**Task: To generate a war file of the project in Jenkins with the help of Jenkinsfile. [ done by Prahallad Rao Gollamoni and Mohammad Ahsan Khan]**

**Jenkins File:**

Jenkins pipelines can be defined using a text file called **Jenkinsfile.**You can implement pipeline as code using Jenkins file, and this can be defined by using a domain specific language (DSL). With Jenkins file, you can write the steps needed for running a Jenkins pipeline.

**The benefits of using Jenkins File are:**

1. You can create pipelines automatically for all branches and execute pull requests with just one Jenkinsfile.
2. You can review your code on the pipeline.
3. You can audit your Jenkins pipeline.
4. This is the singular source for your pipeline and can be modified by multiple users.

**What is Jenkins:**

Jenkins is an open continuous integration server which has the ability to support the automation of software development processes. You can create multiple automation jobs with the help of use cases and run them as a Jenkins pipeline.

**Here are the reasons why you use should use Jenkins pipeline:**

1. Jenkins pipeline is implemented as a code which allows multiple users to edit and execute the pipeline process.
2. Pipelines are robust. So, if your server undergoes an unforeseen restart, the pipeline will be automatically resumed.
3. You can pause the pipeline process and make it wait to resume until there is an input from the user.
4. Jenkins Pipelines support big projects. You can run multiple jobs, and even use pipelines in a loop.

**Declarative versus Scripted pipeline syntax:**

There are two types of syntax used for defining your Jenkinsfile.

* Declarative
* Scripted

**Declarative:**

Declarative pipeline syntax offers an easy way to create pipelines. It contains a predefined hierarchy to create Jenkins pipelines. It gives you the ability to control all aspects of a pipeline execution in a simple, straight-forward manner.

**Scripted:**

Scripted Jenkins pipeline runs on the Jenkins master with the help of a lightweight executor. It uses very few resources to translate the pipeline into atomic commands. Both declarative and scripted syntax are different from each other and are defined totally different.

**Differences:**

1. The basic statements and expressions which are valid in Declarative Pipeline follow the same rules as [Groovy syntax](http://groovy-lang.org/syntax.html) with the following exceptions:
2. The top-level of the Pipeline must be a *block*, specifically: pipeline { }.
3. No semicolons as statement separators. Each statement has to be on its own line
4. Blocks must only consist of [Sections](https://jenkins.io/doc/book/pipeline/syntax/#declarative-sections), [Directives](https://jenkins.io/doc/book/pipeline/syntax/#declarative-directives), [Steps](https://jenkins.io/doc/book/pipeline/syntax/#declarative-steps), or assignment statements.
5. A property reference statement is treated as no-argument method invocation. So for example, input is treated as input().

**Steps to create war file using Jenkinsfile:**

1.Create a new file in the root folder of your application with name “Jenkinsfile”.

Place the following declarative pipeline script in Jenkinsfile to build a war file.

**pipeline**

**{**

**agent any**

**stages**

**{**

**stage('build')**

**{**

**steps**

**{**

**bat 'gradlew.bat clean build'**

**}**

**}**

**}**

**}**

**Sections mentioned in above file:**

**agent:** The agent section specifies where the entire Pipeline, or a specific stage, will execute in the Jenkins environment depending on where the agent section is placed. The section must be defined at the top-level inside the pipeline block, but stage-level usage is optional.

**Stage:** Containing a sequence of one or more stage directives, the stages section is where the bulk of the "work" described by a Pipeline will be located. At a minimum it is recommended that stages contain at least one stage directive for each discrete part of the continuous delivery process, such as Build, Test, and Deploy.

**Steps:** Declarative Pipelines may use all the available steps documented in the Pipeline Steps reference, which contains a comprehensive list of steps, with the addition of the steps listed below which are only supported in Declarative Pipeline.

