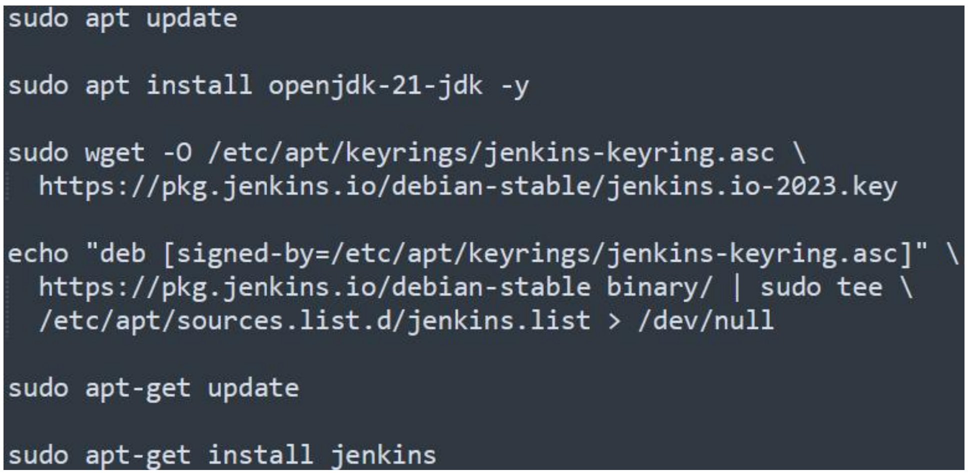
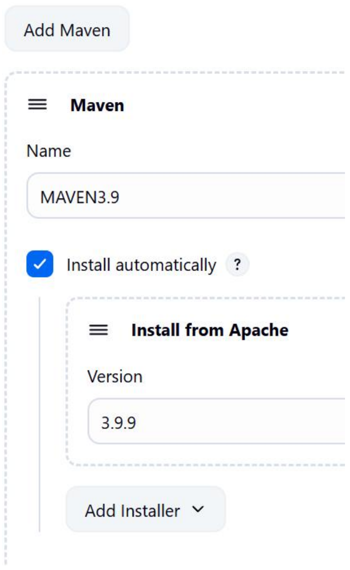
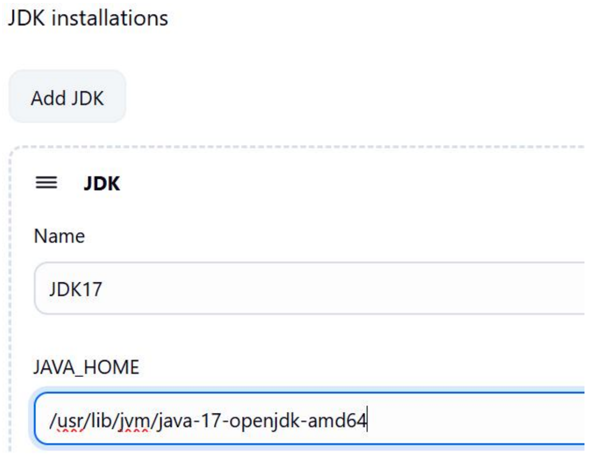
CI/CD With Jenkins

- To install Jenkins in Ubuntu:
  - You need to install Java because *Jenkins is written in Java.*
  - Its not a native program (like .exe or .bin), rather it's a **.war** file (Java Web Application Archive).
  - To run it, you need JVM (Java Virtual Machine), which comes from JDE/JRE.
  - 

