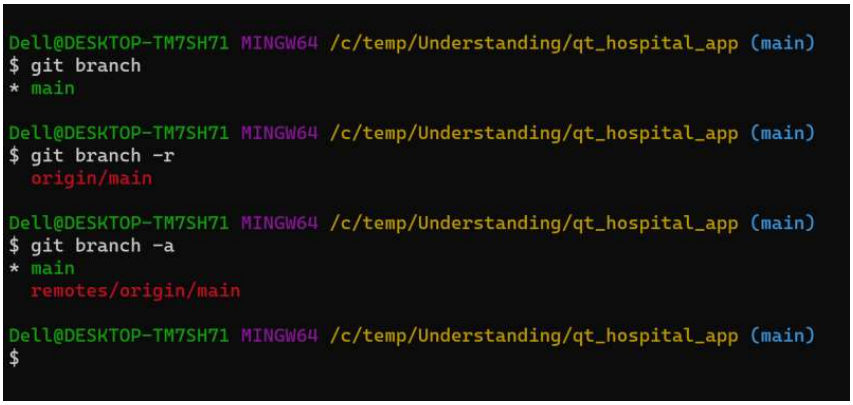
- Remote repository connection can be added to your local repository `git remote add <name-of-remote> <remote-url>`
- The default remote name used is `origin`
- Now my command would be `git remote add origin git@github.com:GitPracticeRepo/qt_hospital_app.git`

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_hospital_app (main)
$ git remote add origin git@github.com:GitPracticeRepo/qt_hospital_app.git

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_hospital_app (main)
$ git remote
origin

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_hospital_app (main)
$
```

- Now we need to push the changes `git push <remote-name> <branch-name>` is the command to push

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_hospital_app (main)
$ git push origin main
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (9/9), 672 bytes | 336.00 KiB/s, done.
Total 9 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To [redacted]/github.com/GitPracticeRepo/qt_hospital_app.git
 * [new branch]      main -> main

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/qt_hospital_app (main)
```





- We have pushed the changes to Github Refer Here and Gitlab Refer Here

Leave a Reply

Enter your comment here...

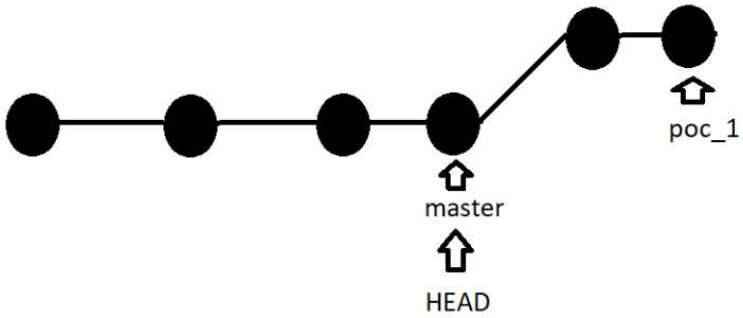
MENU ☰

SEPTEMBER 14, 2022

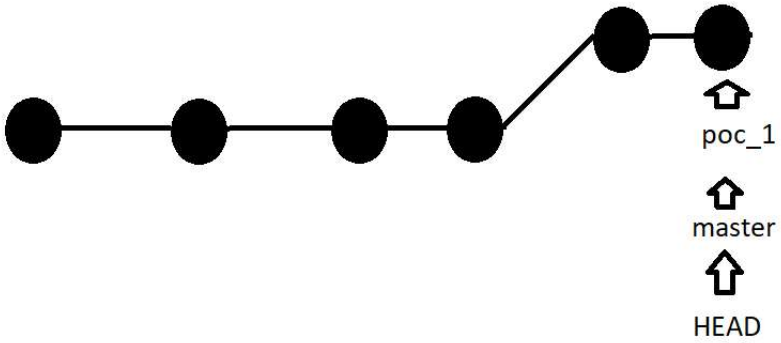
DevOps Classroomnotes 14/Sep/2022

Merging changes between two branches

- Consider the following DAG



- We have create a new branch poc_1.
- Done some commits, the master branch didn't change
- Now when we merge, it would be enough if the master starts pointing towards the latest commit of poc_1 branch.
- This is called as Fast Forward



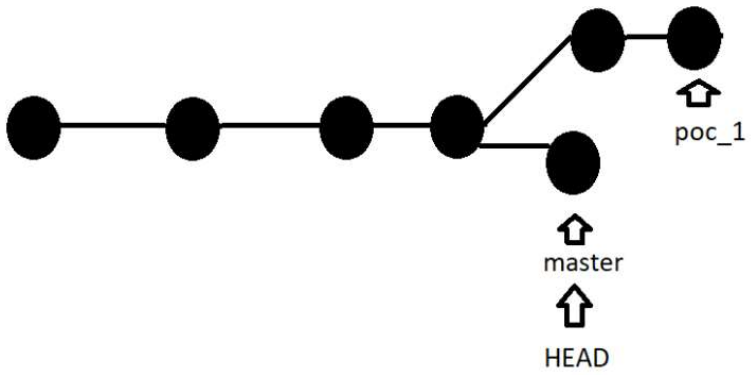
- To merge the changes, switch to the branch you want to merge changes to (destination) and then execute command `git merge <source-branch>`

```
git checkout master
git merge poc_1
```

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-1
poc_1)
$ git checkout master
Switched to branch 'master'

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-1
master)
$ git merge poc_1
Updating 36f2106..0a789e4
Fast-forward
 src/main.py | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)
```

* We can skip fast forwarding `git merge <source-branch> --no-ff`

- Consider the following DAG



- Git history

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-2
master)
$ git log --oneline --graph --all
* b69dfa6 (HEAD -> master) commit 5
| * 0a789e4 (poc_1) poc-commit-2
| * 5ee6510 poc-commit 1
|/
* 36f2106 Commit 4
* 4f1cfac commit-3
* 500b1bb commit 2
* 18d7fba commit 1
```

- Now we want changes done in poc_1 branch onto master
- When merging the changes, automatic merges by git will fail when there are conflicting situations which are referred as merge-conflicts

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-2 (
poc_1)
$ git checkout master
Switched to branch 'master'

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-2 (
master)
$ git merge poc_1
Auto-merging src/main.py
CONFLICT (content): Merge conflict in src/main.py
Automatic merge failed; fix conflicts and then commit the result.

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-2 (
master|MERGING)
```

```
1  # This is main file
2  # author: QT
3  <<<<<<< HEAD
4  # This is master change -1
5  =======
6  # poc_1
7  # This is working
8  >>>>>>> poc_1
9
```

- Fix the changes

C:\temp\Understanding\merge-scenario-2\src\main.py - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window

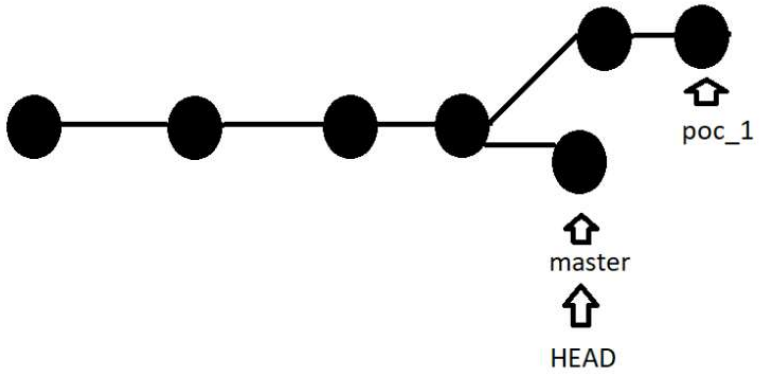
new 120 | new 121 | 10kSamplejson.json | new 122 | new 123 | new 124 | ne

```
1  # This is main file
2  # author: QT
3  # This is master change -1
4  # poc_1
5  # This is working
6
7
```

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-2 (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   src/main.py

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

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-2 (master|MERGING)
$ git add .

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-2 (master|MERGING)
$ git commit
[master 6c4d970] Merge branch 'poc_1'

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-2 (master)
$
```

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-2 (master)
$ git log --oneline --graph --decorate
*   6c4d970 (HEAD -> master) Merge branch 'poc_1'
|\
| * 0a789e4 (poc_1) poc-commit-2
| * 5ee6510 poc-commit 1
* | b69dfa6 commit 5
|/
* 36f2106 Commit 4
* 4f1cfac commit-3
* 500b1bb commit 2
* 18d7fba commit 1

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-2 (master)
$ git cat-file -p 6c4d970
tree be3534d725a6a5a62a4dabb9a2098e2a58f2bb08
parent b69dfa6f394ad5799ec3bf91d7cbc86794c0485d
parent 0a789e45a441c8da0a0d37e108c70b9695dfbacf
author shaikkhajaibrahim <qtkhajacloud@gmail.com> 1663125186 +0530
committer shaikkhajaibrahim <qtkhajacloud@gmail.com> 1663125186 +0530

Merge branch 'poc_1'
```

- This way of merging is referred as three way merge
- Consider the following DAG



- After create the POC_1 from master branch, an important commit that needs to be part of POC_1 is committed to master.

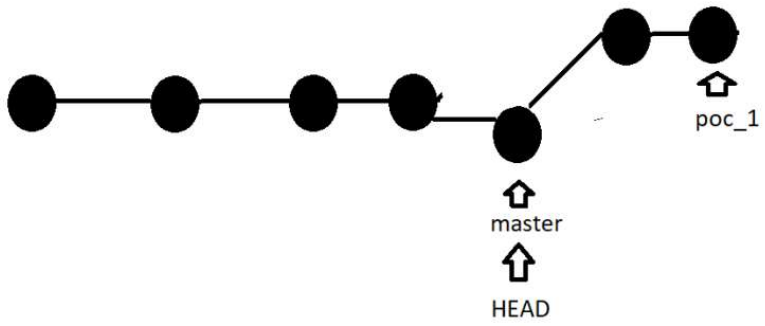
```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-3 (poc_1)
$ git rebase master
Auto-merging src/main.py
CONFLICT (content): Merge conflict in src/main.py
error: could not apply 5ee6510... poc-commit 1
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply 5ee6510... poc-commit 1
```

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-3 (poc_1|REBASE 1/2)
$ git status
interactive rebase in progress; onto e6a8ae8
Last command done (1 command done):
   pick 5ee6510 poc-commit 1
Next command to do (1 remaining command):
   pick 0a789e4 poc-commit-2
  (use "git rebase --edit-todo" to view and edit)
You are currently rebasing branch 'poc_1' on 'e6a8ae8'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original branch)

Unmerged paths:
  (use "git restore --staged <file>..." to unstage)
  (use "git add <file>..." to mark resolution)
        both modified:   src/main.py

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

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-3 (poc_1|REBASE 1/2)
$ git add .

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-3 (poc_1|REBASE 1/2)
$ git rebase --continue
[detached HEAD 6502e80] poc-commit 1
 1 file changed, 3 insertions(+), 1 deletion(-)
Auto-merging src/main.py
CONFLICT (content): Merge conflict in src/main.py
error: could not apply 0a789e4... poc-commit-2
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply 0a789e4... poc-commit-2

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-3 (poc_1|REBASE 2/2)
$ git add .

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-3 (poc_1|REBASE 2/2)
$ git rebase --continue
```

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-3 (poc_1|REBASE 2/2
$ git rebase --continue
[detached HEAD 3cd61d0] poc-commit-2
 1 file changed, 1 insertion(+)
Successfully rebased and updated refs/heads/poc_1.

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-3 (poc_1)
$

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-3 (poc_1)
$ git log --oneline --graph --all
* 3cd61d0 (HEAD -> poc_1) poc-commit-2
* 6502e80 poc-commit 1
* e6a8ae8 (master) commit 5
* 36f2106 Commit 4
* 4f1cfac commit-3
* 500b1bb commit 2
* 18d7fba commit 1

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-3 (poc_1)
```

- DAG



-

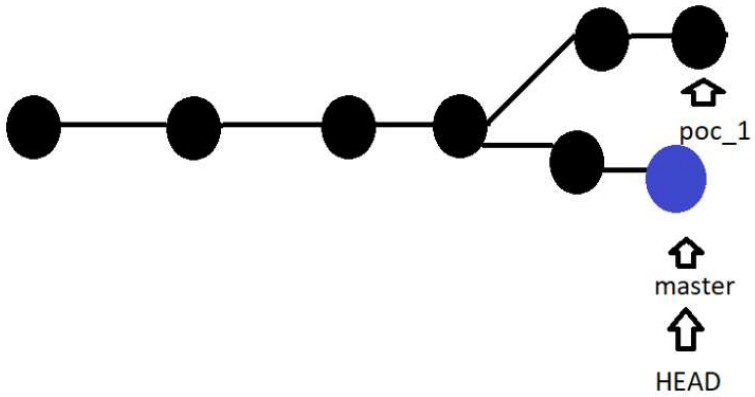
MENU ☰

SEPTEMBER 15, 2022

DevOps Classroomnotes 15/Sep/2022

Cherry-Pick

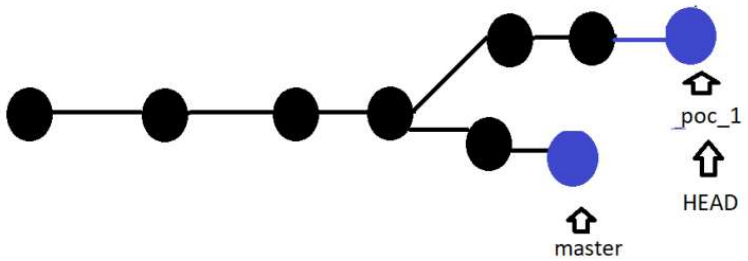
- Consider the following DAG, we need changes in the blue commit on the POC_1 branch



- If we need specific commits or sequence of commits from other branches

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/merge-scenario-4 (master)
$ git log --oneline --graph --all
* 5ed34d0 (HEAD -> master) commit 6
* 82a239b commit 5
| * 0a789e4 (poc_1) poc-commit-2
| * 5ee6510 poc-commit 1
|/
* 36f2106 Commit 4
* 4f1cfac commit-3
* 500b1bb commit 2
* 18d7fba commit 1
```

- Get the commit id of commit 6 (blue commit) `5ed34d0`
- now checkout to target branch `git checkout <target branch>` and To cherry-pick use `git cherry-pick <commit-id>`



- If we get any merge conflicts, resolve add and continue cherry pick like rebase.
- Exercise: Find how to cherry pick range of commits

Bring Remote into Action

- For this demonstration
    - User 1:
        - all the images from powershell will be user 1

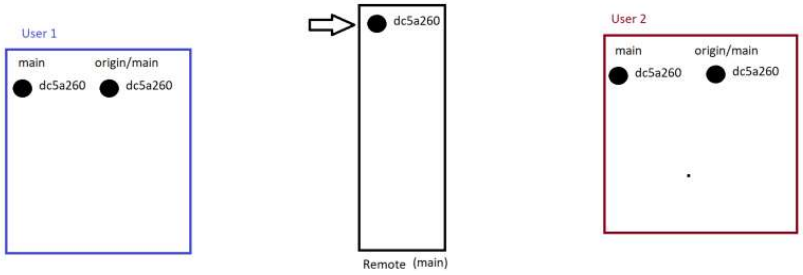
    - config: username: qtdevops
    - `c:\temp\user1`
- User 2: all the images from git bash will be user 2
    - config: username: qtcloud
    - `c:\temp\user2`

- Lets create a new git repository with some content
- We already have remote repository, user1 and user2 want to work, so they need local repos.
- Creating a local repo when we already have a remote is `clone`

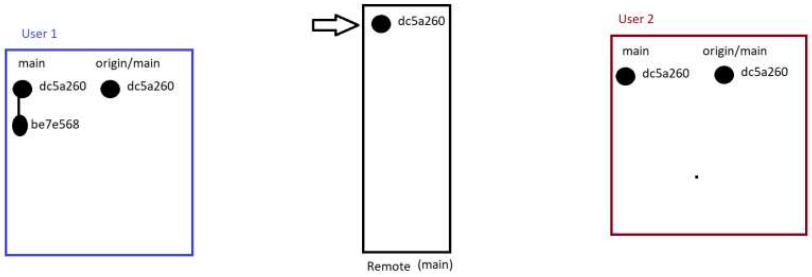
```
PS C:\temp> cd .\user1\
PS C:\temp\user1> git clone https://github.com/GitPracticeRepo/UnderstandingGitSep22
git
Cloning into 'UnderstandingGitSep22'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), 4.51 KiB | 4.51 MiB/s, done.
PS C:\temp\user1>
```

- we will have local and remote branches on local repo

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/user2
$ cd UnderstandingGitSep22/

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/user2/UnderstandingGitSep22 (main)
$ git branch -a
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/main

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/user2/UnderstandingGitSep22 (main)
$
```

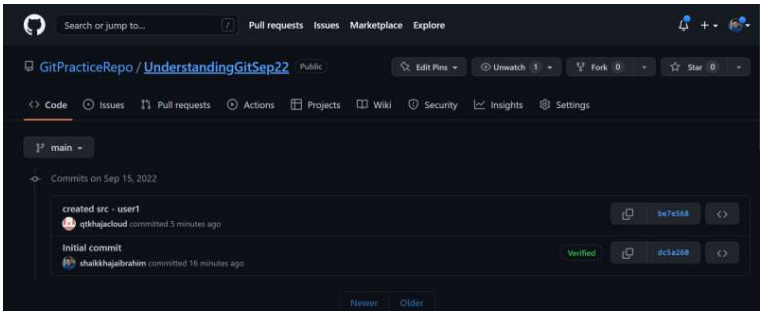


- Now User 1 makes a change, commits to local rep

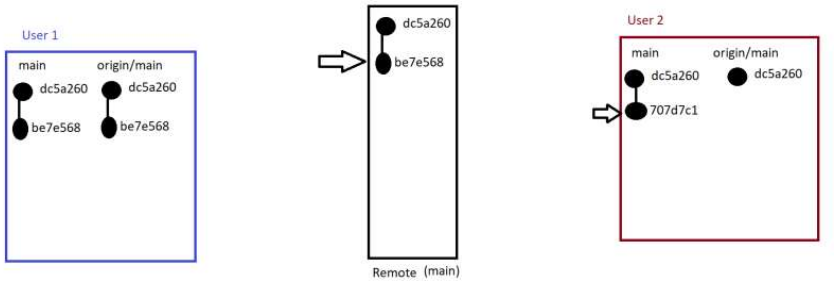


- Now user 1 wants to push the changes to remote repository
    - origin/main and remote repository main commit id should match
    - Since they are matching push `git push <remote name> <branch name>`