2.Include war plugin in the “**build.gradle**” file and set enabled=true .(Since the default value of enabled is false for war plugin, the task war is skipped when we try to build. To generate a jar or war file, we will have to set it to true)

**plugins**

**{**

**id 'war'**

**}**

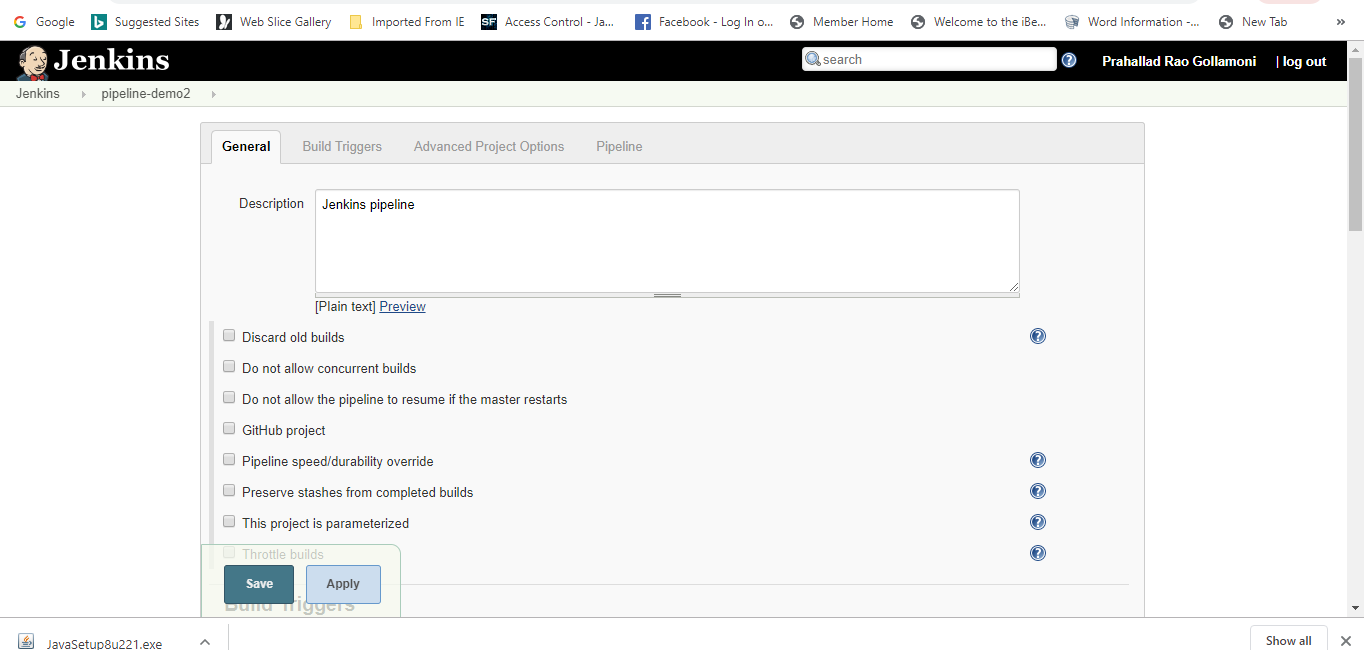
**war**

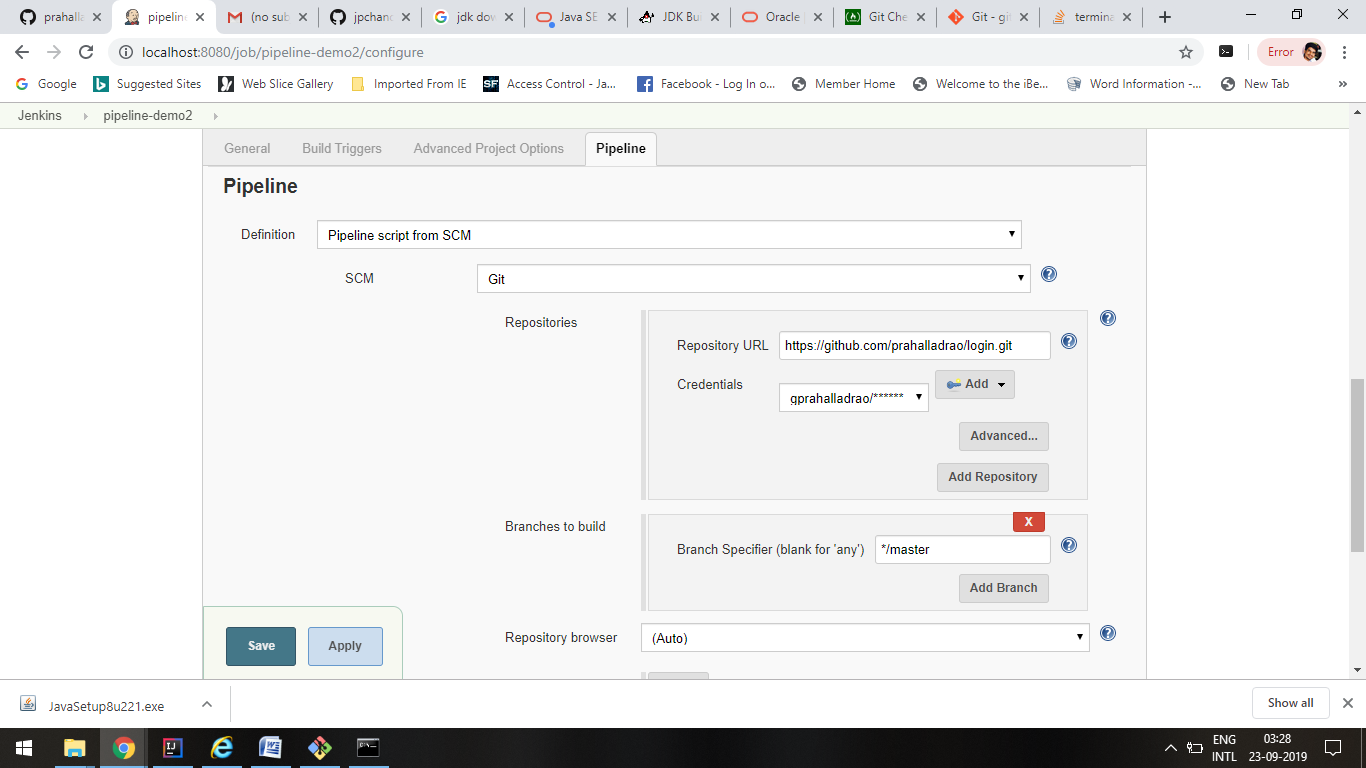