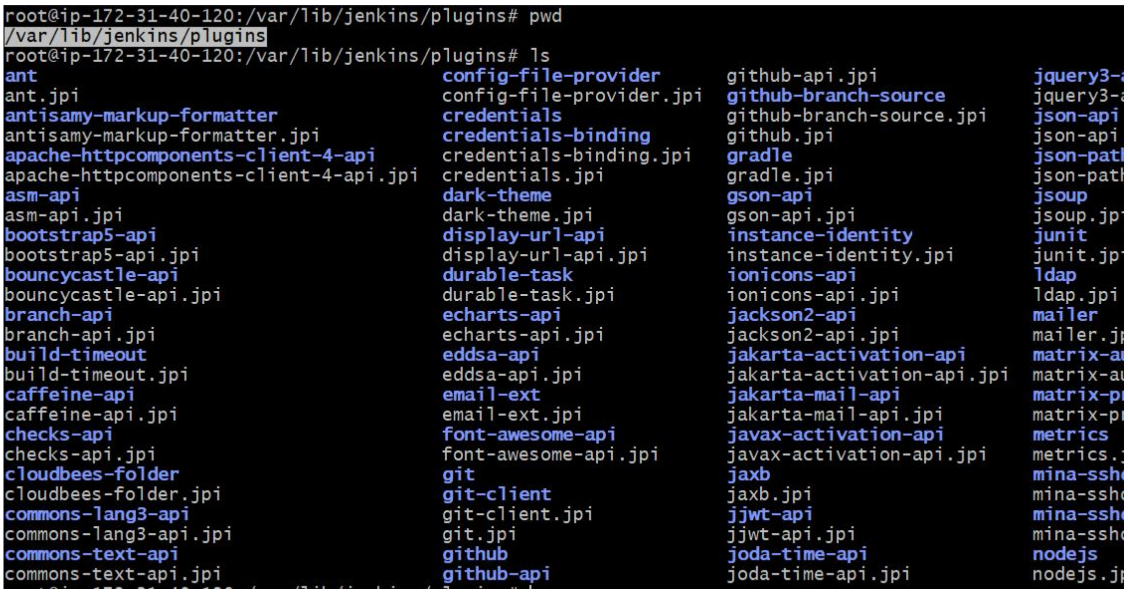
```
sudo apt update

sudo apt install openjdk-21-jdk -y

sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

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

sudo apt-get update

sudo apt-get install jenkins
```

  - **/var/lib/jenkins** is the home directory of Jenkins. You can see this inside /etc/passwd
  - Inside **/var/lib/jenkins** the jenkins configuration (**config.xml**) file exists.
  - After installing jenkins, you can copy the *public IP* of the instance and open in the browser with port *8080* (remember: TCP with port 8080 should be present in the security group attached to the ubuntu instance).
    - After opening the browser, it'll show one path where the initial password is present.
    - **/var/lib/jenkins/secrets/initialAdminPassword** : In this file the initial password is stored.
  - *If you can't open the jenkins ui through browser, then try updating the security inbound rule for TCP 8080 traffic for all IPv4. sometimes, My IP doesn't work.*
  - Change the **jenkins url** to a random domain. Otherwise it'll try to access that public ip only. If your instance is rebooted, then the public IP will be changed, and Jenkins will become slow.

- <u>**Jobs in Jenkins**</u>
  - **Freestyle Job**
    - In freestyle, everything is configured in the Jenkins UI.
    - ***Graphical Jobs***.
    - Each job has a GUI form where you define:
      - Where to get code (GitHub, SVN, etc.)
      - Build steps (e.g., mvn clean install, npm build)
      - Post-build actions (e.g., deploy, send email)
    - **Pros:**
      - Easy to create (beginner friendly)
      - Great for simple projects
      - No need to learn syntax.
    - **Cons:**
      - Hard to maintain (if there are many jobs, have to edit each of them manually)
      - Not portable (configs only stay in Jenkins server, not git repo)
      - Limited flexibility (complex workflows are difficult to manage)
      - If jenkins crashes, you loose job definitions (unless backed up)
  - **Pipeline As A Code**
    - Instead of configuring Jobs in UI, **Jenkinsfile** is used.
    - Jenkins read the file and runs the pipeline automatically.
    - Written in Groovy based DSL (Domain Specific Language)
- <u>**Plugins vs Tools**</u>
  - Simple analogy:
    - Keywords:
      - Programmer (Jenkins)
      - Programming Language (Plugin)
      - Tools (Laptop with compiler installed)
    - If a programmer knows the language (jenkins have plugins installed) but doesn't have a laptop (the server where jenkins present, doesn't have that tool): then it'll be of no use
    - If a programmer doesn't know the language (jenkins don't have the plugin) and he is given a laptop (the server where jenkins is present, have the tools installed): then it'll be of no use
  - ***Plugins** tell Jenkins how to do things; **Tools** let Jenkins actually do the work.*