```
PS C:\temp\user1\UnderstandingGitSep22> git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 360 bytes | 360.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/GitPracticeRepo/UnderstandingGitSep22.git
   dc5a260..be7e568  main -> main
PS C:\temp\user1\UnderstandingGitSep22>
```



- Now User 2 wants to push a change (he is unaware of change done by user 1)
- User 2 does the local commit
- The below image represents the current situation

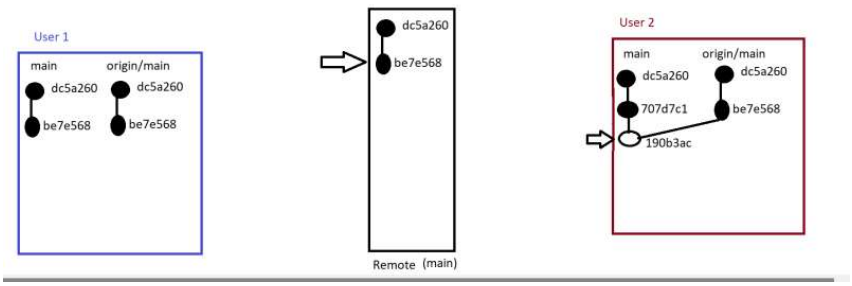


- When User 2 pushes he gets the following error message

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/user2/UnderstandingGitSep22 (main)
$ git push origin main
To https://github.com/GitPracticeRepo/UnderstandingGitSep22.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/GitPracticeRepo/Understanc
ingGitSep22.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/user2/UnderstandingGitSep22 (main)
$
```

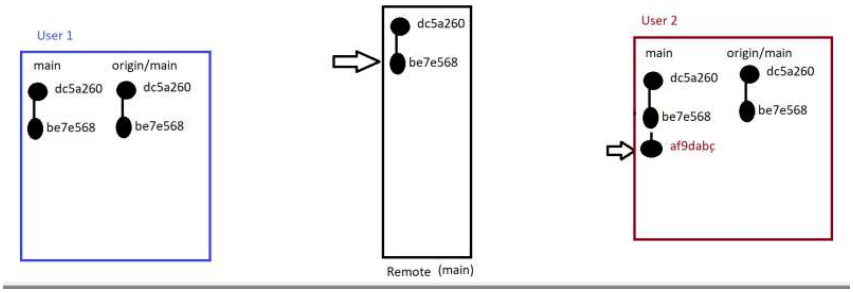
- So pull the changes (pull => Fetch + merge)

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/user2/UnderstandingGitSep22 (main)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 0), reused 4 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 340 bytes | 48.00 KiB/s, done.
From https://github.com/GitPracticeRepo/UnderstandingGitSep22
   dc5a260..be7e568  main       -> origin/main
Merge made by the 'ort' strategy.
 src/main.py | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 src/main.py
```



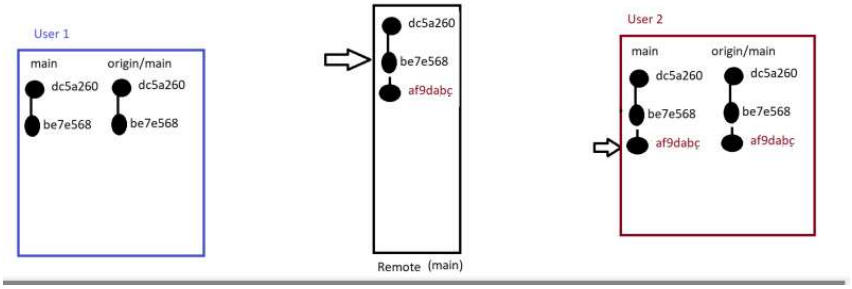
- Here we get an extra merge commit

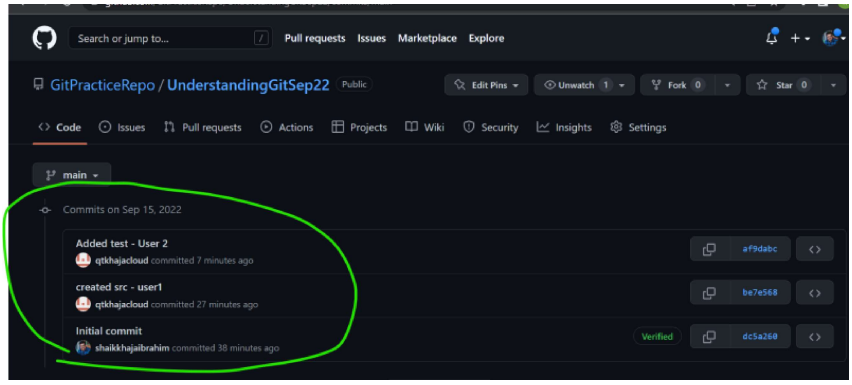
- Alternative to make commit history clean we can use `git pull --rebase`



- Now lets push the changes

```
$ git log --oneline
af9dabc (HEAD -> main) Added test - User 2
be7e568 (origin/main, origin/HEAD) created src - user1
dc5a260 Initial commit

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/user2/UnderstandingGitSep22 - Copy (main)
$ git push origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 252 bytes | 252.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/GitPracticeRepo/UnderstandingGitSep22.git
   be7e568..af9dabc  main -> main

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/user2/UnderstandingGitSep22 - Copy (main)
$
```





- Refer Here for commits on github

MENU ☰

SEPTEMBER 16, 2022

DevOps Classroomnotes 16/Sep/2022

Rewriting History

- Consider the following history

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master
)
$ git log --oneline
9fe050f (HEAD -> master) comit 3
e181f8f commit 2
c44df96 commit 1
```

- For the third commit i.e. your latest commit the commit there is typo in commit message

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master
)
$ git commit --amend
[master f25b7a3] commit 3
 Date: Fri Sep 16 08:08:27 2022 +0530
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 3.txt

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master
)
$ git log --oneline
f25b7a3 (HEAD -> master) commit 3
e181f8f commit 2
c44df96 commit 1
```

- consider the following history

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master
)
$ git log --oneline
14a3d39 (HEAD -> master) commit 5
ea4eb85 fourth commit
f25b7a3 commit 3
e181f8f commit 2
c44df96 commit 1
```

- To rewrite history we need to back by 2 commits and rewrite history `git rebase -i HEAD~2`. This is called as interactive rebase.

```
pick ea4eb85 fourth commit
pick 14a3d39 commit 5

# Rebase f25b7a3..14a3d39 onto f25b7a3 (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previ
#                    commit's log message, unless -C is used, in whi
ase
#                    keep only this commit's message; -c is same as
ut
<tory/.git/rebase-merge/git-rebase-todo [unix] (08:22 16/09/2022)1,1
<istory/.git/rebase-merge/git-rebase-todo" [unix] 30L, 1327B
```

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master)
$  git log --oneline
dd33c6d (HEAD -> master) commit 5
85f7855 commit 4
f25b7a3 commit 3
e181f8f commit 2
c44df96 commit 1

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master)
$
```

- Lets change all the commit messages and add my i.e my commit 1

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master)
$ git log --oneline
a9199be (HEAD -> master) my commit 5
5dffc71 my commit 4
7115adc my commit 3
42e9dbd my commit 2
c44df96 commit 1
```

- combine 3 and 4 commits into one => squash

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master)
$ git log --oneline
3a34eb3 (HEAD -> master) my commit 5
e72bb0a my commit 3 & 4
42e9dbd my commit 2
c44df96 commit 1
```

- Now lets try deleting the commit 3 & 4 which was squashed
- Before delete

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master)
$ git log --oneline
3a34eb3 (HEAD -> master) my commit 5
e72bb0a my commit 3 & 4
42e9dbd my commit 2
c44df96 commit 1
```

- Command `git rebase -i HEAD~3`

```
pick 42e9dbd my commit 2
drop e72bb0a my commit 3 & 4
pick 3a34eb3 my commit 5

# Rebase c44df96..3a34eb3 onto c44df96 (3 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit me
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous
# f, fixup [-C | -c] <commit> = like "squash" but keep on
#                    commit's log message, unless -C is u
ase
@@@
<y/.git/rebase-merge/git-rebase-todo[+] [unix] (08:33 16/
-- INSERT --
```

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master)
$ git log --oneline
bb5e581 (HEAD -> master) my commit 5
42e9dbd my commit 2
c44df96 commit 1

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master)
$ start .

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master)
$ ls
1.txt  2.txt  5.txt

Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master)
$
```

- Impact of rewriting history when working with remote repositories
- Ensure you do the rewriting history either with teams approval
- Git maintains one more log which is referred as reflog

```
Dell@DESKTOP-TM7SH71 MINGW64 /c/temp/Understanding/history (master)
$ git reflog
bb5e581 (HEAD -> master) HEAD@{0}: rebase (finish): returning to refs/heads/maste
r
bb5e581 (HEAD -> master) HEAD@{1}: rebase (pick): my commit 5
42e9dbd HEAD@{2}: rebase (start): checkout HEAD~3
3a34eb3 HEAD@{3}: rebase (finish): returning to refs/heads/master
3a34eb3 HEAD@{4}: rebase (pick): my commit 5
e72bb0a HEAD@{5}: rebase (squash): my commit 3 & 4
7115adc HEAD@{6}: rebase (start): checkout HEAD~3
a9199be HEAD@{7}: rebase (finish): returning to refs/heads/master
a9199be HEAD@{8}: rebase (reword): my commit 5
8388ff1 HEAD@{9}: rebase (reword): commit 5
5dffc71 HEAD@{10}: rebase (reword): my commit 4
f2e7986 HEAD@{11}: rebase (reword): commit 4
7115adc HEAD@{12}: rebase (reword): my commit 3
aab5fe1 HEAD@{13}: rebase (reword): commit 3
42e9dbd HEAD@{14}: rebase (reword): my commit 2
```

- Exercise:
    - Create a new local repostory
    - create 3 commits (commit 1, commit 2, commit 3)
    - note commit id of commit 2
    - do interactive rebase to delete commit 2
    - now check git log
    - Use reflog to recover the deleted commit
- Ignoring some files and folders to be tracked:
    - Refer Here

MENU ☰

SEPTEMBER 17, 2022

DevOps Classroomnotes 17/Sep/2022

Jenkins Contd…

- Installing Jenkins on Centos Refer Here

```
sudo yum install wget -y
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.key
sudo yum upgrade
# Add required dependencies for the jenkins package
sudo yum install java-11-openjdk -y
sudo yum install jenkins -y
sudo systemctl daemon-reload
sudo systemctl enable jenkins
sudo systemctl start jenkins
```

Jenkins Terms

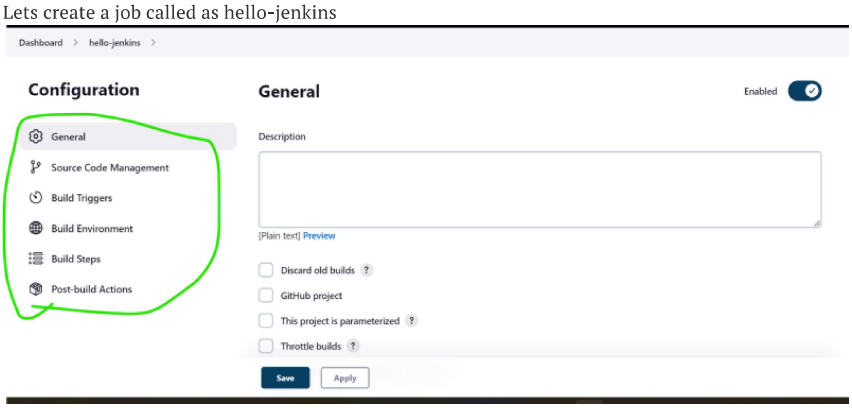
- Project: This is where we configure jenkins to do some activity (ci/cd pipeline).
- Free Style Project: This is project which was present in jenkins from day 1, It relies on UI elements provided.
- Plugin:
    - UI: Jenkins Plugin provides additional UI elements to minimize the work.
- Jenkins Home Directory: Jenkins stores all the information about the projects, jobs, plugins etc in the home directory of jenkins. If we consider taking backup of jenkins, its all about backing up jenkins home directory

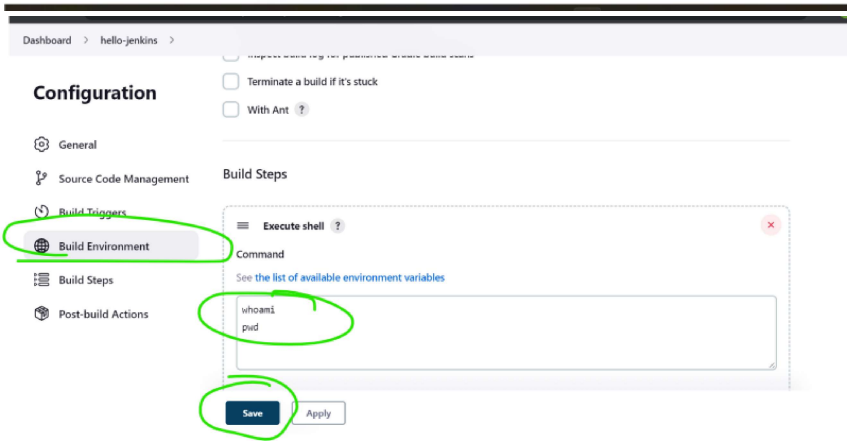
```
bash-4.2$ tree -L 1
.
├── %C
├── config.xml
├── hudson.model.UpdateCenter.xml
├── hudson.plugins.git.GitTool.xml
├── identity.key.enc
├── jenkins.install.InstallUtil.lastExecVersion
├── jenkins.install.UpgradeWizard.state
├── jenkins.model.JenkinsLocationConfiguration.xml
├── jenkins.telemetry.Correlator.xml
├── jobs
├── nodeMonitors.xml
├── nodes
├── plugins
├── secret.key
├── secret.key.not-so-secret
├── secrets
├── updates
├── userContent
└── users

8 directories, 11 files
```

- WorkSpace: Workspace is the folder that stores all the files folders etc created during job execution.

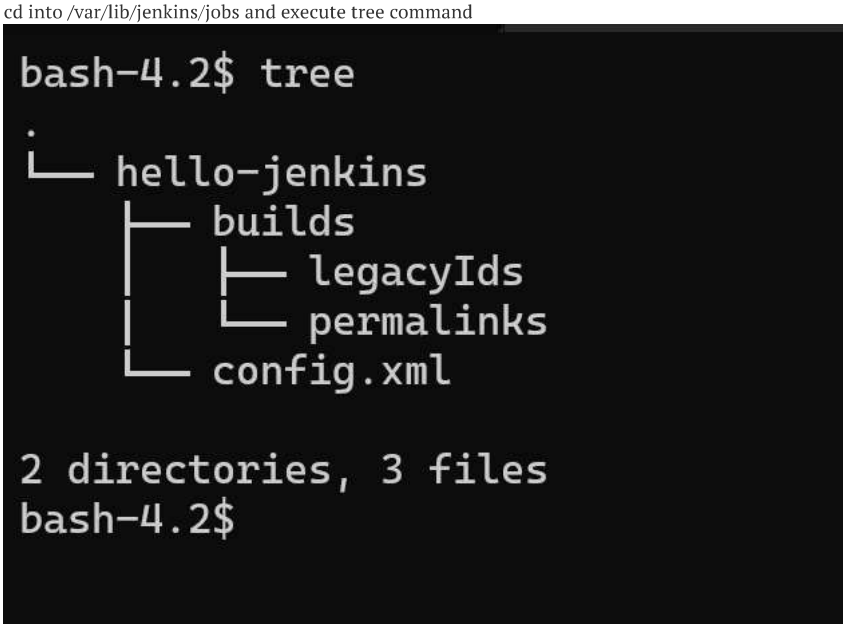
Experiment 1: Create a new Jenkins Project hello-jenkins

- Lets create a job called as hello-jenkins





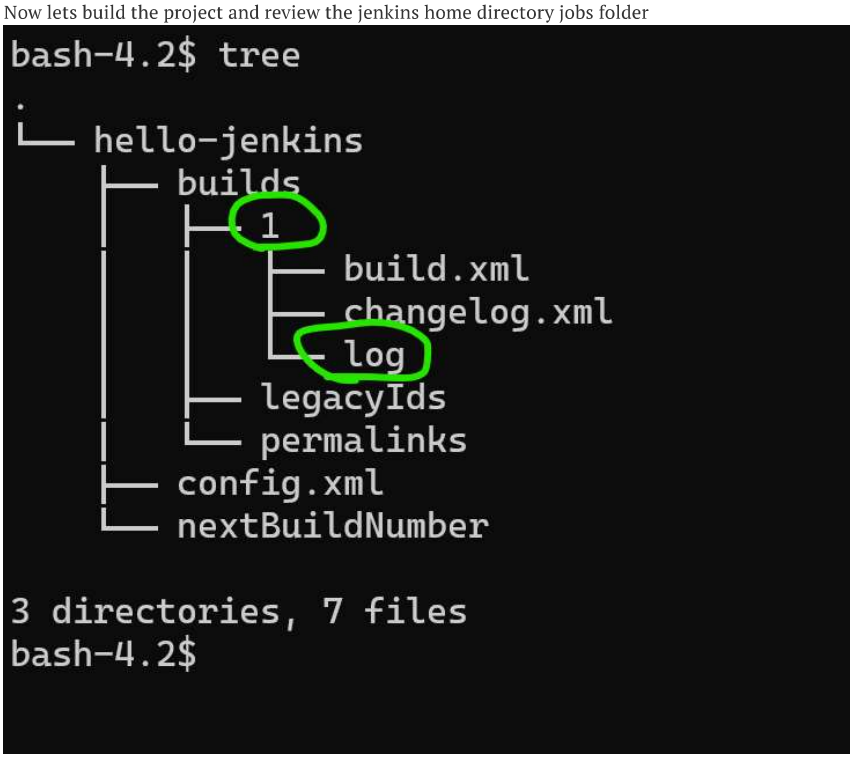
- cd into /var/lib/jenkins/jobs and execute tree command



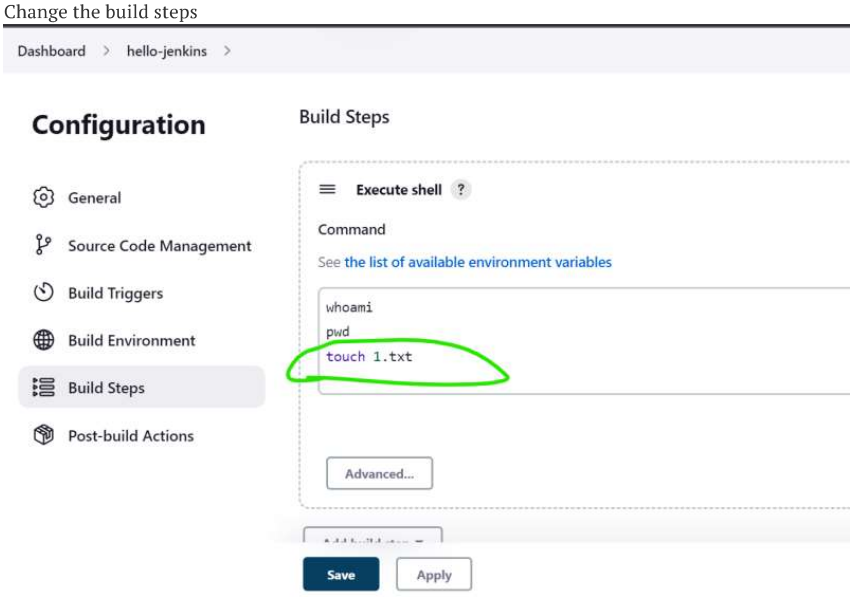
- Let preview config.xml