**{**

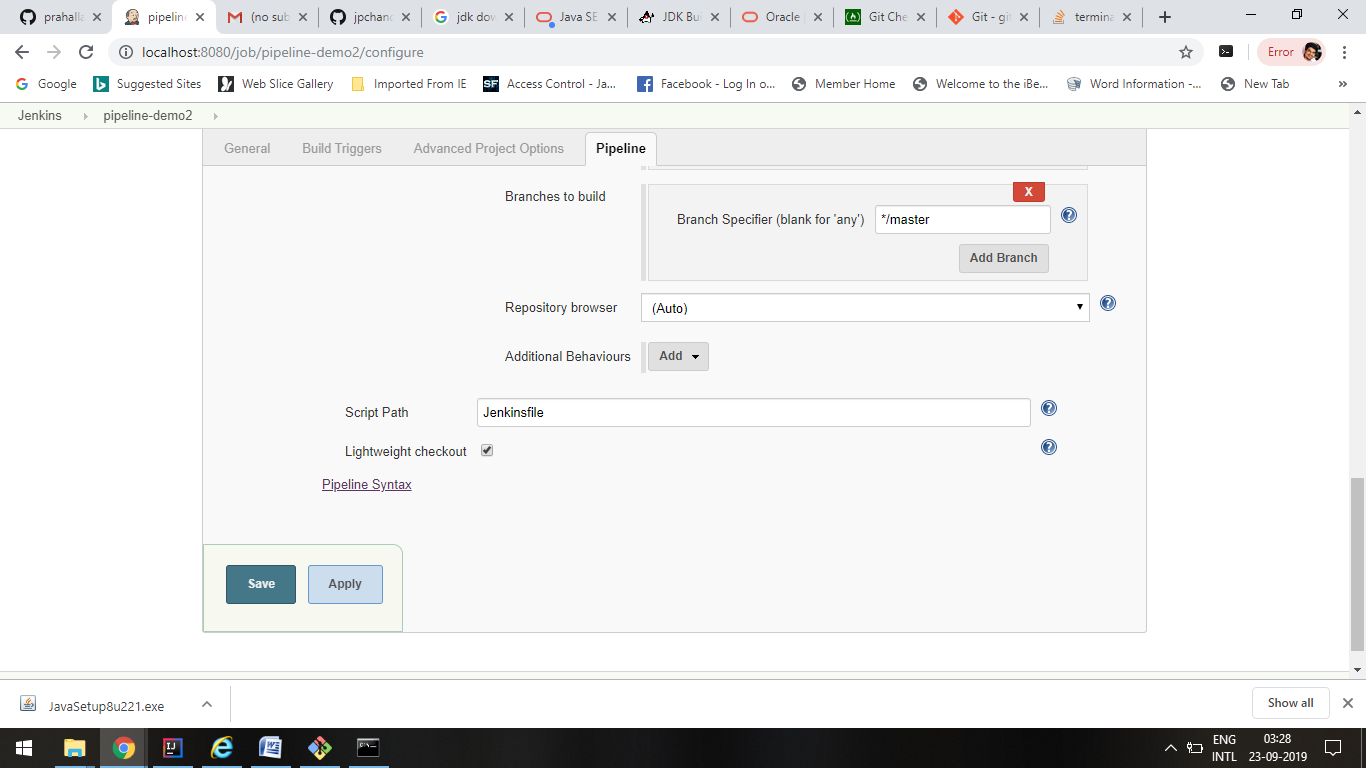
**enabled = true**

**}**

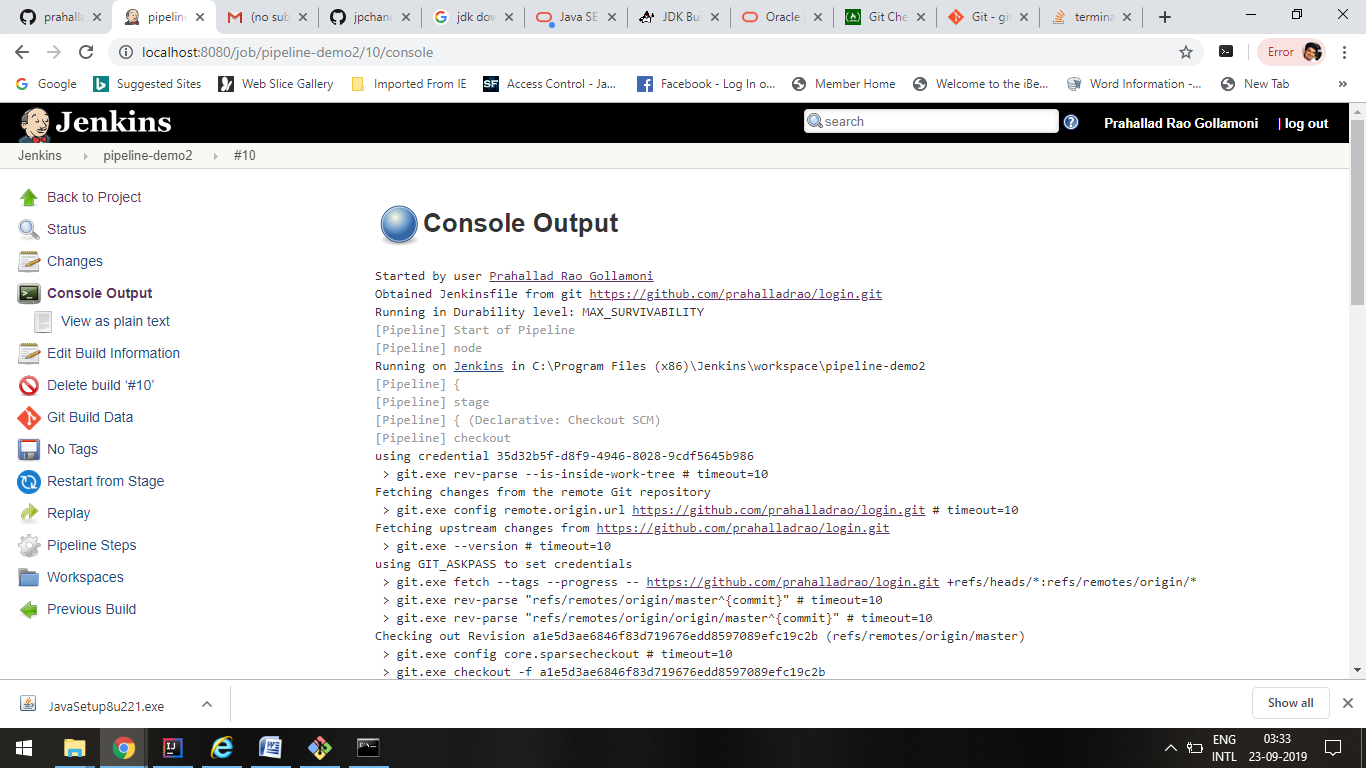
3. Create a new pipeline project in Jenkins. Set the Jenkinsfile path as shown below:

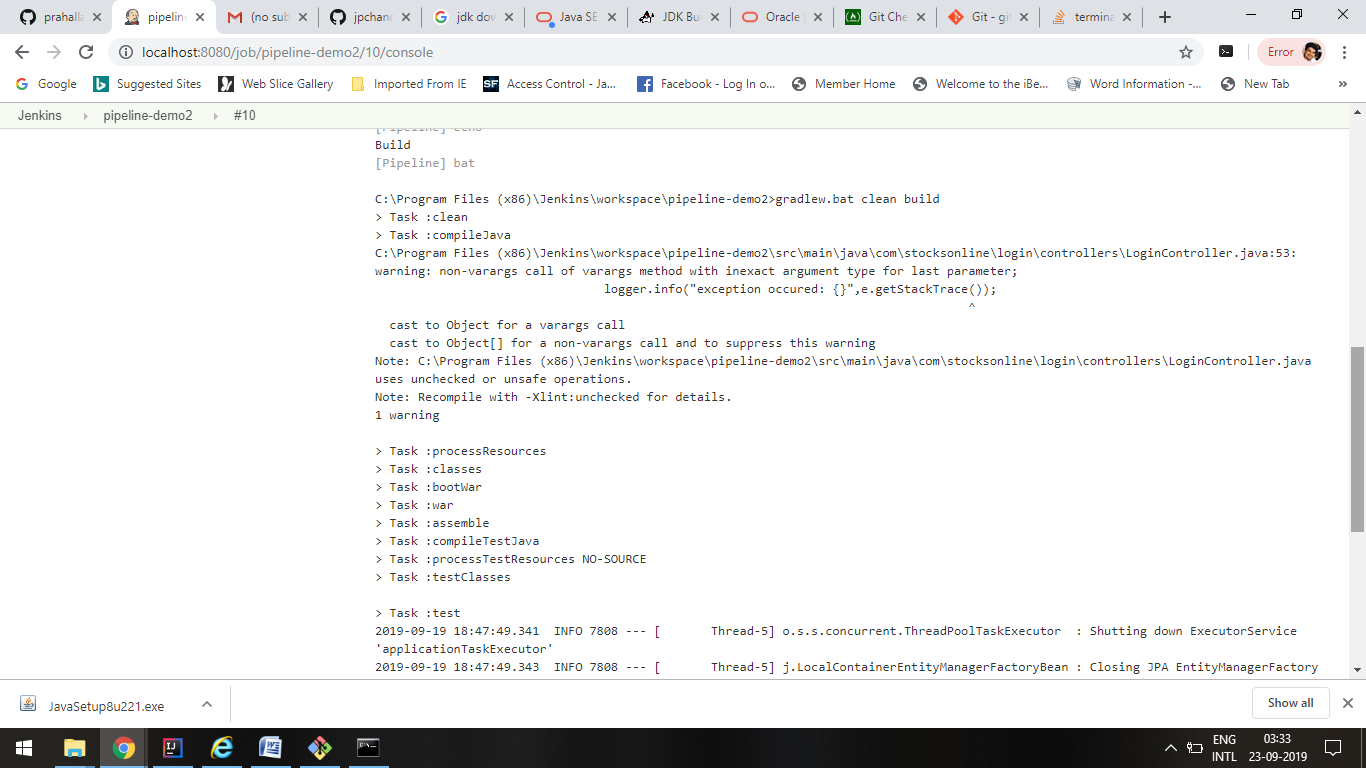


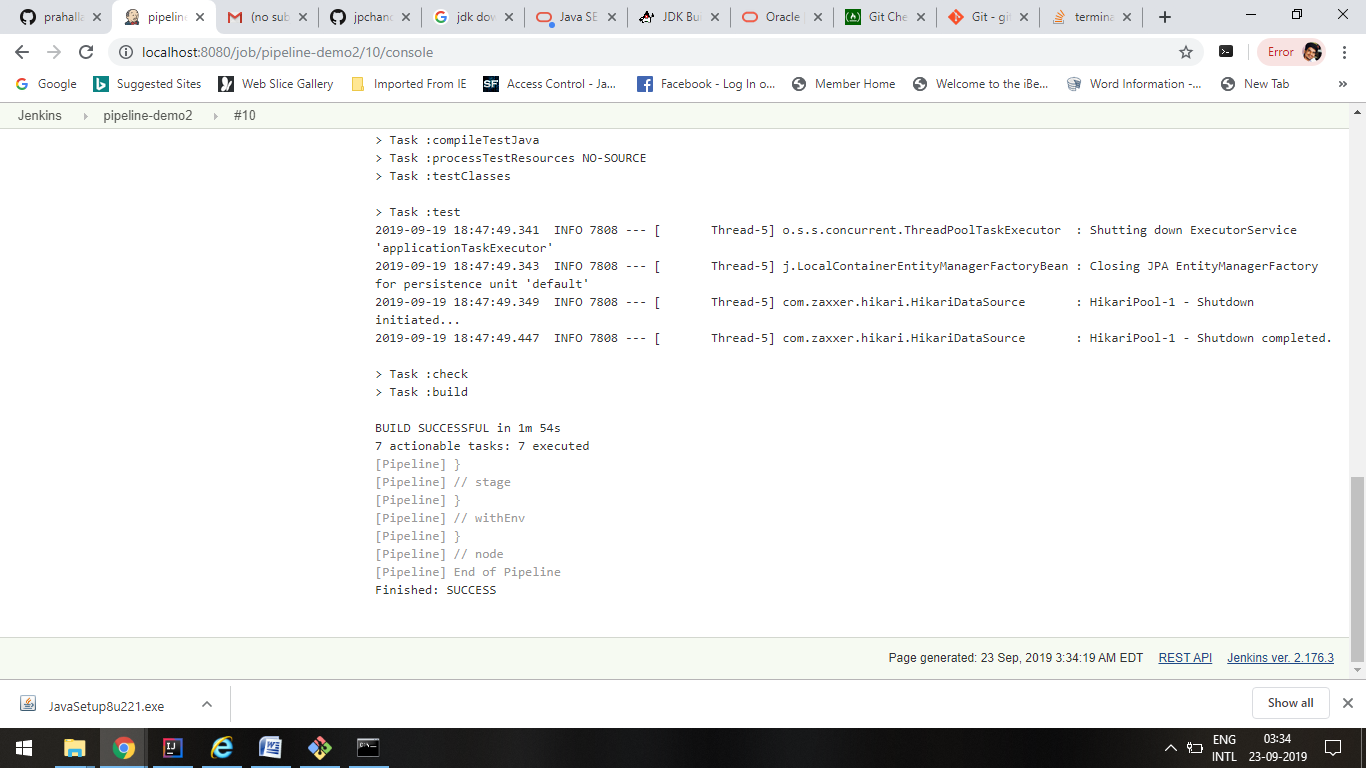




4.When we build the project, the console output of the build is as follows:







5.Successful build indicates that war file of the project is generated in the path **project/build/libs/**