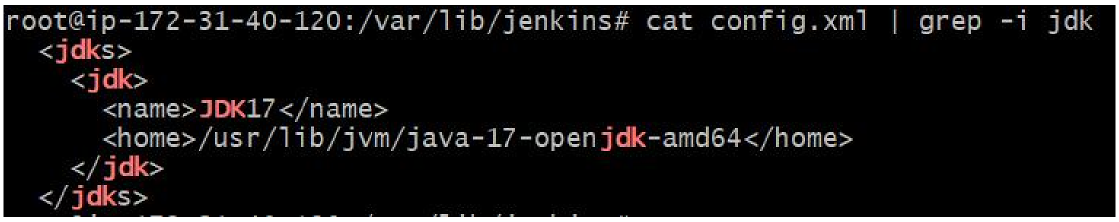
- You can install the tools in the server directly executing the command like **apt install maven** ..etc. OR you can do from the Jenkins GUI as well.
  - Note:
    - In GUI, it'll display only those Tools, whose Plugins are installed.
    - If you don't see the particular Tool you want, then install its Plugin first.
    - If you'll install the Tools via system CLI directly; then also it'll be of no use if the Plugin is not installed in Jenkins.
- Ex: I am installing Maven (tool) via GUI
  - 
    Add Maven
    Maven
    Name: MAVEN3.9
    Install automatically ✓
    Install from Apache
    Version: 3.9.9
    Add Installer
  - Its simple, just give a name and select the version.
- Ex-2: I am installing JDK via GUI. Its little different
  - 
    JDK installations
    Add JDK
    JDK
    Name: JDK17
    JAVA_HOME: /usr/lib/jvm/java-17-openjdk-amd64
  - Its little different. Installed java-17 version in cli, then gave its home directory path in GUI.

- The tools whose multiple versions can be installed at once in a system *(multiple versions of JDK can be installed in a system)*, we need to tell jenkins that which version is to be used by giving that version's home directory path.
- The installed plugins stay in the directory: /var/lib/jenkins/plugins



```
root@ip-172-31-40-120:/var/lib/jenkins/plugins# pwd
/var/lib/jenkins/plugins
root@ip-172-31-40-120:/var/lib/jenkins/plugins# ls
ant                                     config-file-provider      github-api.jpi              jquery3-
ant.jpi                                 config-file-provider.jpi  github-branch-source        jquery3-
antisamy-markup-formatter               credentials               github-branch-source.jpi    json-api
antisamy-markup-formatter.jpi           credentials-binding       github.jpi                  json-api
apache-httpcomponents-client-4-api      credentials-binding.jpi   gradle                      json-pat
apache-httpcomponents-client-4-api.jpi  credentials.jpi           gradle.jpi                  json-pat
asm-api                                 dark-theme                gson-api                    jsoup
asm-api.jpi                             dark-theme.jpi            gson-api.jpi                jsoup.jp
bootstrap5-api                          display-url-api           instance-identity           junit
bootstrap5-api.jpi                      display-url-api.jpi       instance-identity.jpi       junit.jp
bouncycastle-api                        durable-task              ionicons-api                ldap
bouncycastle-api.jpi                    durable-task.jpi          ionicons-api.jpi            ldap.jpi
branch-api                              echarts-api               jackson2-api                mailer
branch-api.jpi                          echarts-api.jpi           jackson2-api.jpi            mailer.j
build-timeout                           eddsa-api                 jakarta-activation-api      matrix-a
build-timeout.jpi                       eddsa-api.jpi             jakarta-activation-api.jpi  matrix-a
caffeine-api                            email-ext                 jakarta-mail-api            matrix-p
caffeine-api.jpi                        email-ext.jpi             jakarta-mail-api.jpi        matrix-p
checks-api                              font-awesome-api          javax-activation-api        metrics
checks-api.jpi                          font-awesome-api.jpi      javax-activation-api.jpi    metrics.
cloudbees-folder                        git                       jaxb                        mina-ssh
cloudbees-folder.jpi                    git-client                jaxb.jpi                    mina-ssh
commons-lang3-api                       git-client.jpi            jjwt-api                    mina-ssh
commons-lang3-api.jpi                   git.jpi                   jjwt-api.jpi                mina-ssh
commons-text-api                        github                    joda-time-api               nodejs
commons-text-api.jpi                    github-api                joda-time-api.jpi           nodejs.j
```

- All global tools configurations (JDK, Maven, Git, Node.js etc) are stored inside: /var/lib/jenkins/hudson.tasks.*
  - Exception: JDKs are stored inside **/var/lib/jenkins/config.xml** because Jenkins treats them as a core runtime tool
  - If you have not updated the JDK in Jenkins UI, then you can't see the JDK inside that **config.xml**. And Jenkins will use the default JDK that is present globally (in my case, global default was JDK version 21).