```
<?xml version='1.1' encoding='UTF-8'?>
<project>
  <actions/>
  <description></description>
  <keepDependencies>false</keepDependencies>
  <properties/>
  <scm class="hudson.scm.NullSCM"/>
  <canRoam>true</canRoam>
  <disabled>false</disabled>
  <blockBuildWhenDownstreamBuilding>false</blockBuildWhenDownstreamBuilding>
  <blockBuildWhenUpstreamBuilding>false</blockBuildWhenUpstreamBuilding>
  <triggers/>
  <concurrentBuild>false</concurrentBuild>
  <builders>
    <hudson.tasks.Shell>
      <command>whoami
pwd</command>
```

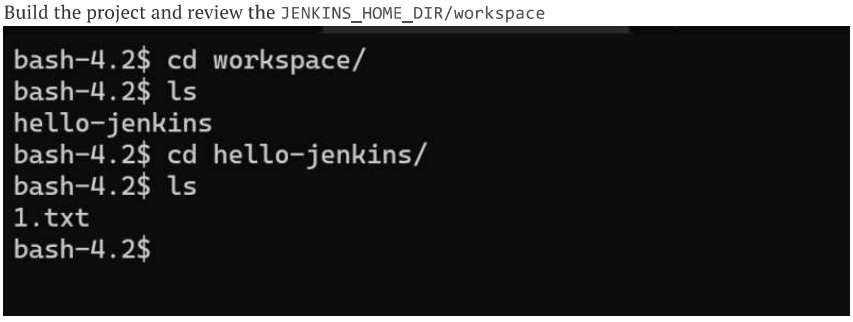
- Now lets build the project and review the jenkins home directory jobs folder

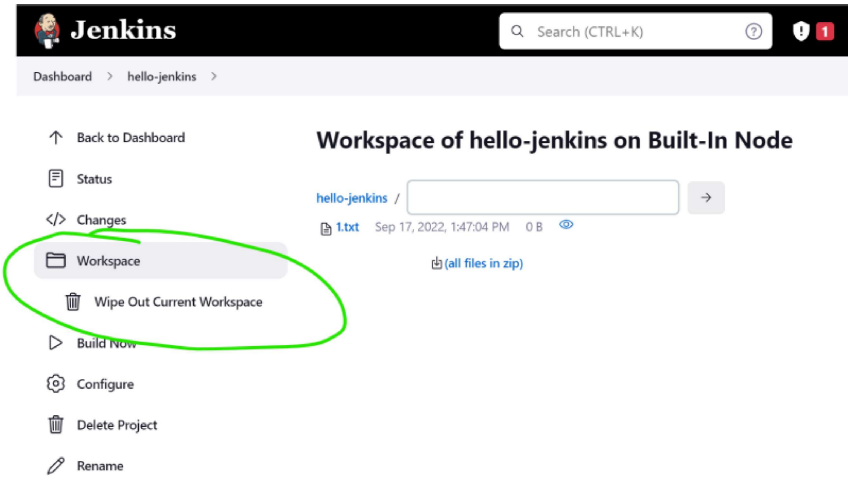


- Change the build steps



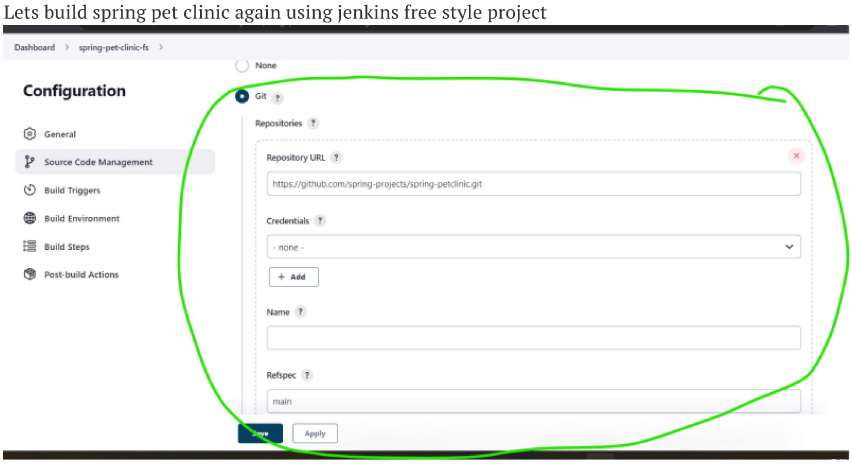
- Build the project and review the `JENKINS_HOME_DIR/workspace`

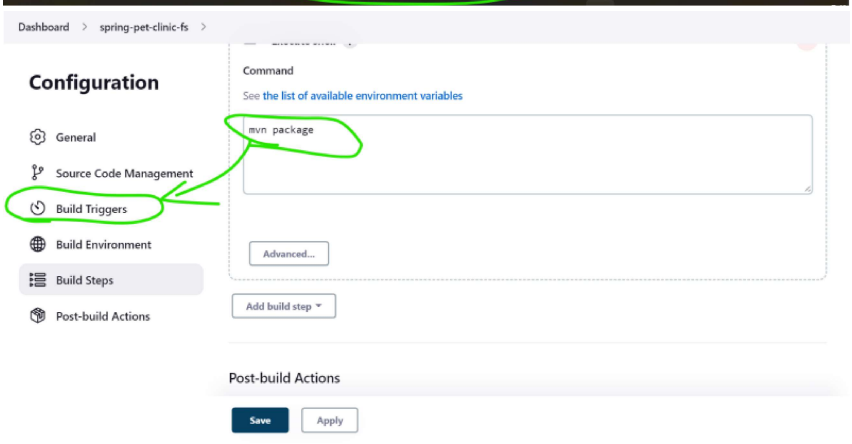


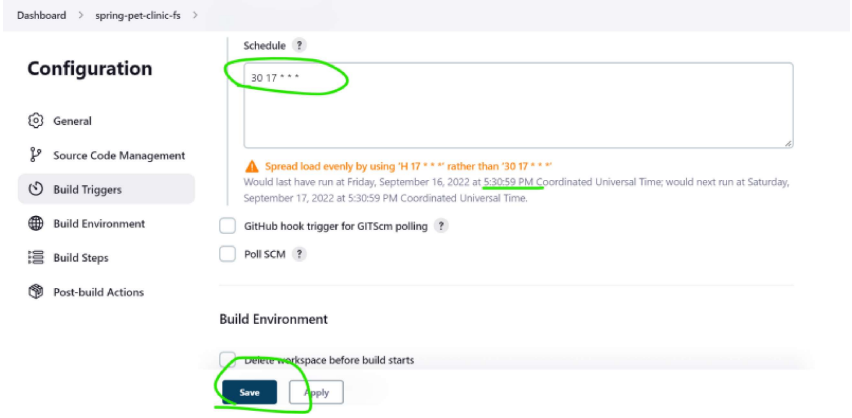


Activity 1: Building a Java Project

- Lets build spring pet clinic again using jenkins free style project

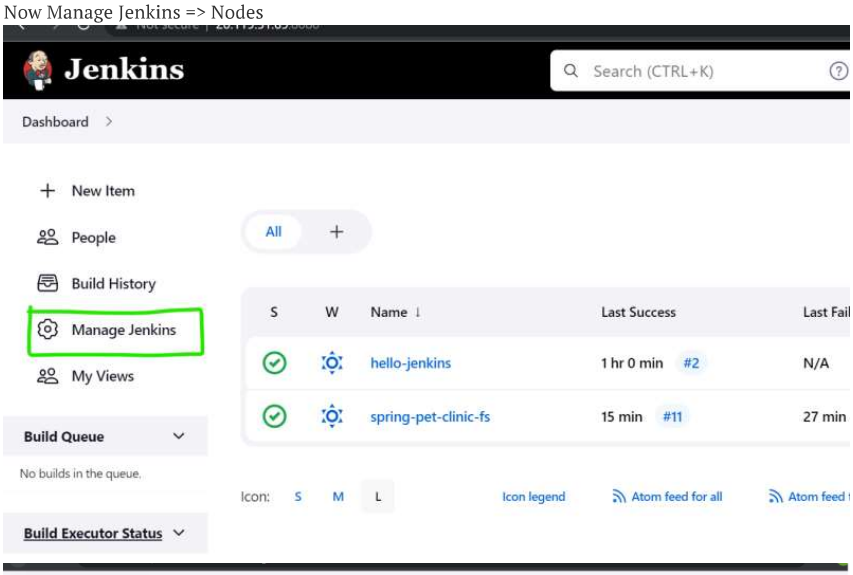


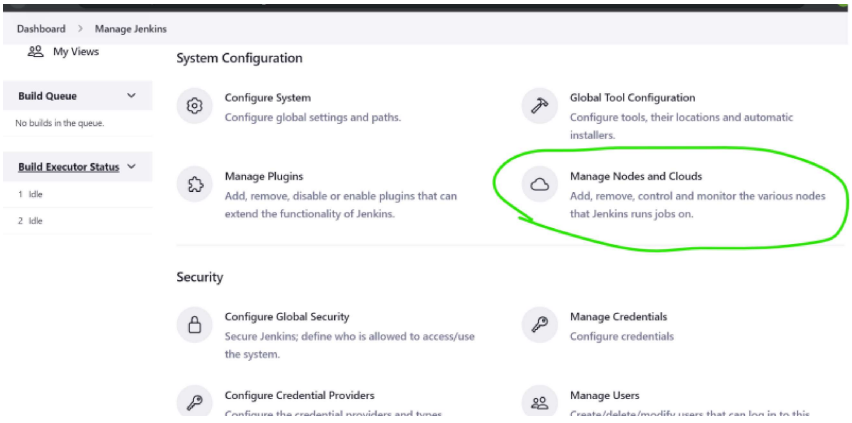


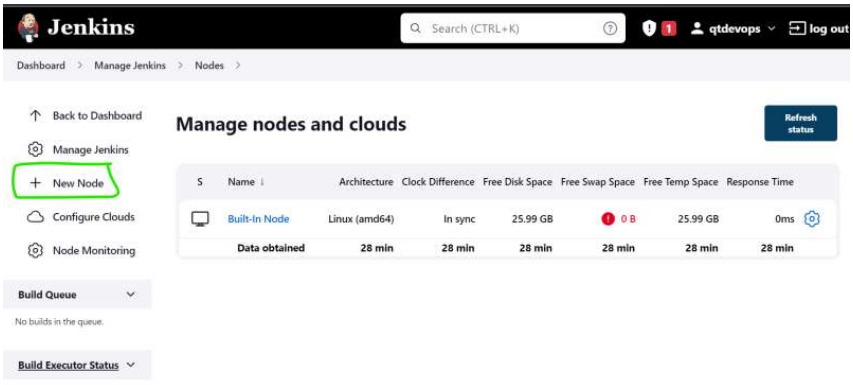


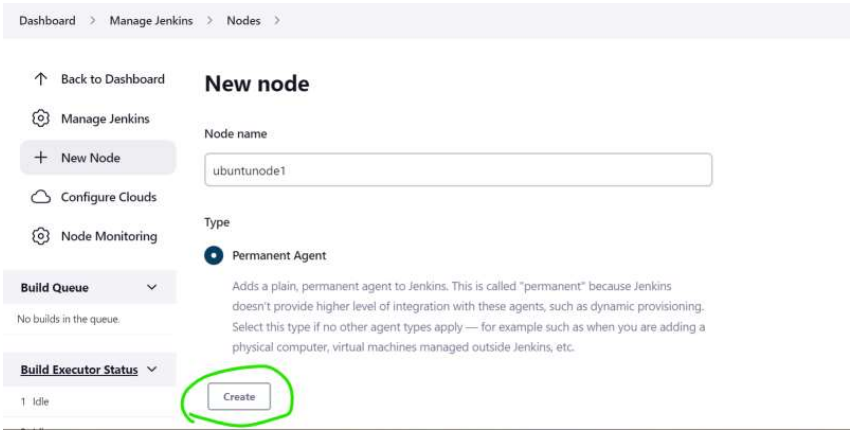
Understanding Distributed Build Concept

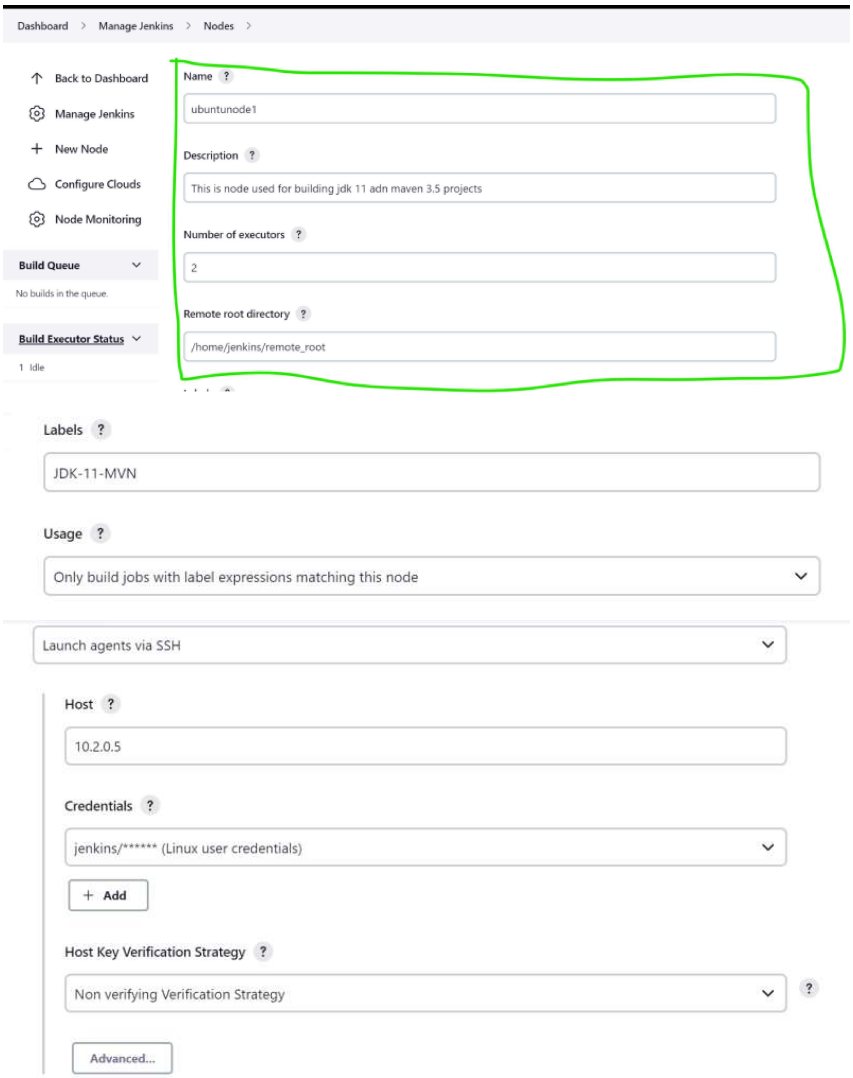
- Jenkins build execute on Nodes. Each node has executors which define number of jobs that can be executed on a node
- Nodes are of two types
    - Master Node:
        - Responsible for hosting jenkins UI and distributing jobs, managing users, plugins etc
        - Ideally Not a good idea to run jobs on master apart from maintenace jobs
    - Nodes:
        - For every project/every technology/os types organizations add nodes to the master.
        - When we add nodes number of executors increase which in turn means we can manage multiple projects from same jenkins server.
- Since we have a jenkins server which is linux based, lets create one more node which is also linux.
- To communicate from one linux node to other linux node, we use ssh.
- Lets use the same concept for configuring node to master
- Node which we have configure takes username and password as credentials.
- Now Manage Jenkins => Nodes

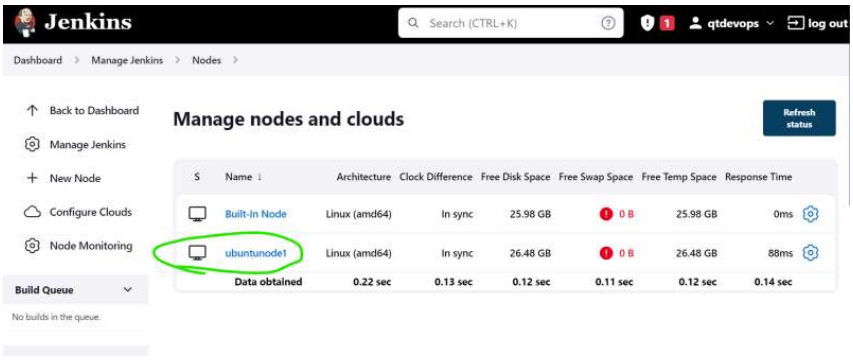


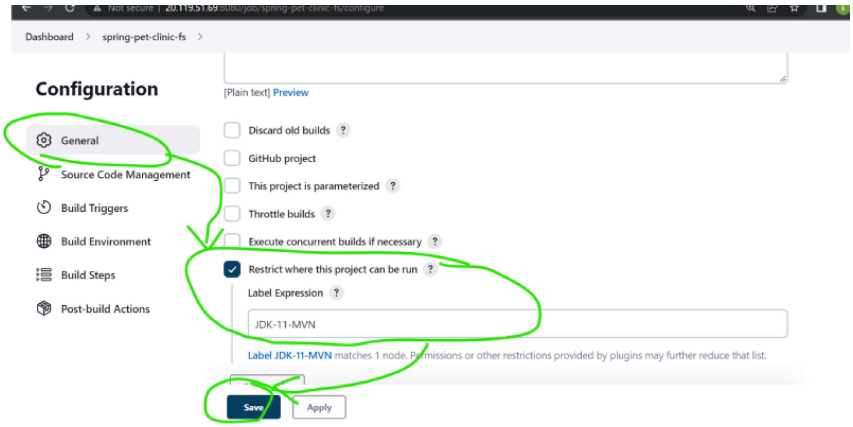



Jenkins
Search (CTRL+K)
qtdevops
log out
Dashboard > Manage Jenkins > Nodes >
Back to Dashboard
Manage Jenkins
New Node
Configure Clouds
Node Monitoring
Build Queue
No builds in the queue.
Build Executor Status
1 Idle
Manage nodes and clouds
Refresh status
S
Name
Architecture
Clock Difference
Free Disk Space
Free Swap Space
Free Temp Space
Response Time
Built-In Node
Linux (amd64)
In sync
25.99 GB
0 B
25.99 GB
0ms
Data obtained
28 min
28 min
28 min
28 min
28 min
28 min

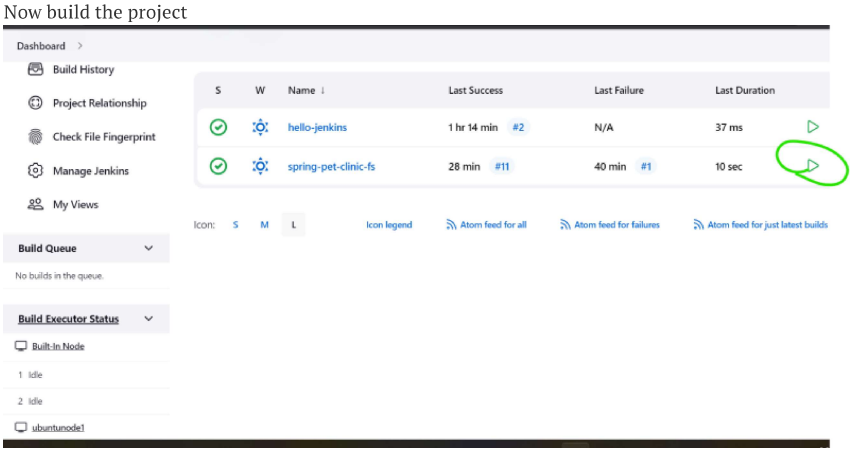

Dashboard > Manage Jenkins > Nodes >
Back to Dashboard
Manage Jenkins
New Node
Configure Clouds
Node Monitoring
Build Queue
No builds in the queue.
Build Executor Status
1 Idle
New node
Node name
ubuntunode1
Type
Permanent Agent
Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.
Create

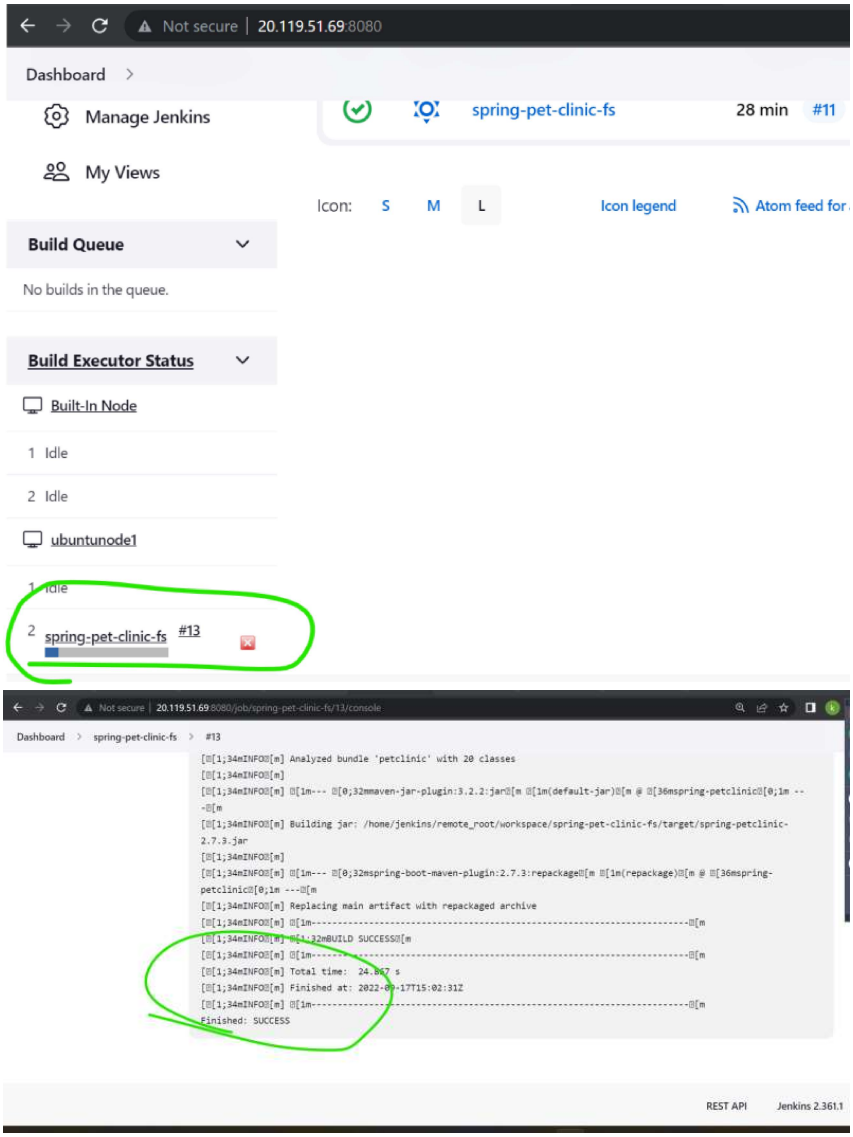

Dashboard > Manage Jenkins > Nodes >
Back to Dashboard
Manage Jenkins
New Node
Configure Clouds
Node Monitoring
Build Queue
No builds in the queue.
Build Executor Status
1 Idle
Name
ubuntunode1
Description
This is node used for building jdk 11 adn maven 3.5 projects
Number of executors
2
Remote root directory
/home/jenkins/remote_root
Labels
JDK-11-MVN
Usage
Only build jobs with label expressions matching this node
Launch agents via SSH
Host
10.2.0.5
Credentials
jenkins/****** (Linux user credentials)
Add
Host Key Verification Strategy
Non verifying Verification Strategy
Advanced...





- Now build the project





MENU ☰

SEPTEMBER 18, 2022

DevOps Classroomnotes 18/Sep/2022

Maven

- Maven is one of most popular build and project management tool for Java and Java Based Languages
- The Other build tools for java include Ant, Gradle.
- Any Language has lot of built in functionalities which are provided by language or community, These are referred as dependencies.
- We need to perform dependency management.
- Maven used the approach of Convention over configuration.
- Maven reads a file which is `pom.xml` where the following are defined
    - project metadata
    - dependencies

- Refer Here
- Maven projects have goals
    - compile => Compile java sources and generate class files in target folder
    - test => Execute tests and generate test report
    - package => Create a packaging format (jar, war) after generating classes and test execution
    - install => The package created in the target folder will be copied into .m2 folder for other projects in your system to reuse the code
    - deploy => The package created will be copied into configured remote repository for other users to reuse.
    - clean => remove target folder.

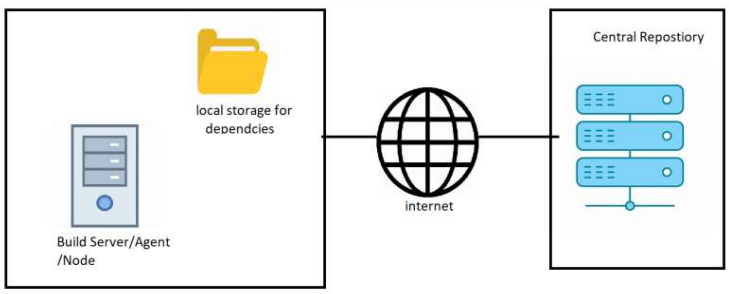
- With this we can configure

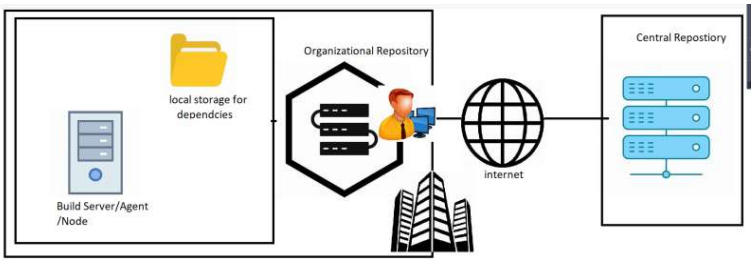
    - archive the artifact
    - Generate Test Result Report.

- Dependencies:

    - Java (Maven) => Refer Here
    - .net (Nuget)
    - Python (pip)
    - javascript (npm)

- Package Repository:
    - local
    - central





- Unit Tests:
- Automated Tests using Test Automation Frameworks
    - Selenium
    - Postman
    - RPA
    - Jmeter
- To change maven's behavior i.e. where to download dependencies from and where to deploy dependencies to, http proxy etc create a file called as settings.xml in .m2 folder Refer Here
- The tool configurations can be done from jenkins as well.

Exercises

- Learn about Environment Variables, Local Variables in Linux and Windows (Command Prompt/Powershell)
- Learn about YAML, JSON Refer Here
- Create a Linux vm (free instance)
    - Install python, pip
    - Install nodejs, npm
    - Install dotnet core

Leave a Reply

Enter your comment here...

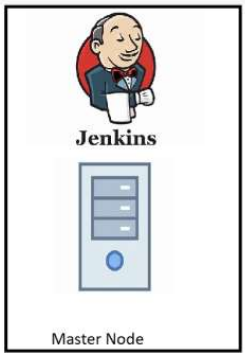
MENU ☰

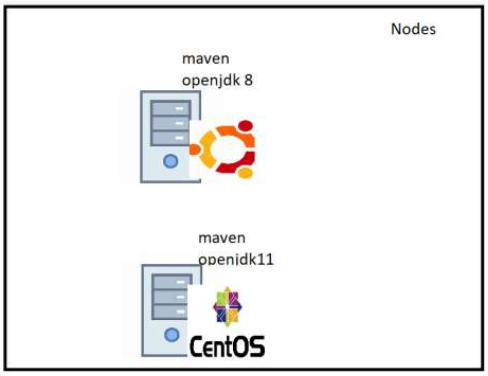
SEPTEMBER 21, 2022

DevOps Classroomnotes 21/Sep/2022

Exercise

- Create the below Jenkins Distributed Setup





- Create two jenkins jobs free style
  - openjdk 11 => spring pet clinic
  - openjdk 8 => game of life
- Steps, Clone the repo and execute

```
mvn package
```

MENU ☰

SEPTEMBER 22, 2022

DevOps Classroomnotes 22/Sep/2022

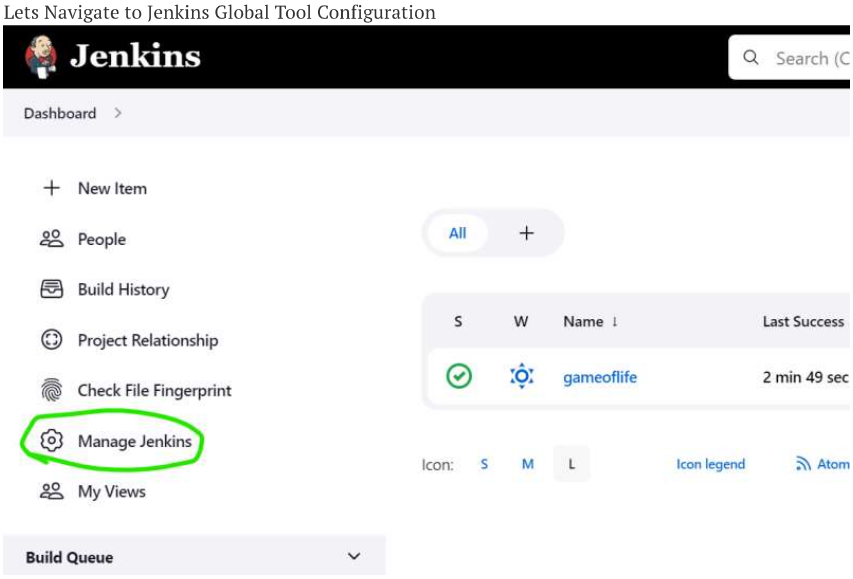
Exercise

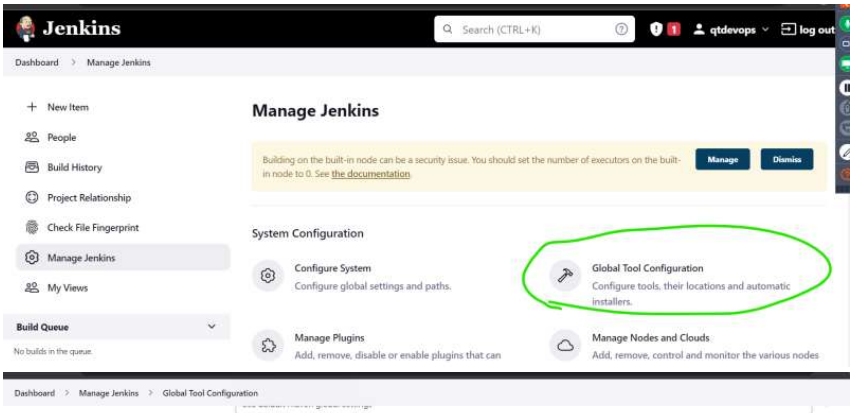
- Create the below Jenkins Distributed Setup
  
- Create two jenkins jobs free style
  - openjdk 11 => spring pet clinic
  - openjdk 8 => game of life
- Steps, Clone the repo and execute

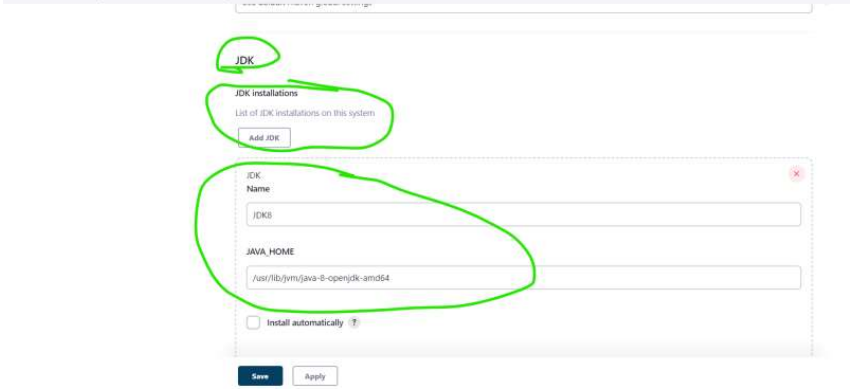
```
mvn package
```

Global Tool Configurations

- To make the management of multiple tools or different versions of the same tools Jenkins has Global Tool Configuration
- Lets Navigate to Jenkins Global Tool Configuration



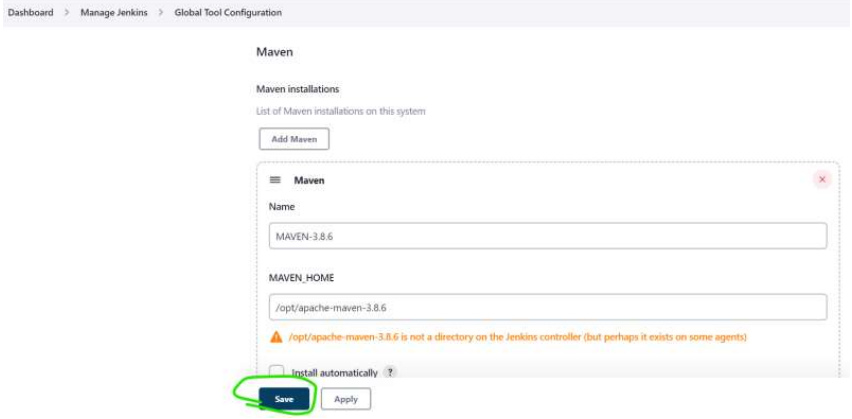




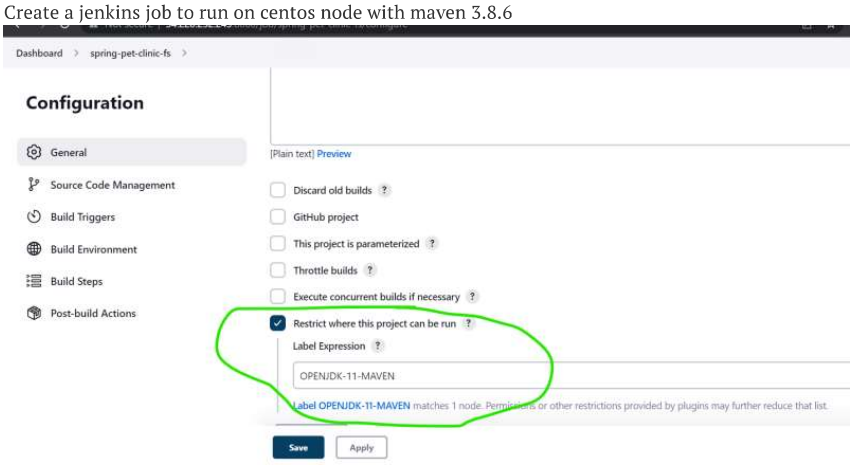
- Now lets install Latest version of Maven on the centos node Refer Here for the installation steps.
- Steps followed

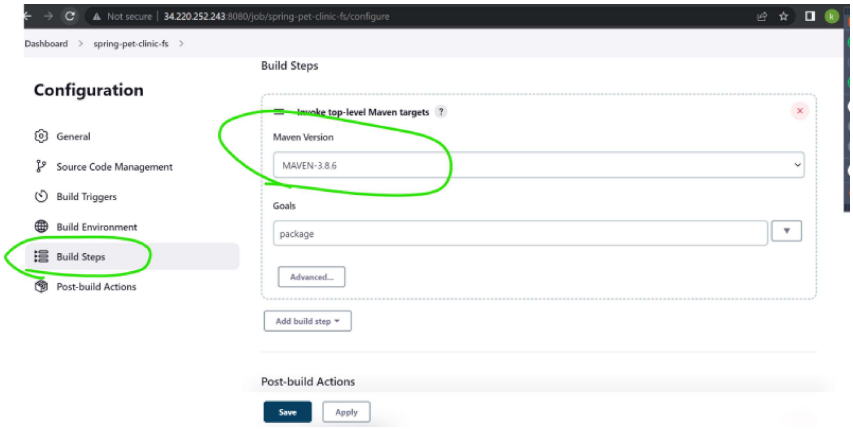
```
sudo yum install wget
cd /tmp
wget https://dlcdn.apache.org/maven/maven-3/3.8.6/binaries/apache-maven-3.8.6-bin.tar.gz
tar -xvzf apache-maven-3.8.6-bin.tar.gz
sudo cp -r apache-maven-3.8.6 /opt
```

- Now add maven location in the Global tools

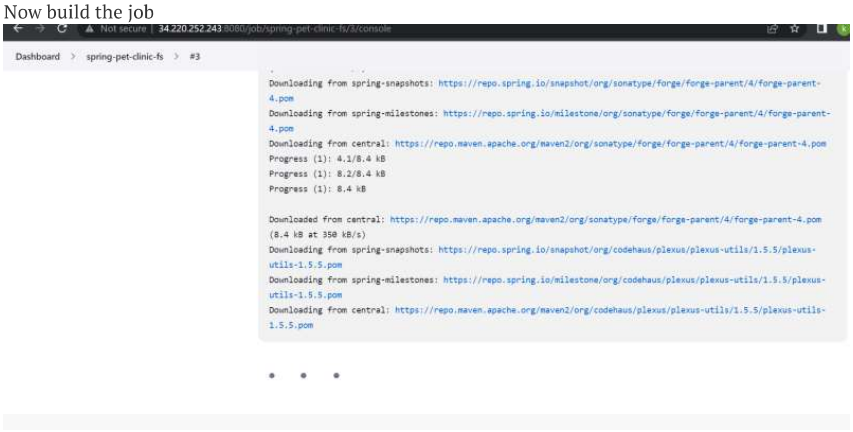


- Create a jenkins job to run on centos node with maven 3.8.6





- Now build the job



-

MENU ☰

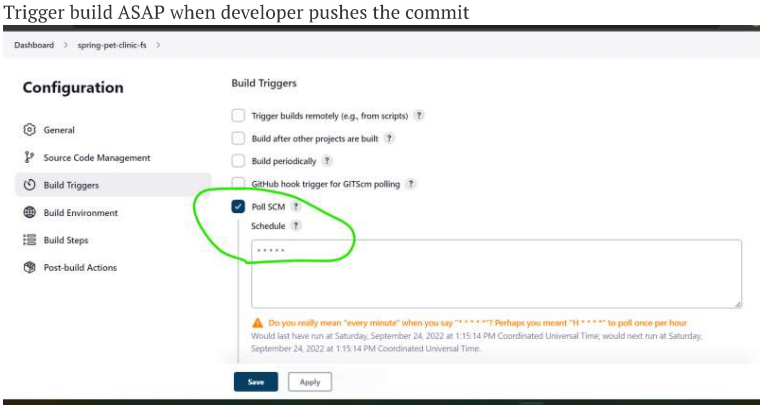
SEPTEMBER 24, 2022

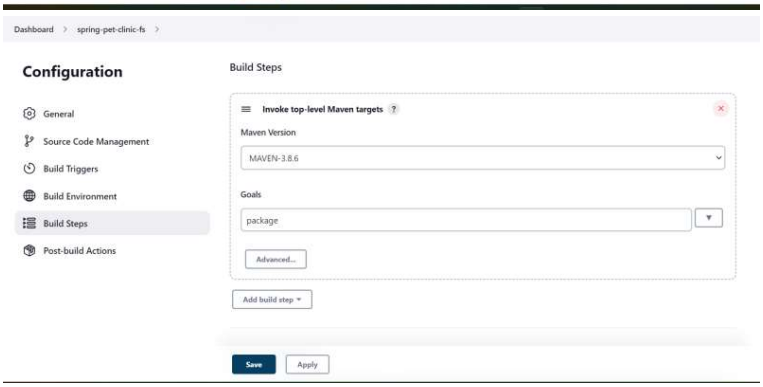
DevOps Classroomnotes 24/Sep/2022

Jenkins Contd

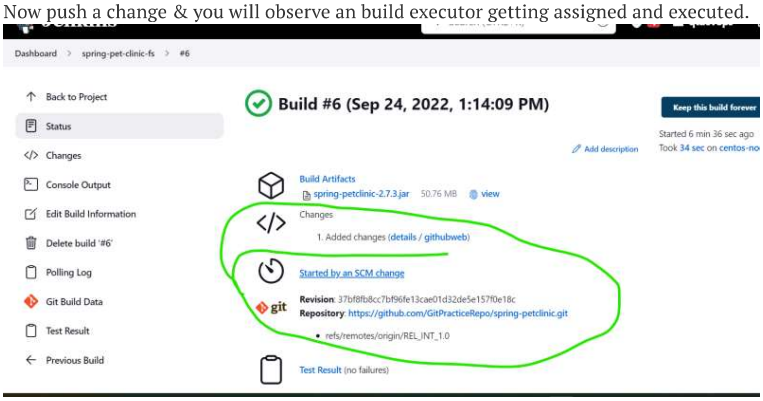
Triggering Jenkins Job on every new commit

- In Build Triggers, we have an option to poll scm where jenkins watches your source code repository (git) for the new changes i.e. commit in git.
- While using poll scm we use CRON Syntax to configure how frequently jenkins will poll SCM for changes
- Scenario 1: Build time is less
    - Trigger build ASAP when developer pushes the commit





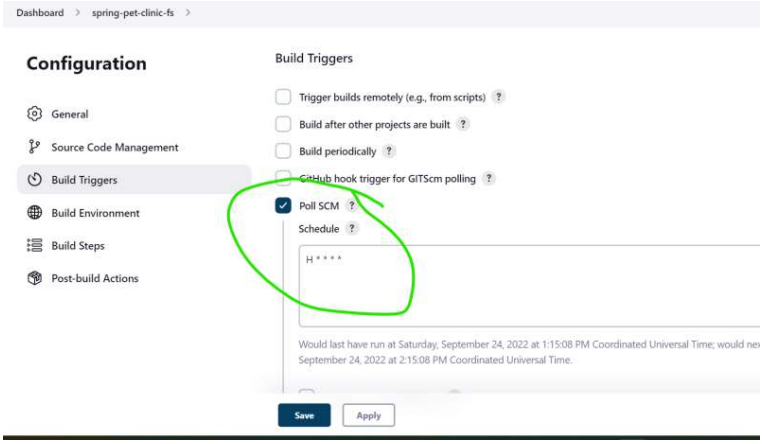
    - Now push a change & you will observe an build executor getting assigned and executed.



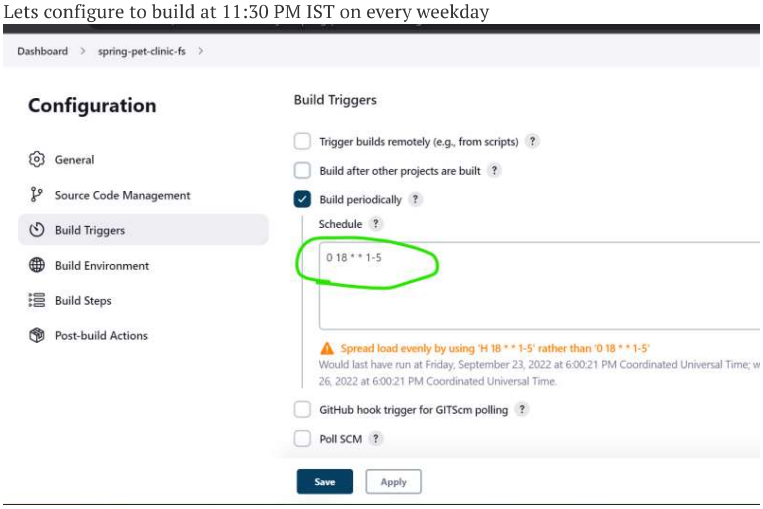
    - This is quite a common configuration for day build in projects where we have build times less than 10-15 minutes.

- Scenario 2: Build time is more (50 minutes)

- In this case we will configure Day build to run every n minutes or hours, in the above case every hour.
- Lets configure Poll SCM to poll git for changes every hour

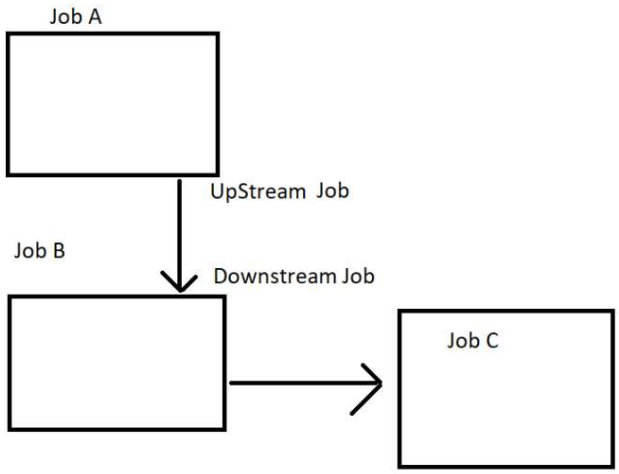


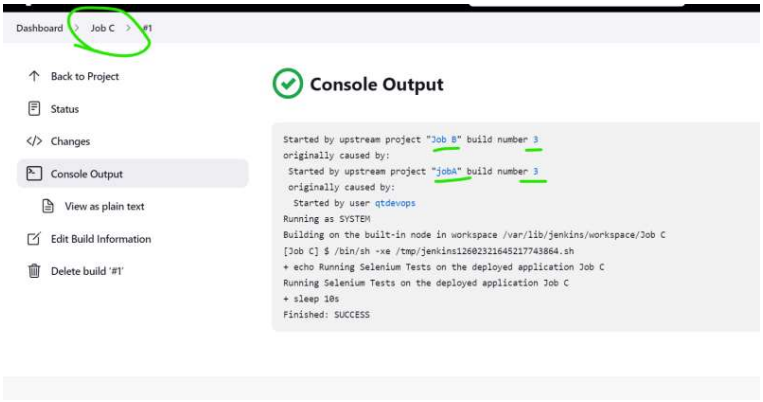
- Scenario 3: Build for the release to testing team next day
  - Here we need to build the code, deploy the package created on some servers.
  - Probably run some automated tests.
  - This is reffered as nightly build
  - This is generally configured to run at a particular time on every working day.
  - Lets configure to build at 11:30 PM IST on every weekday



Upstream and Downstream Jobs

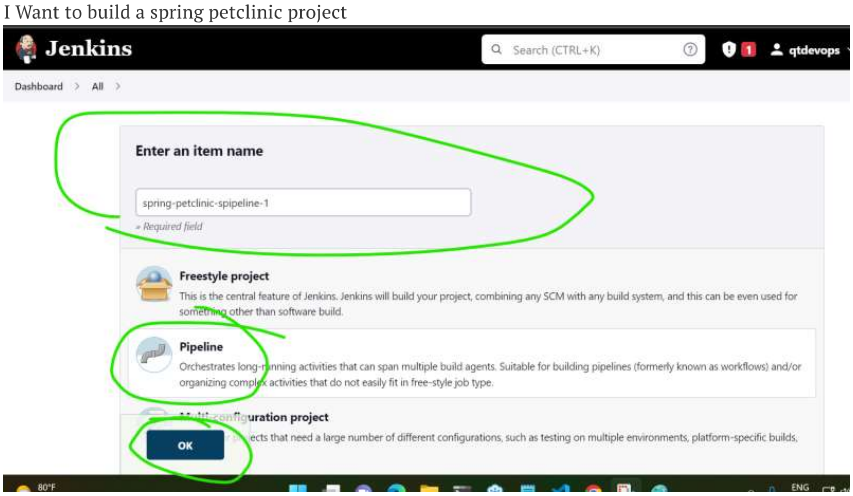
- We can link/chain multiple jobs by using
  - PostBuild Actions => Build other Project
  - Build Triggers => Build after Job



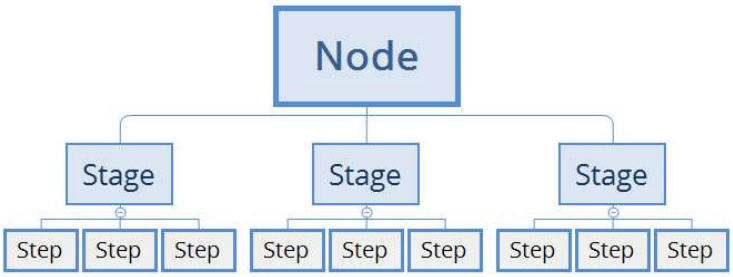


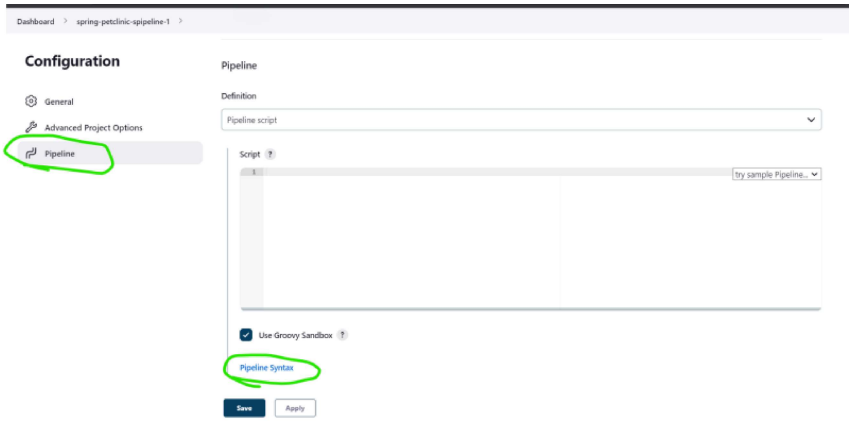
Scripted Pipeline With Free Style References

- I Want to build a spring petclinic project

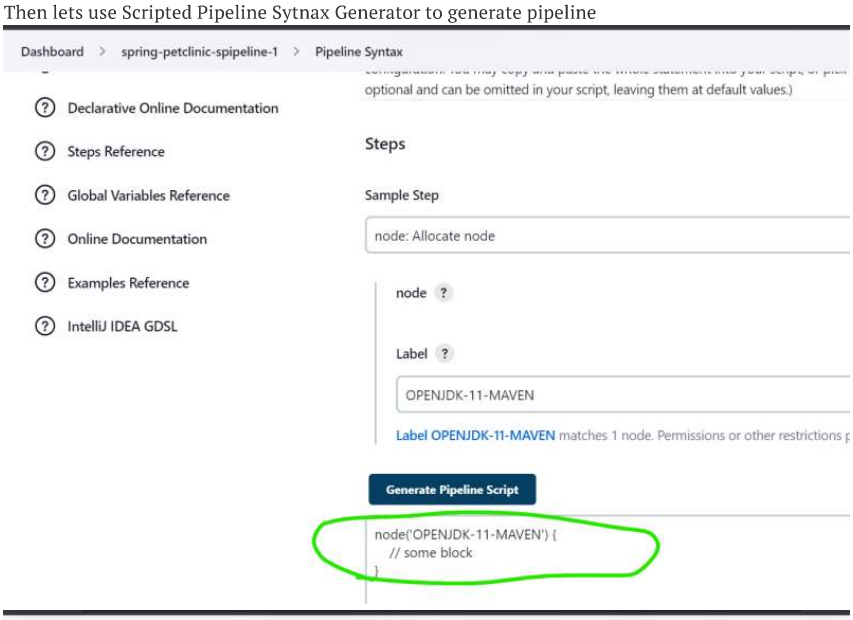


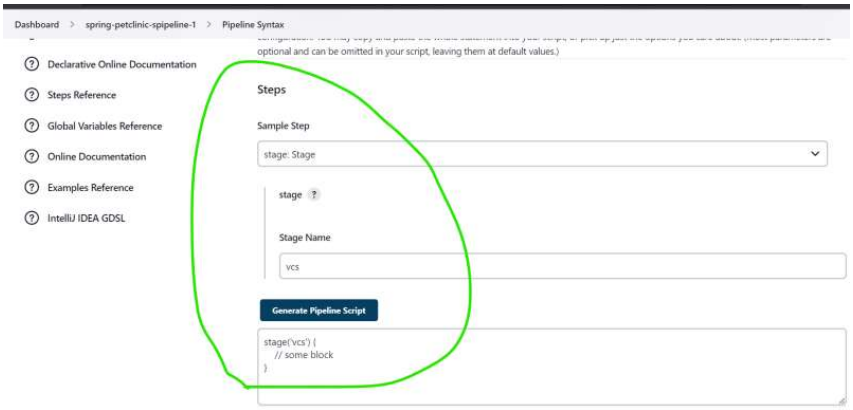
- In scripted pipeline, we need to select node and then stage and then step





- Then lets use Scripted Pipeline Sytnax Generator to generate pipeline



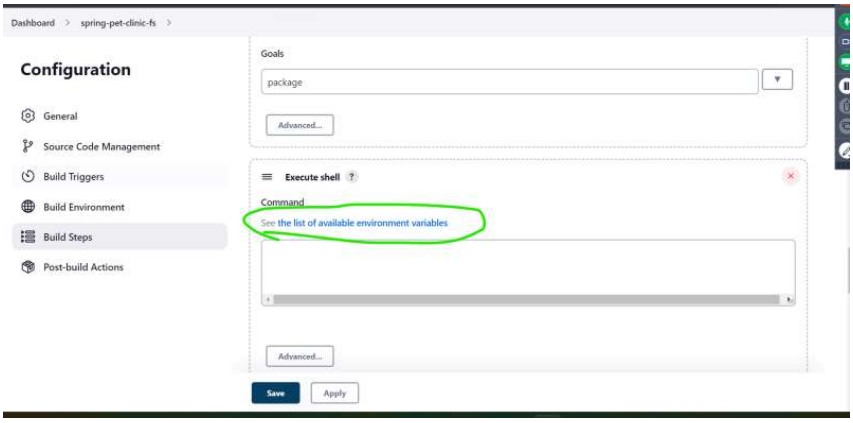


- Ideally the Generated script should be part of the source code and every change can be tracked as it is part of version control system.
- When you define your steps in pipeline in any file which is part of version control we refer it as pipeline as Code.
- The Scripted Pipeline which we generated in the class