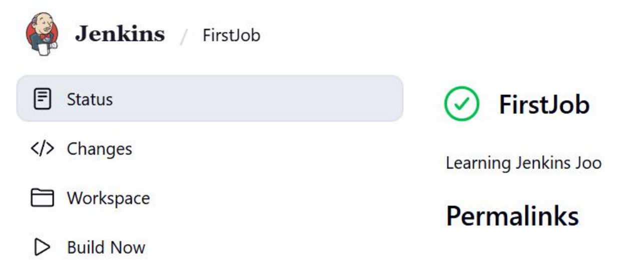
```
root@ip-172-31-40-120:/var/lib/jenkins# cat config.xml | grep -i jdk
  <jdks>
    <jdk>
      <name>JDK17</name>
      <home>/usr/lib/jvm/java-17-openjdk-amd64</home>
    </jdk>
  </jdks>
```

  - If multiples JDKs are configured inside this, then whatever version mentioned in the Job will be used while running the Job inside pipeline.

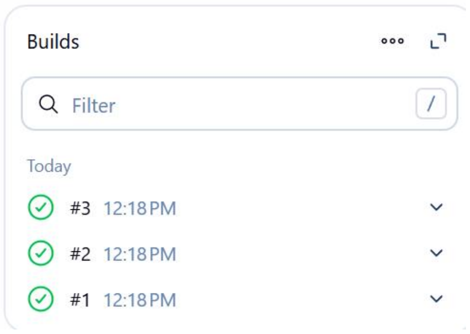
➢ Lets create out first Job

- Create **FreeStyle** project.
  - Give one description like "Learning Jenkins Jobs"
  - Skip **Triggers** and **Environments** for now.

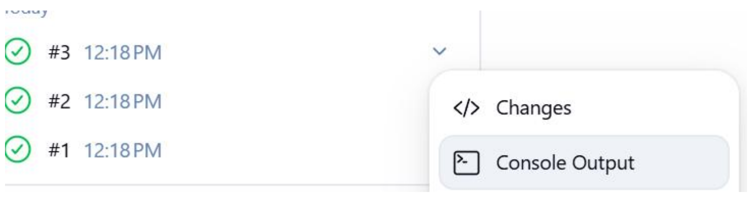
- Under **Build Steps**, select **Execute Shell** (the windows part like **execute windows batch commands** will not work as the Jenkins is hosted in Ubuntu in our case).
- **Save** this now.



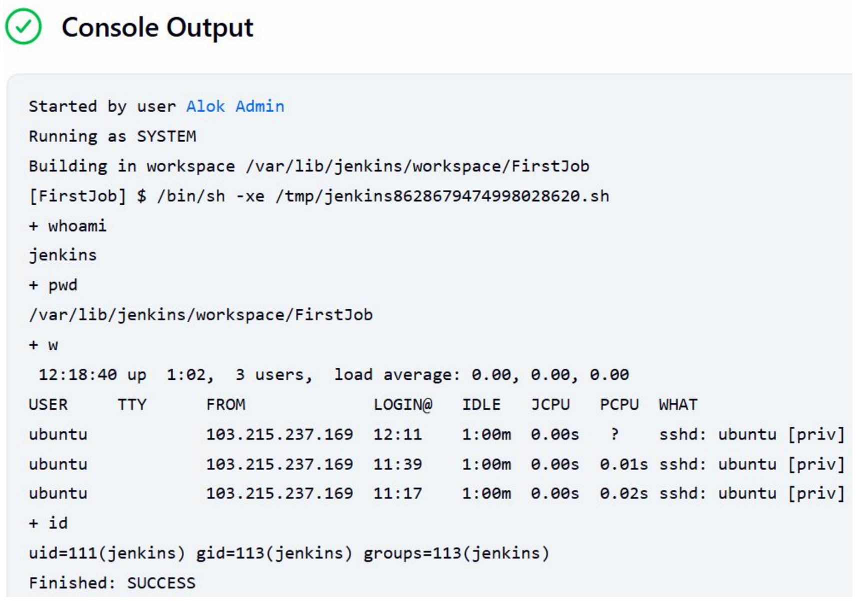
- Under the created Job, click on that **Build Now** button 2 or 3 times.



- You'll see something like this.