```
node('OPENJDK-11-MAVEN') {
    stage('vcs') {
        git branch: 'REL_INT_1.0', url: 'https://github.com/GitPracticeRepo/spring-petcli
    }
    stage("build") {
        sh '/opt/apache-maven-3.8.6/bin/mvn package'
    }
    stage("archive results") {
        junit '**/surefire-reports/*.xml'

    }
}
```

Exercise:

- Find out the different environment variables



Leave a Reply

Enter your comment here...

About continuous learner

devops & cloud enthusiastic learner

VIEW ALL POSTS

MENU ☰

SEPTEMBER 25, 2022

DevOps Classroomnotes 25/Sep/2022

Jenkins 2.0

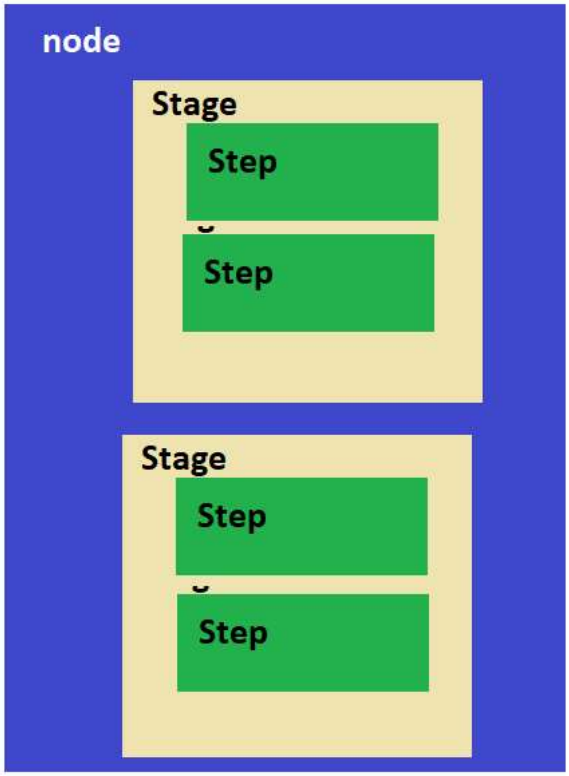
- Jenkins 2.0 Was a release to include pipeline as a code.
- Earlier Jenkins released Scripted Pipelines and then Declarative Pipelines.
- Scripted Pipeline:
    - This is groovy language which we use in the scripted pipeline
    - This is extreemly customizable
- Declarative Pipeline:
    - This has jenkins DSL (Domain Specific Language) which is developed in groovy
    - This is optimized around most of the ci/cd pipelines.

Groovy

- Groovy is a Java Based Language Refer Here
- Groovy is popular for simplicity and used for creating Domain Specific Languages and scripting purposes.

Scripted Pipelines

- The Structure of the Pipeline is as follows



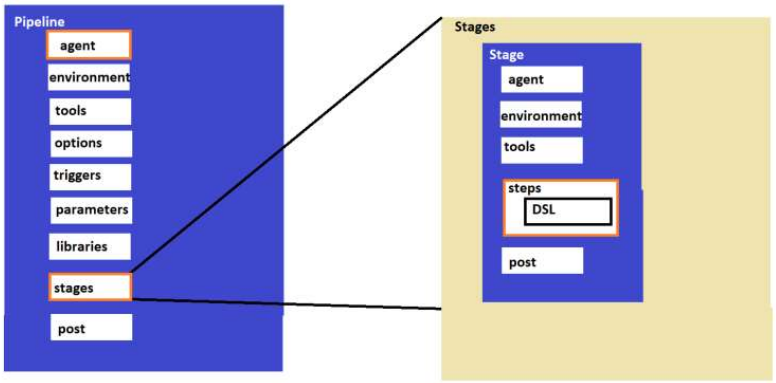
- The Sample Jenkinsfile created has the following content