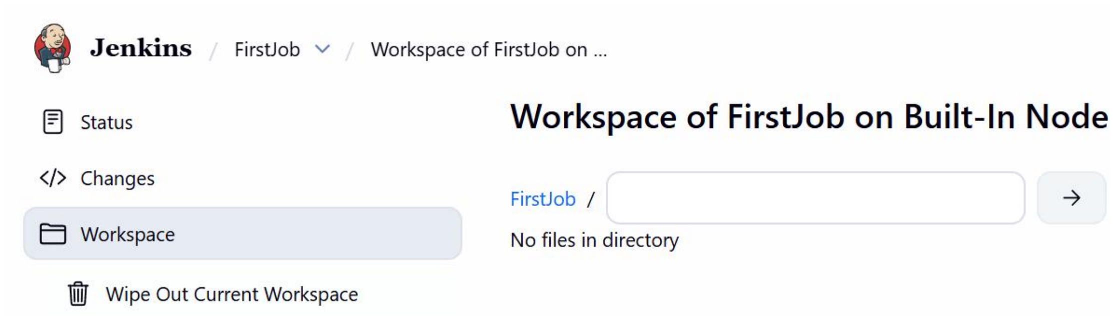
- You can also see the console output of the build.



Console Output

```
Started by user Alok Admin
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/FirstJob
[FirstJob] $ /bin/sh -xe /tmp/jenkins8628679474998028620.sh
+ whoami
jenkins
+ pwd
/var/lib/jenkins/workspace/FirstJob
+ w
 12:18:40 up  1:02,  3 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU  WHAT
ubuntu            103.215.237.169  12:11    1:00m  0.00s   ?    sshd: ubuntu [priv]
ubuntu            103.215.237.169  11:39    1:00m  0.00s  0.01s sshd: ubuntu [priv]
ubuntu            103.215.237.169  11:17    1:00m  0.00s  0.02s sshd: ubuntu [priv]
+ id
uid=111(jenkins) gid=113(jenkins) groups=113(jenkins)
Finished: SUCCESS
```

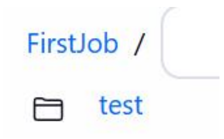
- You can see the path where the Job ran was **/var/lib/jenkins/workspace/FirstJob**

- You can see some folders inside the path **/var/lib/jenkins**, in which **jobs** and **workspace** are there.
  - **jobs**
    - It contains every detail about the job.
    - Like the build history, configurations, metadata etc.
  - **workspace**
    - It is where **Jenkins** actually run build the code and do stuffs.
    - You can think it like it's a local folder for the **Jenkins user** where it does the things like pulling any repo, building that and testing etc etc.
    - 

      Workspace of FirstJob on Built-In Node

      FirstJob /

      No files in directory
    - Here there is an option **Workspace**, which remain in sync with the path **/var/lib/jenkins/workspace**.
    - I created one folder inside that path manually using **mkdir** command inside the linux and now it came inside the Jenkins website as well.

      FirstJob /

      test

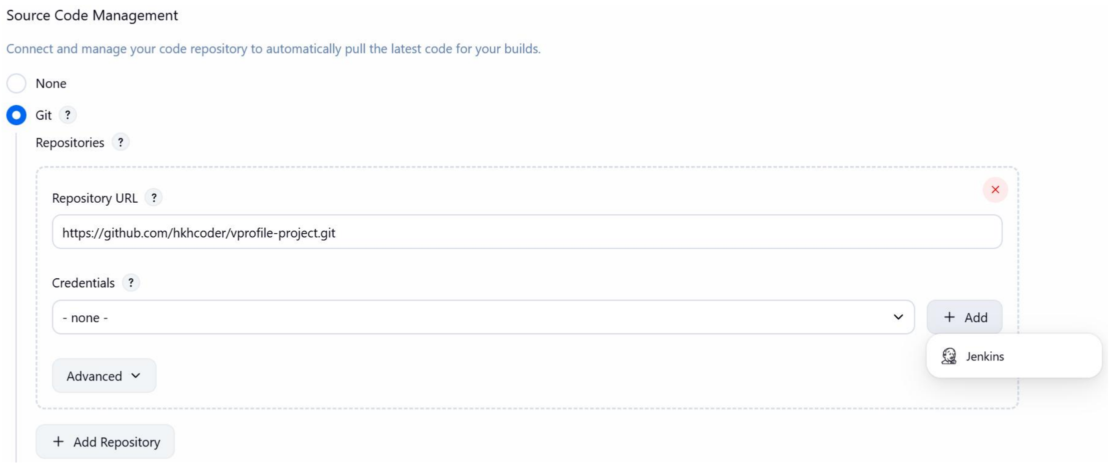
➢ **Note**

- The tools that we configure are available globally for all the jobs. Its not bounded to any particular job.
- Lets suppost JDK, if I have 2 different JDK present inside the tools, then inside the Job, I can select which JDK will be used in my current Job.