```
node {
    stage('test') {
        sh 'echo hello'
    }
}
```

- Refer Here for the steps
- Refer Here for the changes done which was written based on a step called as git Refer Here

Declarative Pipelines

- These were created for the users who are familiar with jenkins to create pipelines without too much of learning curve.
- Refer Here
- The syntax overview



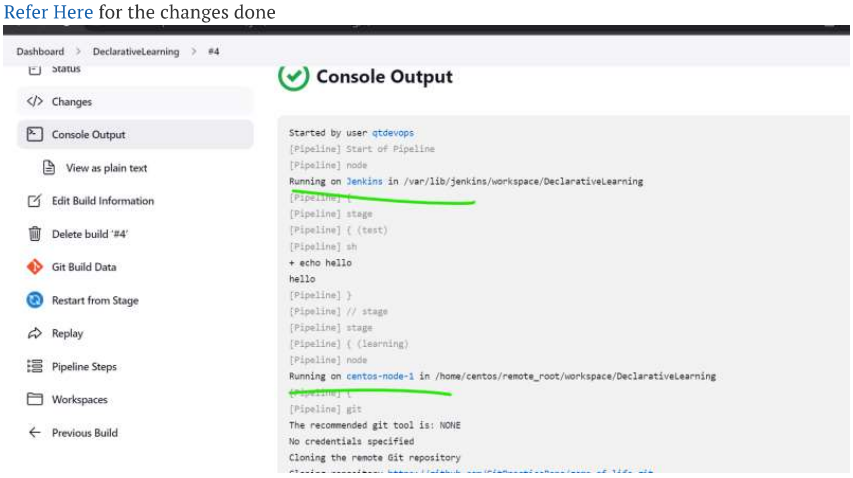
- In the Scripted Pipeline we have used the following

```
node {
    stage('test') {
        sh 'echo hello'
    }
    stage('learning') {
        git url: 'https://github.com/GitPracticeRepo/game-of-life.git',
            branch: 'master'
    }
}
```

- Now lets try doing this in Declarative Refer Here for the changeset containing the declarative pipeline

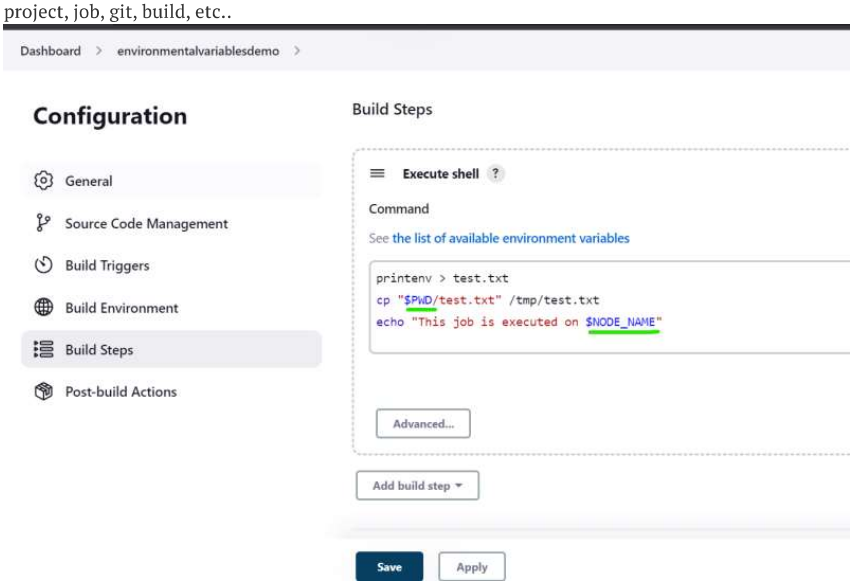
```
pipeline {
    agent any
    stages {
        stage('test') {
            steps {
                sh 'echo hello'
            }
        }
        stage('learning') {
            steps {
                git url: 'https://github.com/GitPracticeRepo/game-of-life.git',
                    branch: 'master'
            }
        }
    }
}
```

- Lets run stage test on any node and stage learning on a node with some label `OPENJDK-11-MAVEN`. Refer Here for the changes done



Environmental Variables in Jenkins

- On the node we will have some Environmental variables set by Operating system. In addition to that, Jenkins adds some environment variables which will contain information about the Jenkins project, job, git, build, etc..



Exercise

- Create a Jenkins JOb to build spring petclinic using maven
  - free style project
  - scripted pipeline
  - declarative pipeline

Leave a Reply

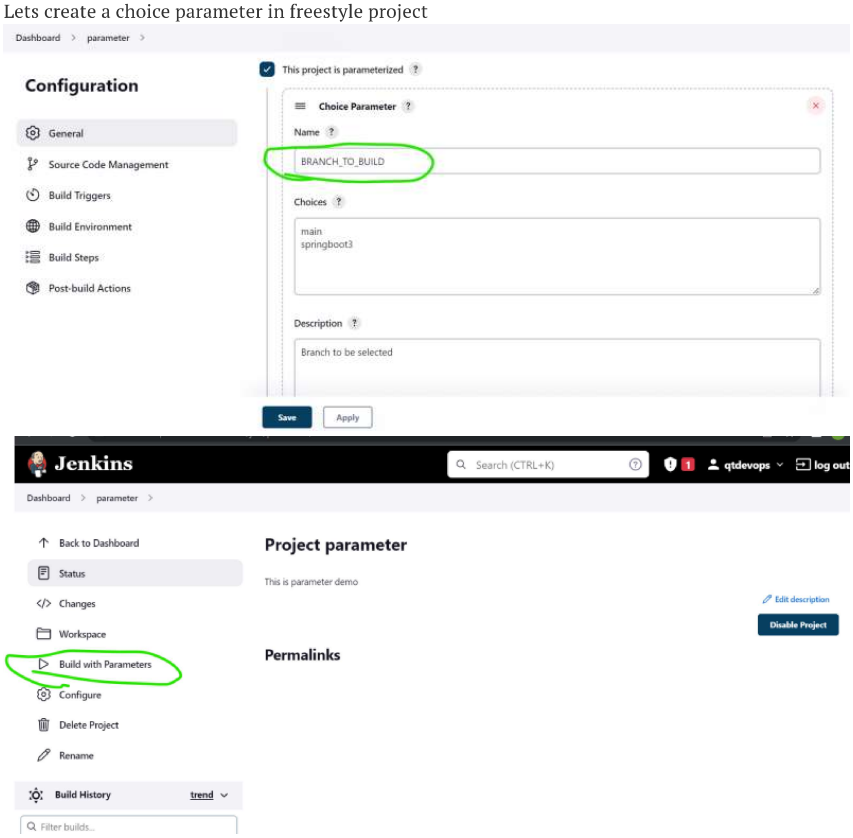
MENU ☰

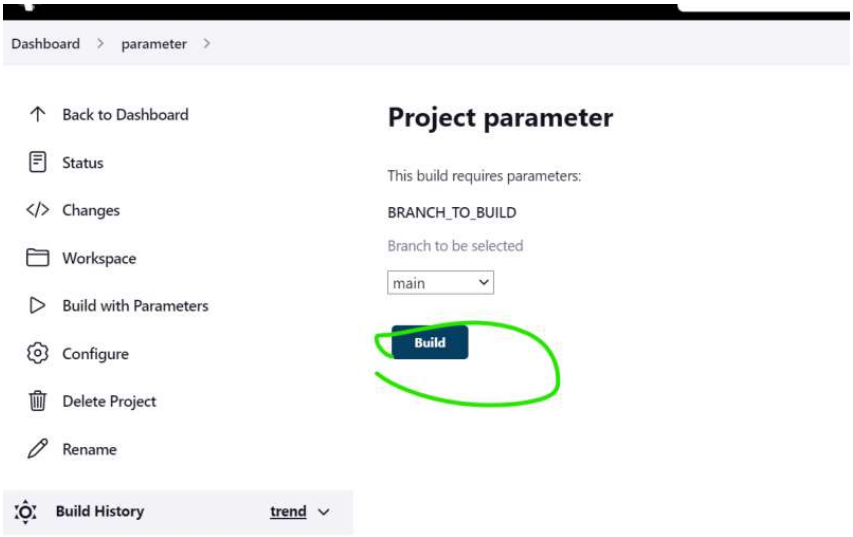
SEPTEMBER 27, 2022

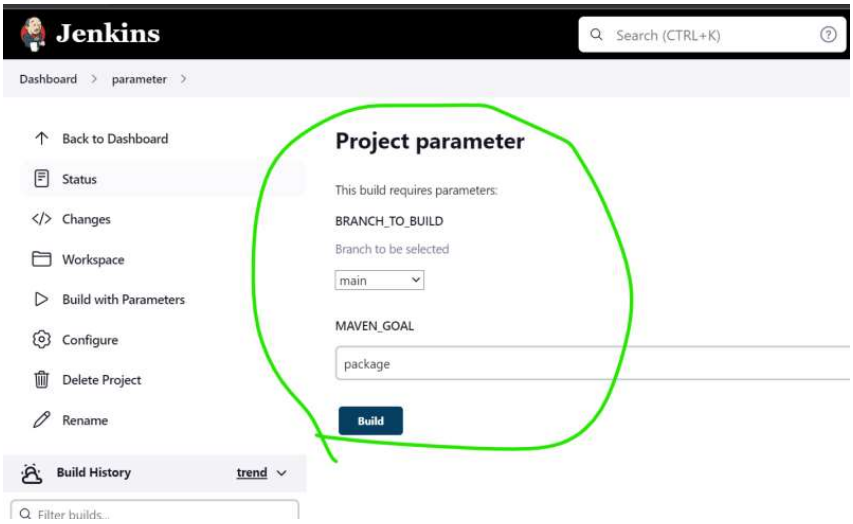
DevOps Classroomnotes 27/Sep/2022

Jenkins Parameters

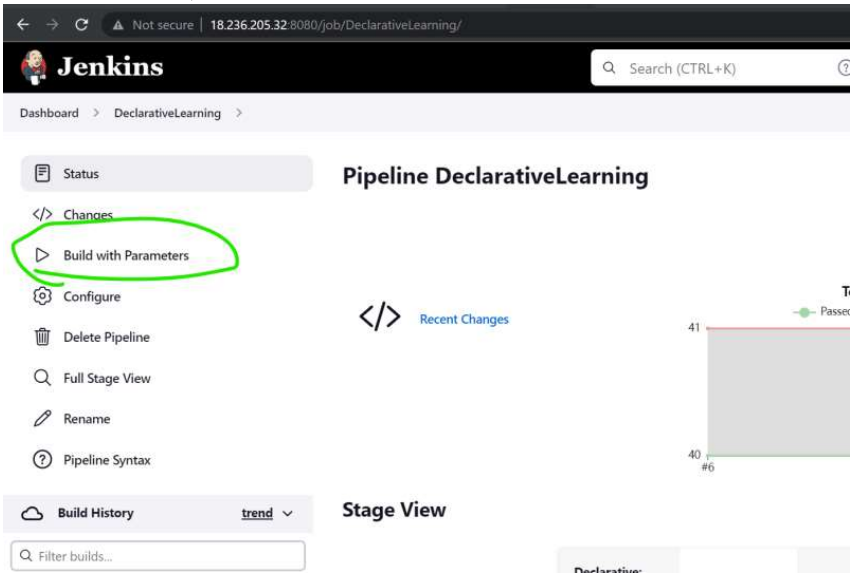
- Jenkins Parameters helps in getting additional information during the build. User can pass options
- Lets create a choice parameter in freestyle project

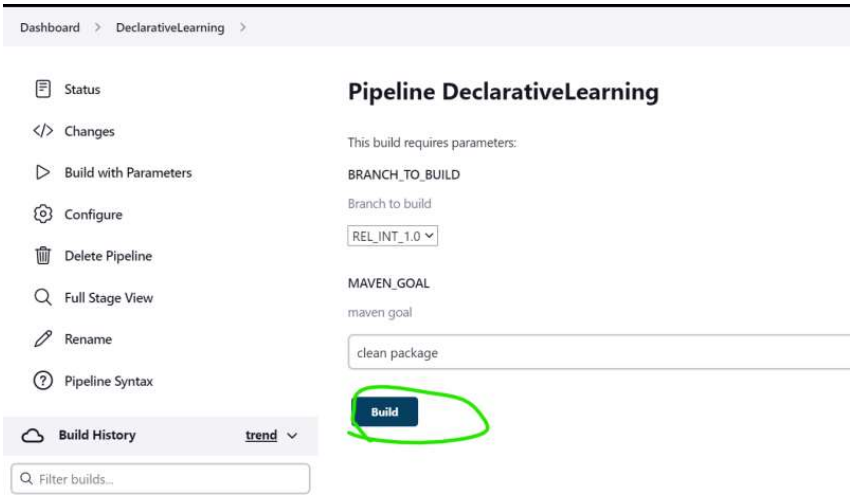






- Lets acheive the same in the Declarative Pipeline
- Refer Here for the declarative pipeline
- Refer Here for the parameter documentation
- Refer Here for the changes done to include parameters
- Now lets check in the Jenkins UI





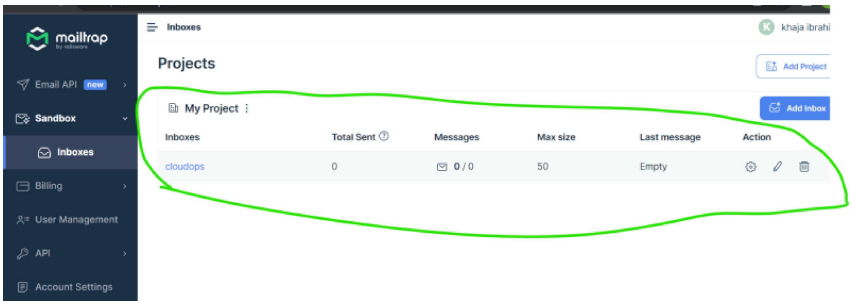
Triggers in Declarative Pipeline

- Lets add the day build trigger i.e. POLL SCM Every minute Refer Here for the changes
- Please refer classroom video for demonstration purposes

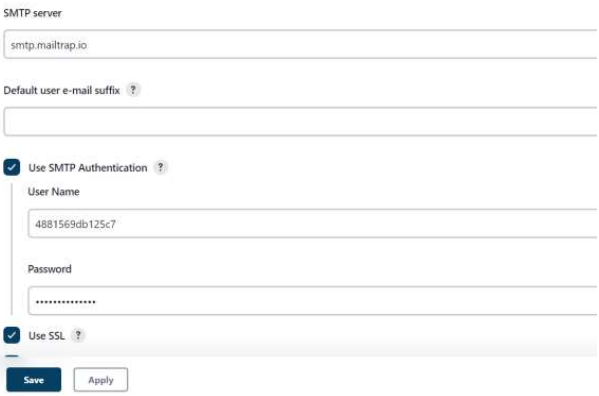
Notifications

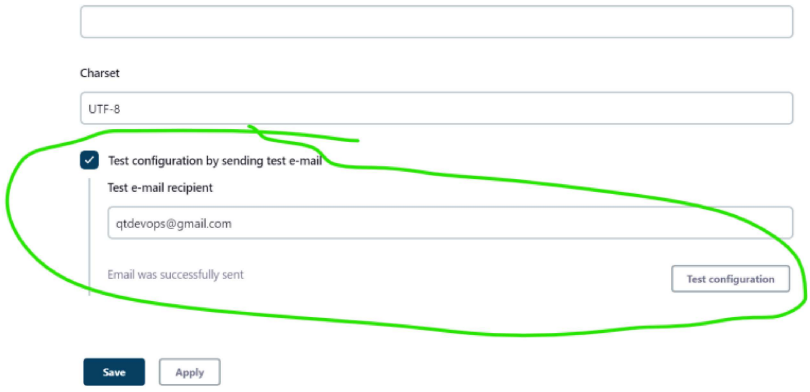
Email Notifications

- For email configurations, we have SMTP servers which we need to configure on the Jenkins
- Create an account in mail trap Refer Here
- Create one inbox

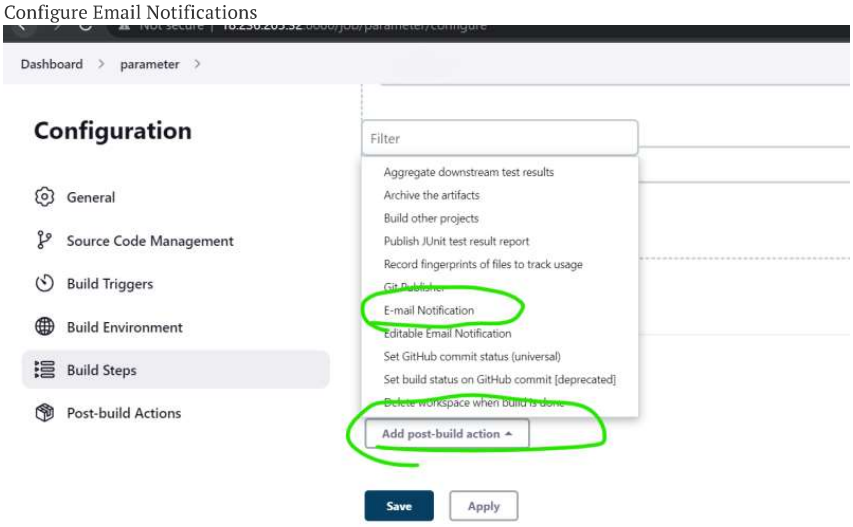


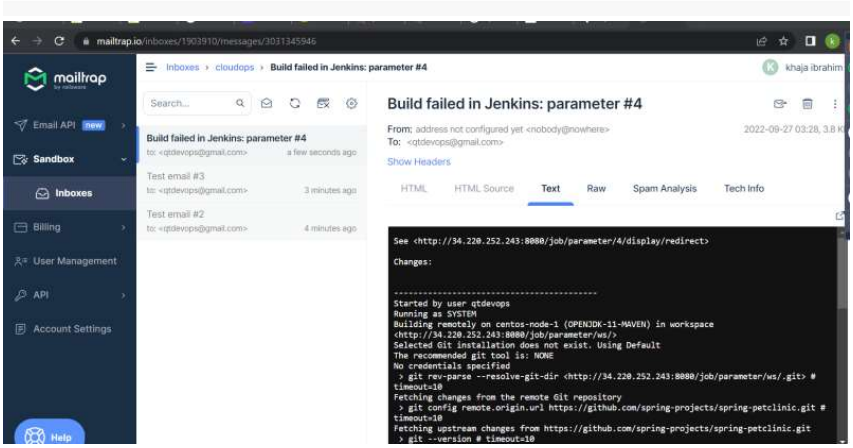
- Get the credentials from the mail trap
- Now Navigate to Manage Jenkins => Configure System => Email Notification and configure SMTP





- Configure Email Notifications





Link to upload Activity Completions

- Refer Here to submit the activity to build the project via scripted, declarative pipeline

MENU ☰

SEPTEMBER 28, 2022

DevOps Classroomnotes 28/Sep/2022

Activity

- Create a Jenkins Free Style Project and then Declartive Pipeline.
- Ensure you have a node or master with ubuntu
- Install node js version 16 and npm on the master/node
- clone the code https://github.com/GitPracticeRepo/js-e2e-express-server.git
- Execute the following commands