➢ **Creating another job to build the vprofile project from github**

- Give a name and description to the job. (it is also **Free Style**).
- Select the JDK version. (I chose 17)
- Source Code Management: Choose **Git**.
  - If the repo is public, then no need to give the credentials.

- Otherwise you need to give clicking on that Add button present in the right.



Source Code Management

Connect and manage your code repository to automatically pull the latest code for your builds.

None

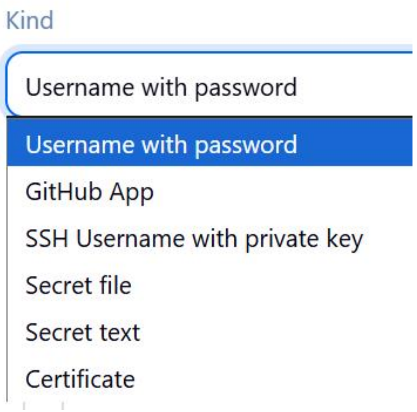
[✓ Git]

Repositories

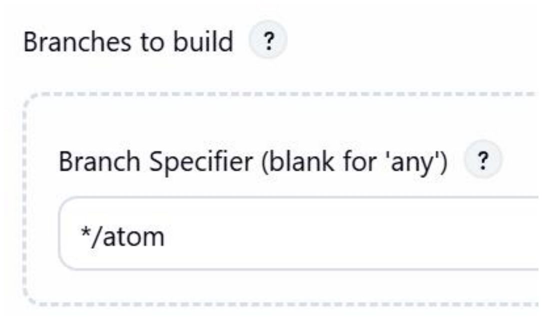
Repository URL: https://github.com/hkhcoder/vprofile-project.git

Credentials: - none -

- You have so many methods using which you can connect to Github.



Kind: Username with password

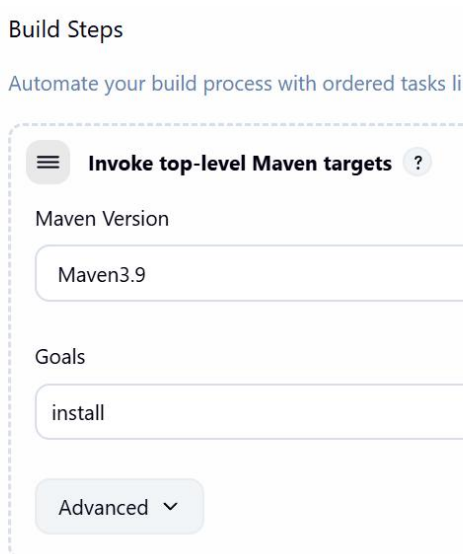


Branches to build

Branch Specifier (blank for 'any'): */atom

  - Also select the branch from which the code will be build.

- In the previous job, we used **Execution Shell**. But its not recommended.
  - Every time use Plugins to do some specific task.
  - If there is no plugin to do the task you are interested in, then only you should write commands in **Execution Shell**.
  - Here, I chose **Invoke top-level Maven targets**, chose the maven version and the command in the goal i.e. **install** because I want to build the source code.
  - You have some advanced settings as well that you can checkout.


Build Steps

Automate your build process with ordered tasks li

Invoke top-level Maven targets

Maven Version

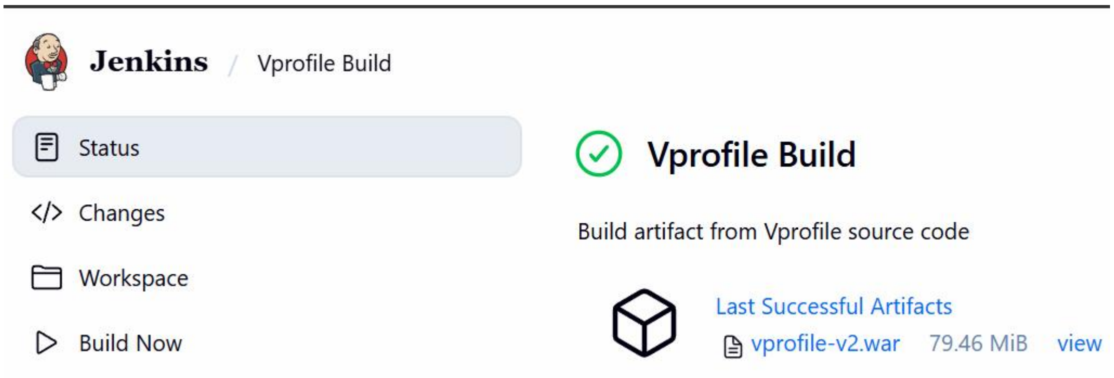
Maven3.9

Goals

install

Advanced

- Now Lets see the **Post-Build Actions**
    - I chose **Archive the artifacts** and gave ****/*.war** inside the input field *Files to archive*.
        - ****** means it'll go to every sub-directory and check if any ***.war** file present and archive that.
    - It stores the archived file in somewhere else and give you one link to download or view that. (in the **status** section)


Jenkins / Vprofile Build

Status

Changes

Workspace

Build Now

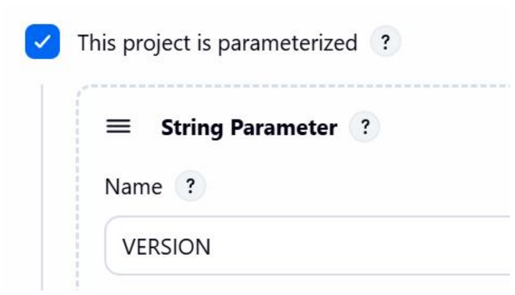
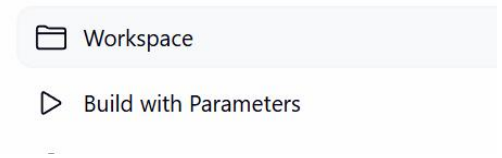
Vprofile Build

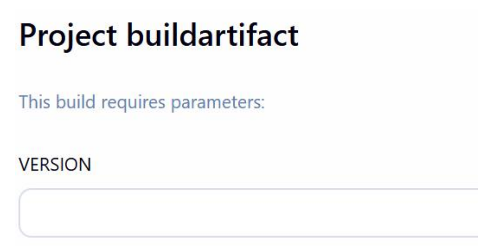
Build artifact from Vprofile source code

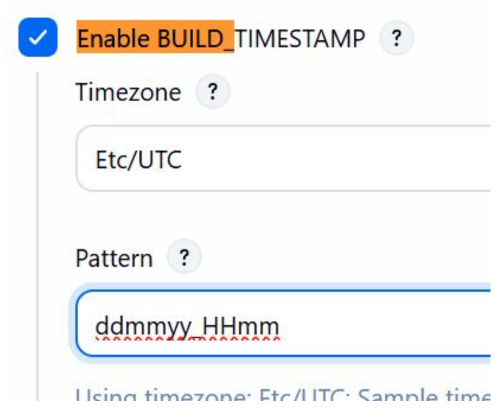
Last Successful Artifacts

vprofile-v2.war 79.46 MiB view

- **IMPORTANT**
    - When we install any tools from the Jenkins, it install the tool in the Linux (or whatever server where Jenkins is hosted) for the **Jenkins** user only; not **globally**.
    - I installed **maven3.9** in the tools section of **Jenkins**.
    - Ran one job 2 or 3 times (PS: inside the job under the **invoke top-level Maven targets** the **maven3.9** was selected).
    - Then I selected **Default** instead of **maven3.9** in that drop-down and ran built the job again. Now it failed.