```
npm install
npm run build
```


Leave a Reply

Enter your comment here...

This site uses Akismet to reduce spam. Learn how your comment data is processed.

MENU ☰

SEPTEMBER 29, 2022

DevOps Classroomnotes 29/Sep/2022

Notifications Contd…

- For sending mail we have pipeline step Refer Here
- Refer Here for the changes done
- Refer Here for the changes which include tghe Environmental variables
- Mail Sending in the case of scripted pipeline

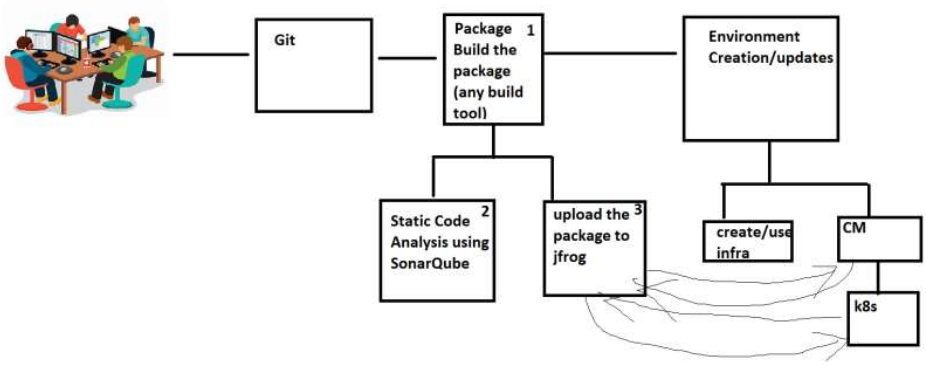
```
node {
    try {
        stage('test') {
            sh 'echo hello'
        }
        stage('learning') {
            git url: 'https://github.com/GitPracticeRepo/game-of-life.git',
                branch: 'master'
        }
    }
    catch(err) {
        mail subject: "Build Failed for Jenkins JOB $env.JOB_NAME with Build ID $env.B
             body: "Build Failed for Jenkins JOB $env.JOB_NAME",
             to: 'qtdevops@gmail.com'
    }
```

Static Code Analysis And Quality Gates

- For every commit it is difficult & time consuming to review the code manually on application features, language best practices, code coverage (unit test imporovements) etc..
- Static Code Analysis tools can scan the code for
    - Language Best Practices
    - Code Coverage
    - Design issues
    - Security Issues (SAST, SCA)
- Sonarqube is the tool which can perform static code analysis.
- If you want your ci/cd pipeline to consider build as failure if the report from static code analysis is not ok/ not meeting your organization criteria, then we need to implement Quality Gate.

Package Repository or Artifact Repository

- During CI/CD we build packages. If we want to store these packages in some repository so that we can deploy any version on our Dev, QA, UAT, PROD environments
- There are lot of options here, But JFrog/Artifactory is widely adopted Refer Here
- The other options are
    - Nexus
    - GitHub Packages
    - Azure Artifacts (Azure DevOps)



Exercise

- Create a free account for jfrog cloud Refer Here
- Create a free account for docker hub Refer Here
- Create a free account for sonar cloud Refer Here

Leave a Reply

Enter your comment here...

About continuous learner

devops & cloud enthusiastic learner

MENU ☰

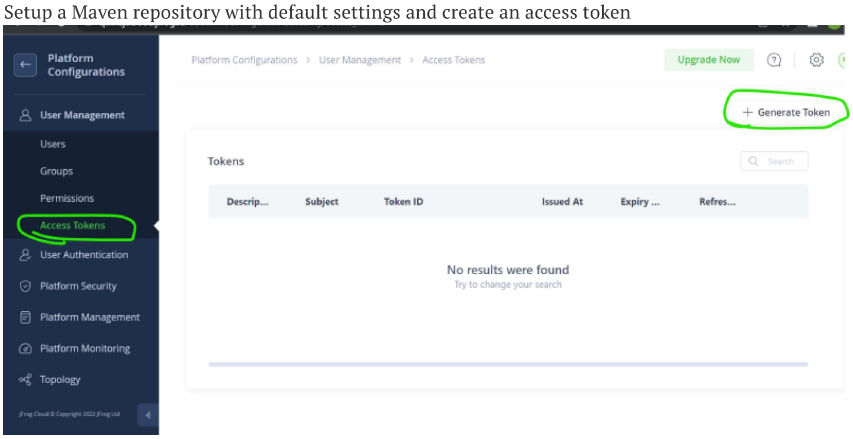
SEPTEMBER 30, 2022

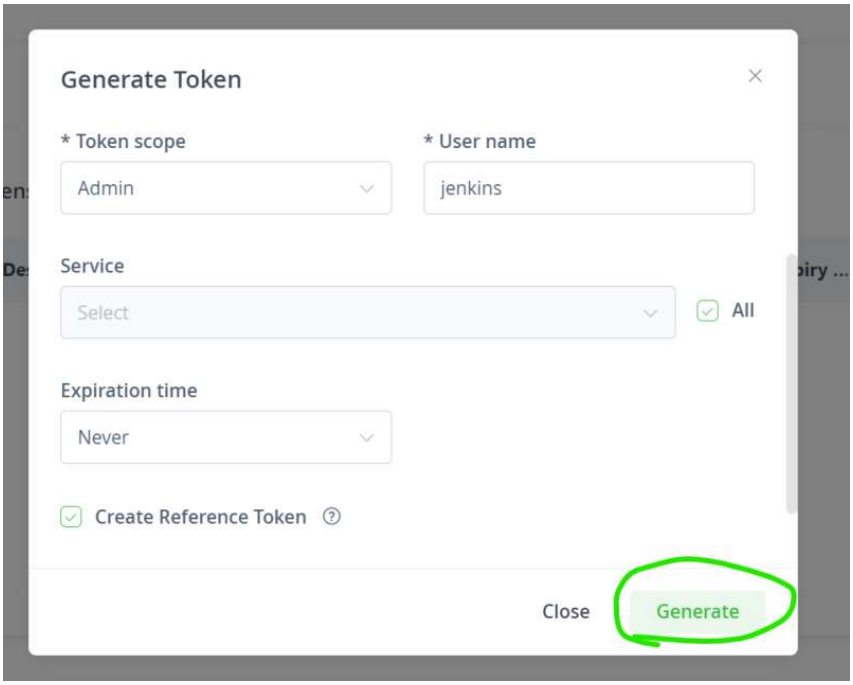
DevOps Classroomnotes 30/Sep/2022

Jenkins Integration With SonarQube

Jenkins Integration with JFrog/Artifactory

- Create a free account which lasts for 14 days
- Setup a Maven repository with default settings and create an access token





- For integration of Jenkins and artifactory Refer Here
- The artifactory plugin installed is expecting JDK 8 and mvn deploy with manually configuring settings.xml is failed while copying jars from releases

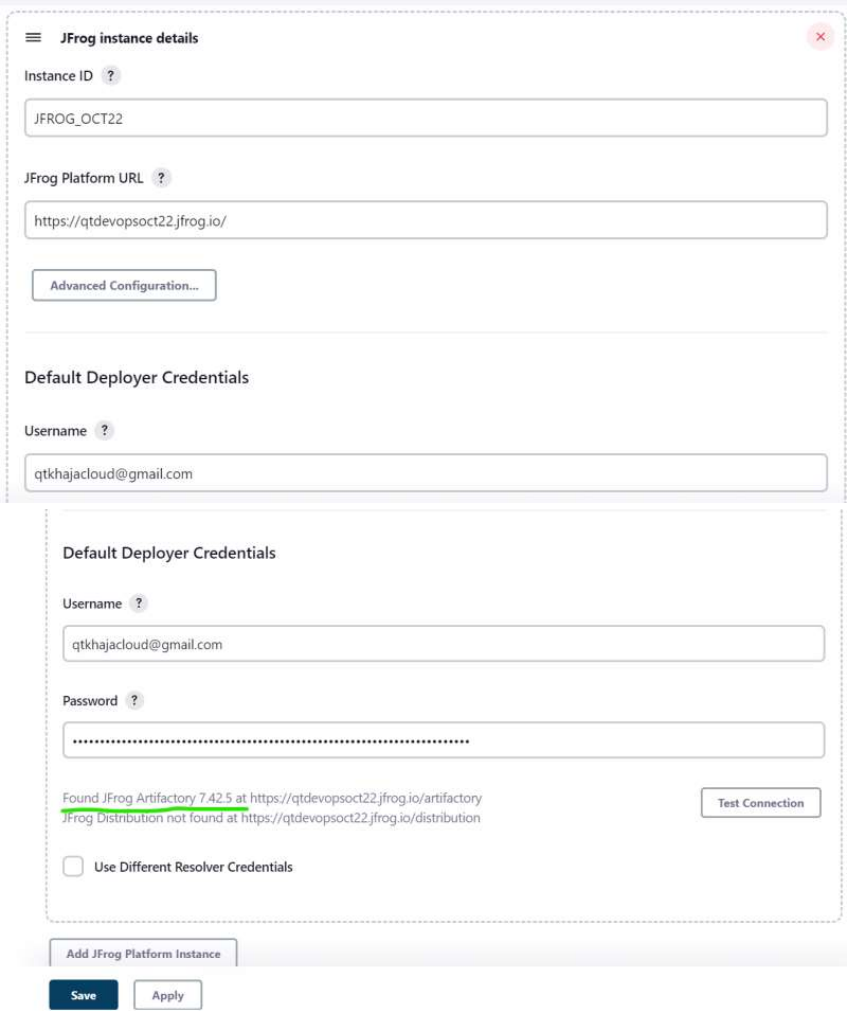
MENU ☰

OCTOBER 1, 2022

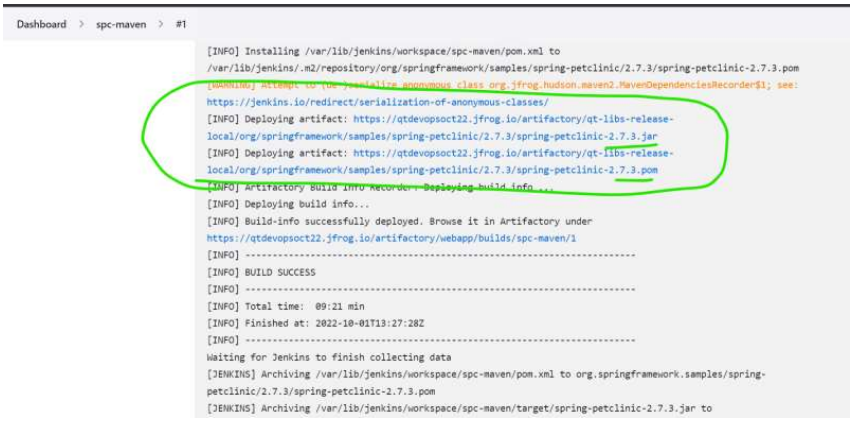
DevOps Classroomnotes 01/Oct/2022

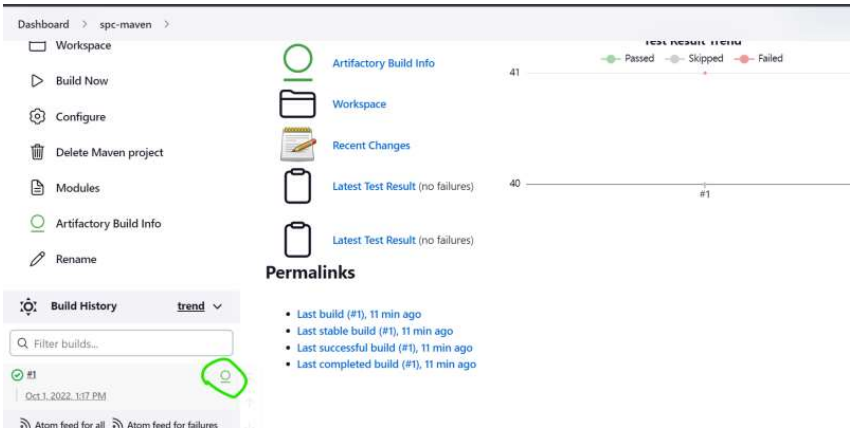
Artifactory/JFROG Configuration

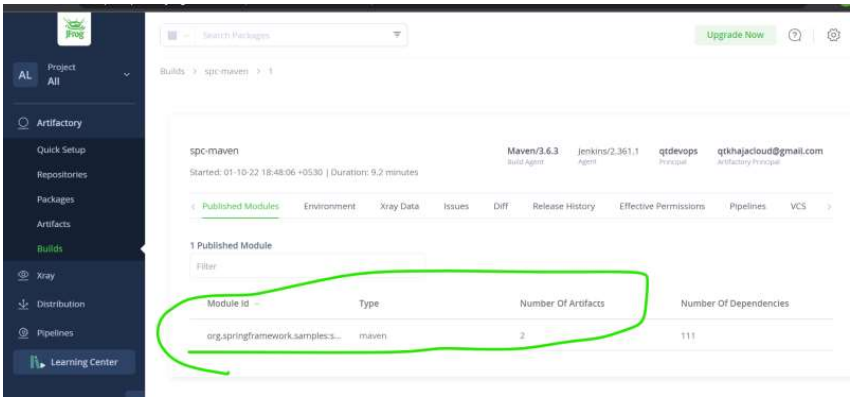
- Configuring Artifactory Settings in Jenkins
- Navigate to Configure Sytem => JFROG

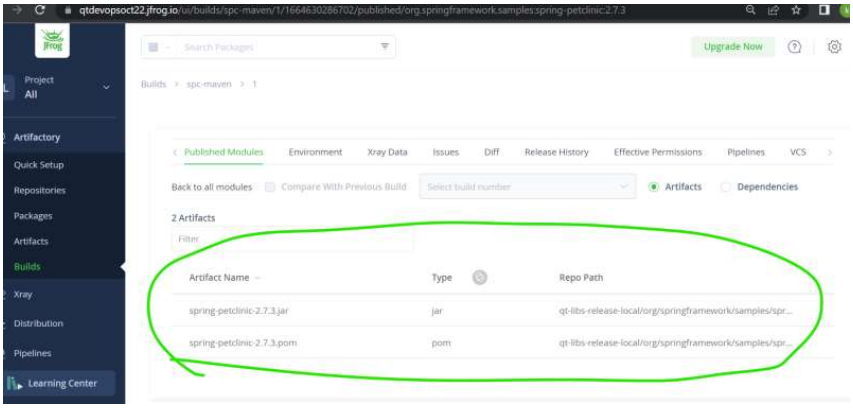


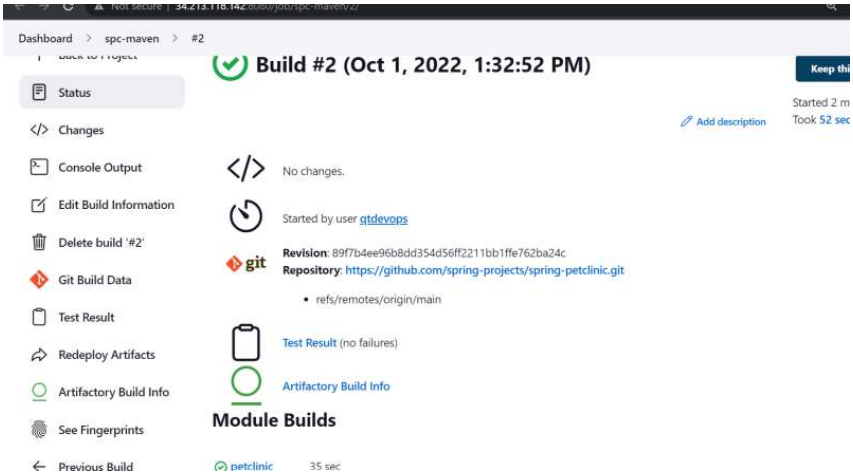
- Now Lets create a maven project for spring petclinic. For configuration refer previous classroom screenshots


Dashboard > spc-maven > #1
[INFO] Installing /var/lib/jenkins/workspace/spc-maven/pom.xml to
/var/lib/jenkins/.m2/repository/org/springframework/samples/spring-petclinic/2.7.3/spring-petclinic-2.7.3.pom
[WARNING] Attempt to (de-)serialize anonymous class org.jfrog.hudson.maven2.MavenDependenciesRecorder$1; see:
https://jenkins.io/redirect/serialization-of-anonymous-classes/
[INFO] Deploying artifact: https://qtdevopsoct22.jfrog.io/artifactory/qt-libs-release-
local/org/springframework/samples/spring-petclinic/2.7.3/spring-petclinic-2.7.3.jar
[INFO] Deploying artifact: https://qtdevopsoct22.jfrog.io/artifactory/qt-libs-release-
local/org/springframework/samples/spring-petclinic/2.7.3/spring-petclinic-2.7.3.pom
[INFO] Artifactory Build Info Recorder: Deploying build info ...
[INFO] Deploying build info...
[INFO] Build-info successfully deployed. Browse it in Artifactory under
https://qtdevopsoct22.jfrog.io/artifactory/webapp/builds/spc-maven/1
[INFO] --
[INFO] BUILD SUCCESS
[INFO] --
[INFO] Total time: 09:21 min
[INFO] Finished at: 2022-10-01T13:27:28Z
[INFO] --
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving /var/lib/jenkins/workspace/spc-maven/pom.xml to org.springframework.samples/spring-
petclinic/2.7.3/spring-petclinic-2.7.3.pom
[JENKINS] Archiving /var/lib/jenkins/workspace/spc-maven/target/spring-petclinic-2.7.3.jar to


Dashboard > spc-maven >
Workspace
Build Now
Configure
Delete Maven project
Modules
Artifactory Build Info
Rename
Build History trend
Filter builds...
#1
Oct 1, 2022, 1:17 PM
Atom feed for all
Atom feed for failures
Artifactory Build Info
Workspace
Recent Changes
Latest Test Result (no failures)
Latest Test Result (no failures)
Permalinks
Last build (#1), 11 min ago
Last stable build (#1), 11 min ago
Last successful build (#1), 11 min ago
Last completed build (#1), 11 min ago
Passed
Skipped
Failed
41
40
#1


Upgrade Now
Project
All
Artifactory
Quick Setup
Repositories
Packages
Artifacts
Builds
Xray
Distribution
Pipelines
Learning Center
spc-maven
Started: 01-10-22 18:48:06 +0530 | Duration: 9.2 minutes
Maven/3.6.3
jenkins/2.361.1
qtdevops
qtkhajacloud@gmail.com
Published Modules
Environment
Xray Data
Issues
Diff
Release History
Effective Permissions
Pipelines
VCS
1 Published Module
Filter
Module Id
Type
Number Of Artifacts
Number Of Dependencies
org.springframework.samples:s...
maven
2
111





- Declarative Pipeline for doing the same Refer Here
- Scripted Pipeline for jenkins Refer Here
- Refer Here for the repo maintained by jfrog to work with multiple technologies
- Refer Here for the artificatory specific steps that get added to jenkins after Artifactory plugin installation.