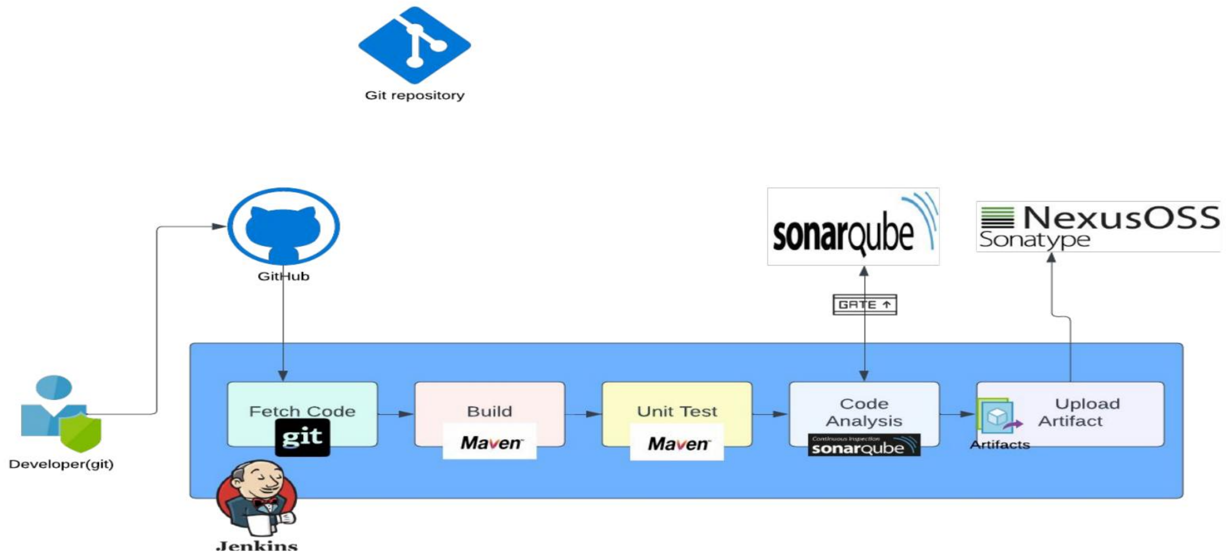
        - Because, when you choose **default** in that option, it checks **system default maven**, i.e. inside **/usr/bin/mvn** folder which is accessible globally. But maven is not installed in our server globally.
        - So, you need to install **maven** in the **linux server *globally*** then build the job again. Now it'll pass.
        -
    - When you create a new job, at the bottom there is an option **Copy from**, there you can give the name of any existing job you have.
        - It'll copy all the configs from there to this new job by default.
        - Means all the fields will be **auto-selected** according to that reference Job.
    - When you install any plugins, then only it'll be visible in the job.
- Just like Gitlab CI/CD, Jenkins also has **environment variables** like **BUILD_ID, BUILD_NUMBER** ..etc etc.
- You can use your **own variables** inside the job.
    - 
    This project is parameterized ?
    String Parameter ?
    Name ?
    VERSION
        - Inside the configure section, select this checkbox "**This project is parameterized**"
    - Then you'll get the button **Build with Parameters** in place of **Build now**.
        - 

    - When you click that **Build with parameters** button, you'll get one page where you can enter the values.
        - 
        Project buildartifact
        This build requires parameters:
        VERSION
    - Also, you can add the **default value** in that **configure** page.

- Inside the **Manage Jenkins** path, there is an option **System.**
  - Here you can configure the global configurations. (its not specific to any particular Job)
  - 
    Enable BUILD_TIMESTAMP
    Timezone
    Etc/UTC
    Pattern
    ddmmyy_HHmm
    Using timezone: Etc/UTC; Sample time
    (Here I changed the timestamp pattern)

  -
    ```
    mkdir -p versions
    # cp target/vprofile-v2.war versions/vpro$BUILD_ID.war
    #cp target/vprofile-v2.war versions/vpro$VERSION.war
    cp target/vprofile-v2.war versions/vpro$BUILD_TIMESTAMP.war
    ```
    - Added in the **execution shell** in **Build Steps**.
- **Disk Space Issue**
  - Whenever you get any issue for disk space, just increase the volume capacity.

➢ Flow of Continuous Integration Pipeline



-
- SonarQube analyse the code and generate report in **XML** format which will be uploaded to the **sonarqube** server.
  - Also we can build one quality **gate** means if the code doesn't follow the required practices then fail the build.
  - If it fails, then pipeline will stop.
- If the pipeline passes, we'll have a verified copy of the artiface.
  - Now we can distribute the artifact to be deployed on the server.
  - Before deploying, the artifacts should be versioned and uploaded to NexusSonatype repository.

- **Steps for Continuous Integration Pipeline**
  - Jenkins setup
  - Nexux setup
  - Sonarqube setup
  - Security group
  - Install necessary plugins in Jenkins (like Nexus, Sonar, Git etc)
  - Integrate
    - Nexus
    - Sonarqube
  - Write pipeline script
  - Set notification
-
-
  - Dfd
  - Dfd
  -
  -