- Refer Here for the pipeline changeset
- Refer Here for the fix done, There was no need to define the artifactory server as it is already defined in System Settings.
- Refer Here for downloading and install artifactory/jfrog opensource.

Static Code Analysis

- Refer Here for the previous blog about sonar qube
- Create the token in the sonar cloud
- Refer Here for the pipeline steps
- To be continued in the next session

MENU ☰

OCTOBER 2, 2022

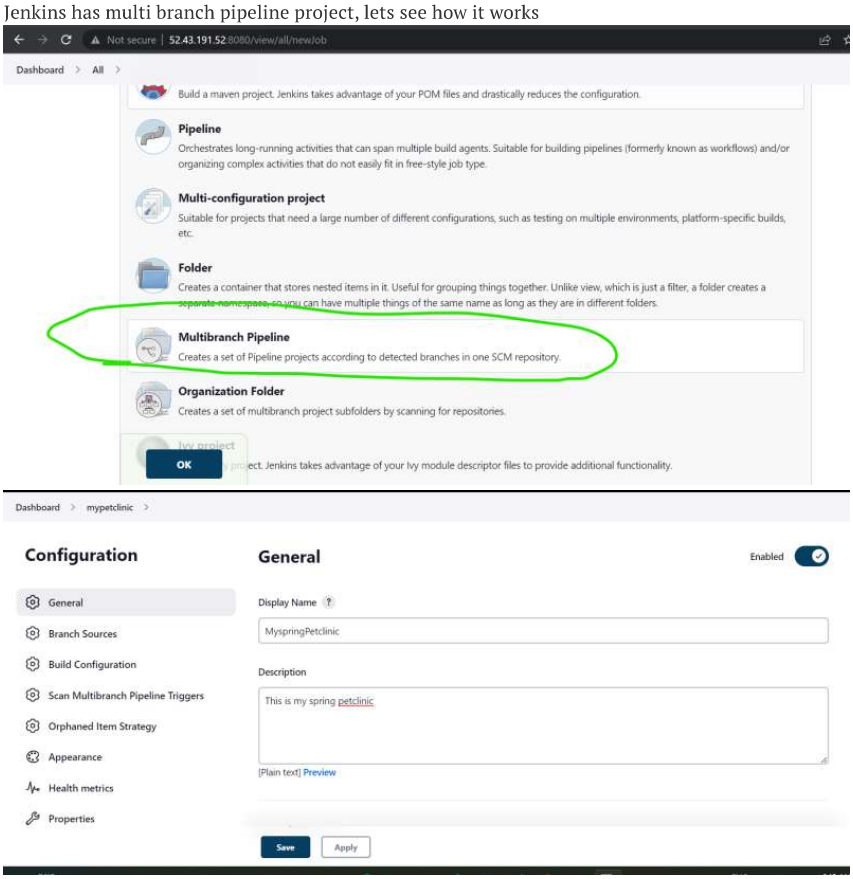
DevOps Classroomnotes 02/Oct/2022

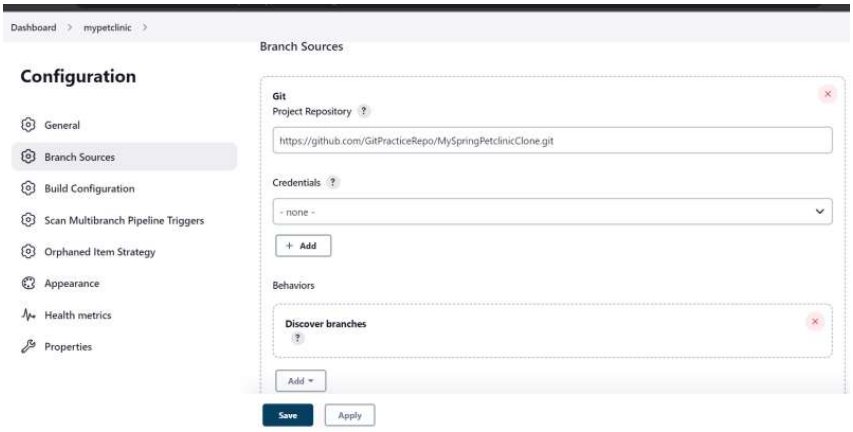
Sonarqube Contd

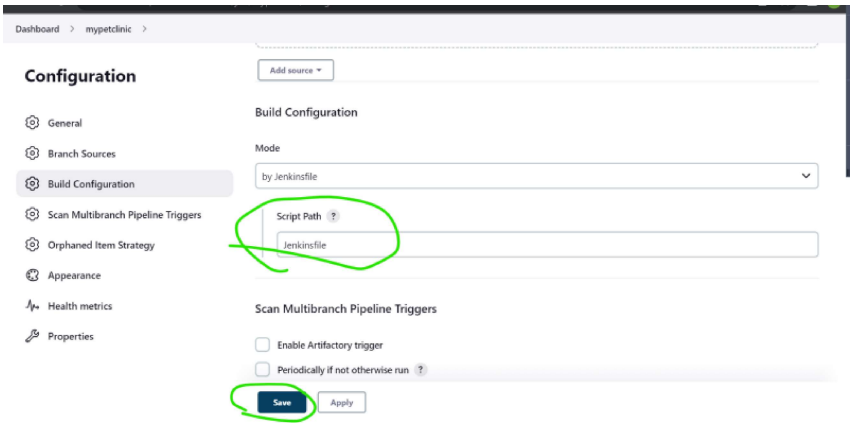
- Refer Here for jenkins integration with sonar qube
- Refer Here for jenkins integration with sonar cloud
- For the Jenkinsfile for the sonarqube project Refer Here

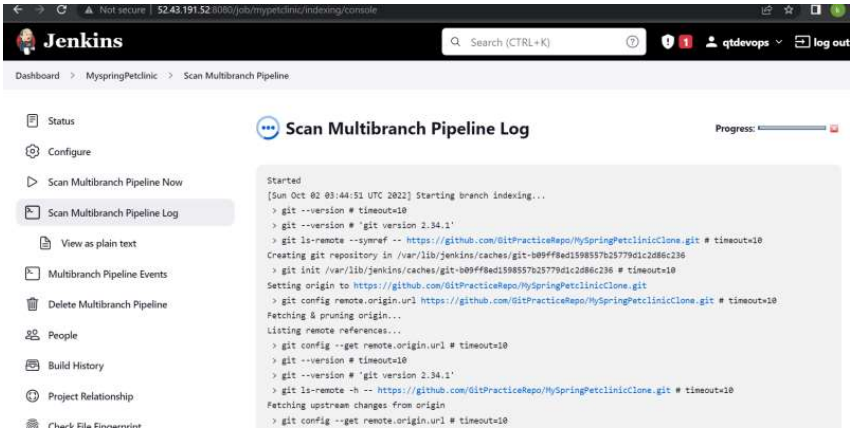
Multi-Branch Pipelines

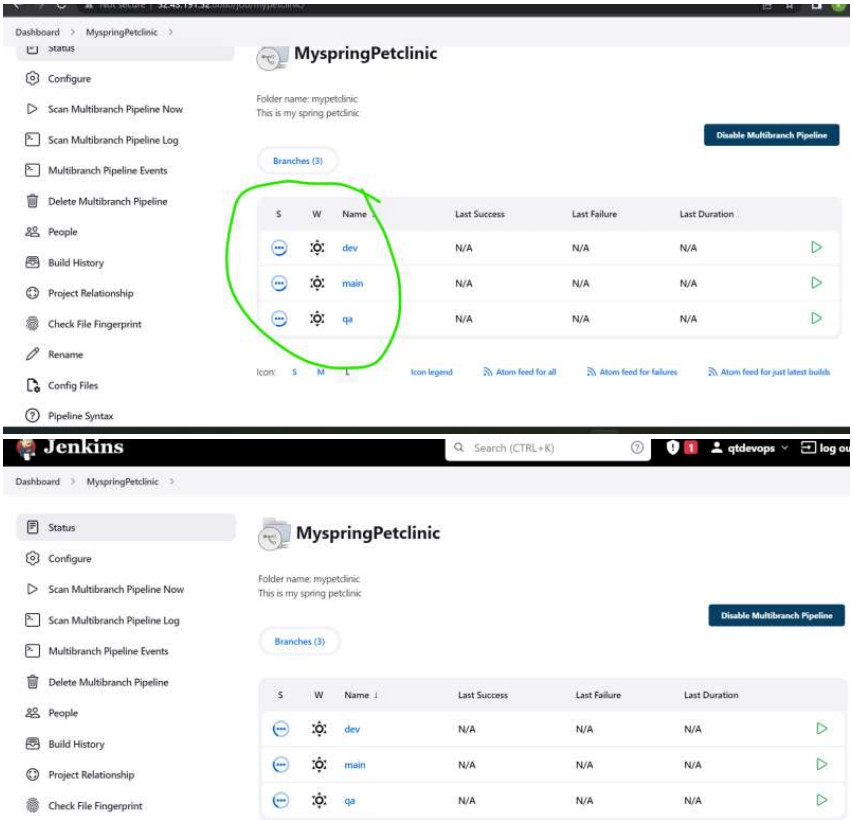
- Consider the following repo, where we have 3 branches which reflect 3 different environments with 3 different Jenkinsfile i.e pipeline steps
- Jenkins has multi branch pipeline project, lets see how it works




Dashboard > mypetclinic >
Configuration
General
Branch Sources
Build Configuration
Scan Multibranch Pipeline Triggers
Orphaned Item Strategy
Appearance
Health metrics
Properties
Branch Sources
Git
Project Repository
https://github.com/GitPracticeRepo/MySpringPetclinicClone.git
Credentials
- none -
+ Add
Behaviors
Discover branches
Add
Save
Apply


Dashboard > mypetclinic >
Configuration
General
Branch Sources
Build Configuration
Scan Multibranch Pipeline Triggers
Orphaned Item Strategy
Appearance
Health metrics
Properties
Add source
Build Configuration
Mode
by Jenkinsfile
Script Path
Jenkinsfile
Scan Multibranch Pipeline Triggers
Enable Artifactory trigger
Periodically if not otherwise run
Save
Apply


Not secure | 52.43.191.52:8080/job/mypetclinic/indexing/console
Jenkins
Search (CTRL+K)
qtdevops
log out
Dashboard > MyspringPetclinic > Scan Multibranch Pipeline
Status
Configure
Scan Multibranch Pipeline Now
Scan Multibranch Pipeline Log
View as plain text
Multibranch Pipeline Events
Delete Multibranch Pipeline
People
Build History
Project Relationship
Scan Multibranch Pipeline Log
Progress:
Started
[Sun Oct 02 03:44:51 UTC 2022] Starting branch indexing...
> git --version # timeout=10
> git --version # 'git version 2.34.1'
> git ls-remote --symref -- https://github.com/GitPracticeRepo/MySpringPetclinicClone.git # timeout=10
Creating git repository in /var/lib/jenkins/caches/git-b09ff8ed1598557b25779d1c2d86c236
> git init /var/lib/jenkins/caches/git-b09ff8ed1598557b25779d1c2d86c236 # timeout=10
Setting origin to https://github.com/GitPracticeRepo/MySpringPetclinicClone.git
> git config remote.origin.url https://github.com/GitPracticeRepo/MySpringPetclinicClone.git # timeout=10
Fetching & pruning origin...
Listing remote references...
> git config --get remote.origin.url # timeout=10
> git --version # timeout=10
> git --version # 'git version 2.34.1'
> git ls-remote -h -- https://github.com/GitPracticeRepo/MySpringPetclinicClone.git # timeout=10
Fetching upstream changes from origin
> git config --get remote.origin.url # timeout=10



User Management

- Jenkins does have two ways of dealing with users
    - Users from external sources like Active Directory, Open LDAP
    - Users created and maintained in Jenkins

- Navigate to Manage Jenkins -> Global Security
- For Authorization. Generally
    - Matrix based Security
    - Project Based Matrix Authorization.

Note

- Jenkins plugins can be installed by Manage Pluigns -> Advanced and then Deploy. Jenkins plugin will be of the extension hpi (hudson plugin interface) or jpi (jenkins plugin interface)

- Cloudbees is the organization supporting jenkins and for major contributions to jenkins open source. Cloudbees offers enterprise support to jenkins , This edition of jenkins is referred as Cloudbees enterprise Jenkins.
- Jenkins has two more interfaces

    - REST API Refer Here
    - Jenkins CLI Refer Here

- Blue Ocean is a new UI for jenkins pipeline projects Refer Here

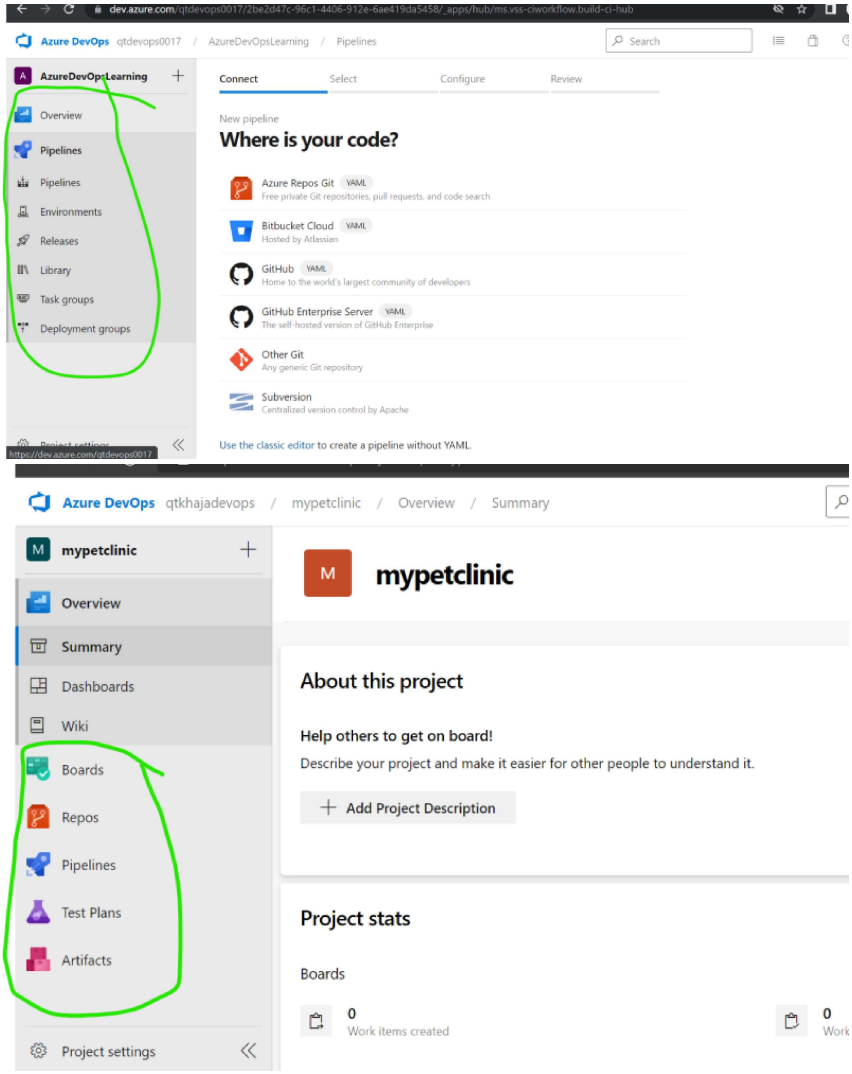
MENU ☰

OCTOBER 2, 2022

DevOps Classroomnotes 02/Oct/2022

Azure DevOps

- This is a suite of Products by microsoft and opensources to manage the lifecylcle of the application
- Azure DevOps has features for
    - Project Management with Agile
        - Product Backlogs
        - Sprint Backlogs
        - Defect Management
        - Release Management
        - Test Case Management
    - For Documentations wiki is supported
    - For CI/CD it has Pipelines (in yaml format)
    - For Storing Package (Package Repository) => Azure Artifacts
- Azure DevOps has two ways of usage
    - Self Hosted: Azure DevOps Express and Enterprise edition can be installed on Windows Servers Refer Here
    - Cloud Hosted: Refer Here
- Once you create the free account in Azure DevOps, An organization will be created and then we are asked to create a project



- Documentations for your activities are done in markdown format Refer Here. For this we use Wiki (Azure DevOps) / Confluence (Atlassian – JIRA ALM)

Important Terms

- Project Management/Working
    - Agile Project
    - Scrum
    - Product Backlog
    - Sprint Backlog
    - Sprint Formalities
        - Sprint Kick off
        - Daily Standup
        - Sprint Demo
        - Sprint Retrospective
    - Roles:
        - Team Member
        - Scrum Master
        - Product Owner
- Types of Tests which we Need to know as DevOps Engineers
    - Regression Test
    - Smoke Tests
    - Sanity Tests
    - Integration Test
    - White Box Testing:
        - Unit Tests
    - Black Box

- Selenium Browser Automation

- Grey Box
  - Postman

- Performance Testing
  - Jmeter

- Mean Time To Fail (MTTF)
- Mean Time To Recover (MTTR)

Leave a Reply

Enter your comment here...

This site uses Akismet to reduce spam. Learn how your comment data is processed.

About continuous learner

devops & cloud enthusiastic learner

VIEW ALL POSTS