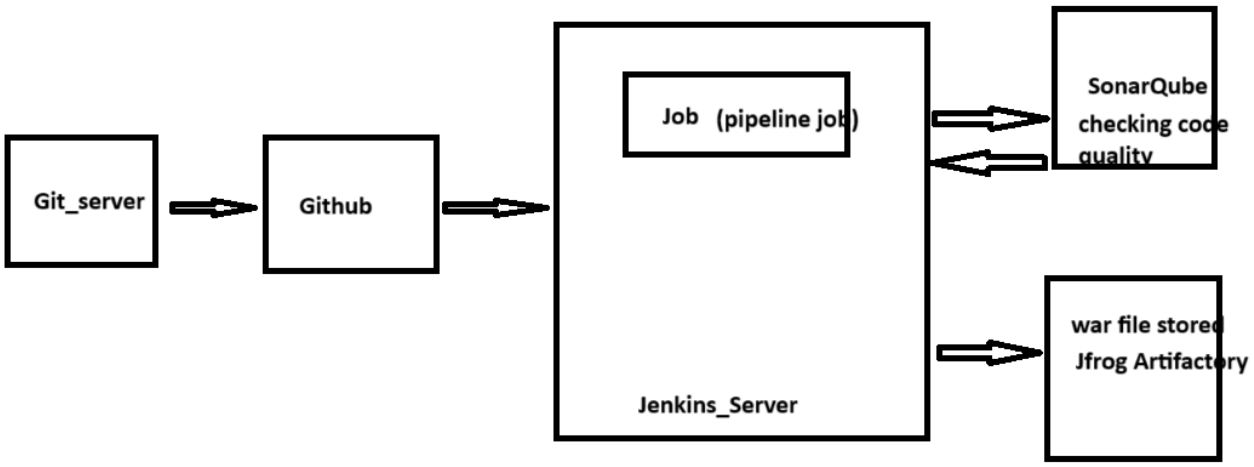
Sonarqube+jfrog

Project Overview

The document outlines a CI/CD pipeline integrating SonarQube and JFrog Artifactory with Jenkins. It begins with launching and configuring EC2 instances for developers and Jenkins, installing Java, Maven, Git, and required Jenkins plugins. A SonarCloud account is created to analyze code quality, with tokens added to Jenkins credentials. A JFrog Maven repository is set up for storing build artifacts, and access tokens are configured in Jenkins. The pipeline includes stages for building (mvn clean deploy), testing (unit tests via Maven Surefire), SonarQube analysis, and publishing JAR files to JFrog with metadata. If SonarQube quality gates are met, artifacts are stored, and their URLs can be shared for download. This setup ensures code quality checks, efficient artifact management, and seamless DevOps practices.

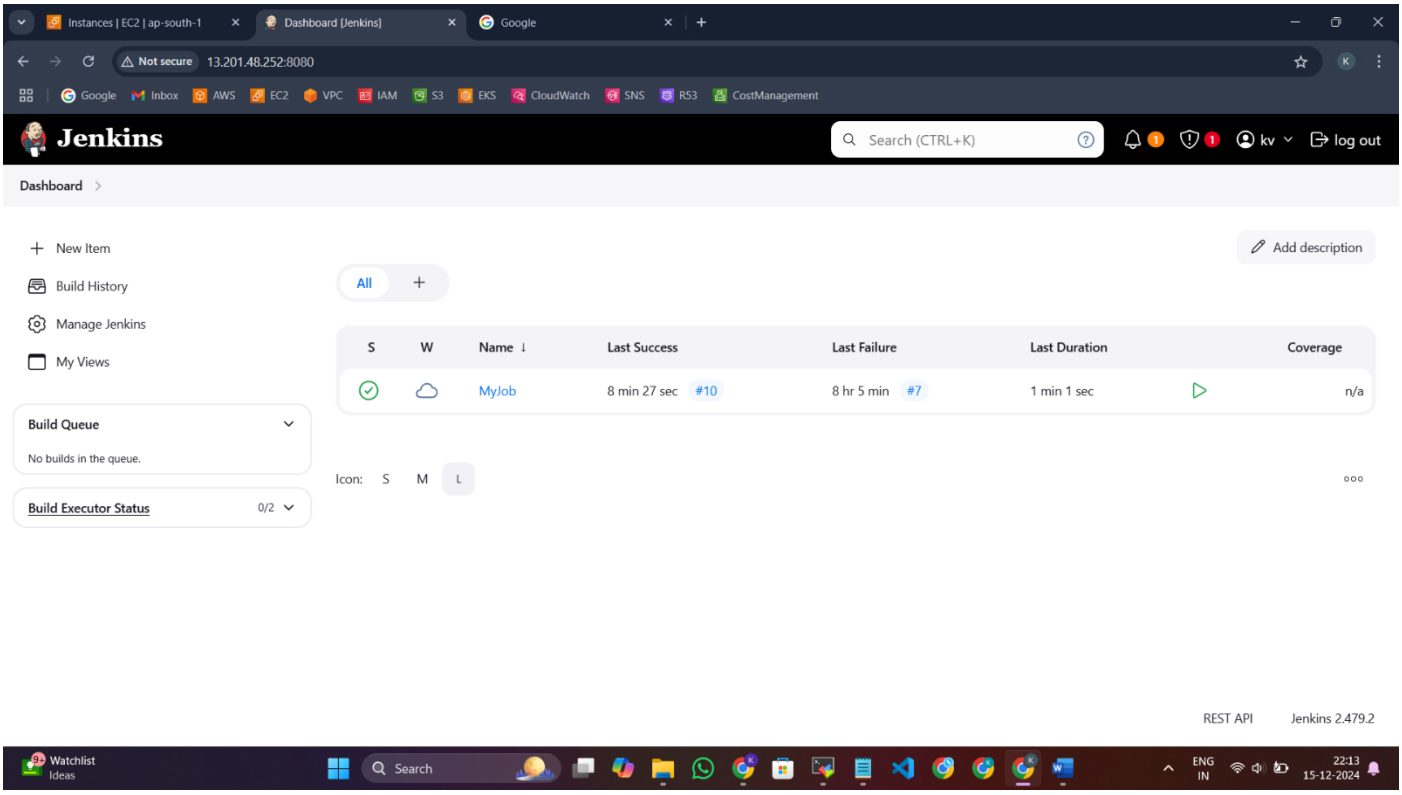
Technologies Used

- Git - For version control for tracking changes in the code files
- Maven – For Continuous Build
- Jenkins - For continuous integration and continuous deployment
- SonarQube - Code Quality Checking tool
- JFrog - Artifactory tool

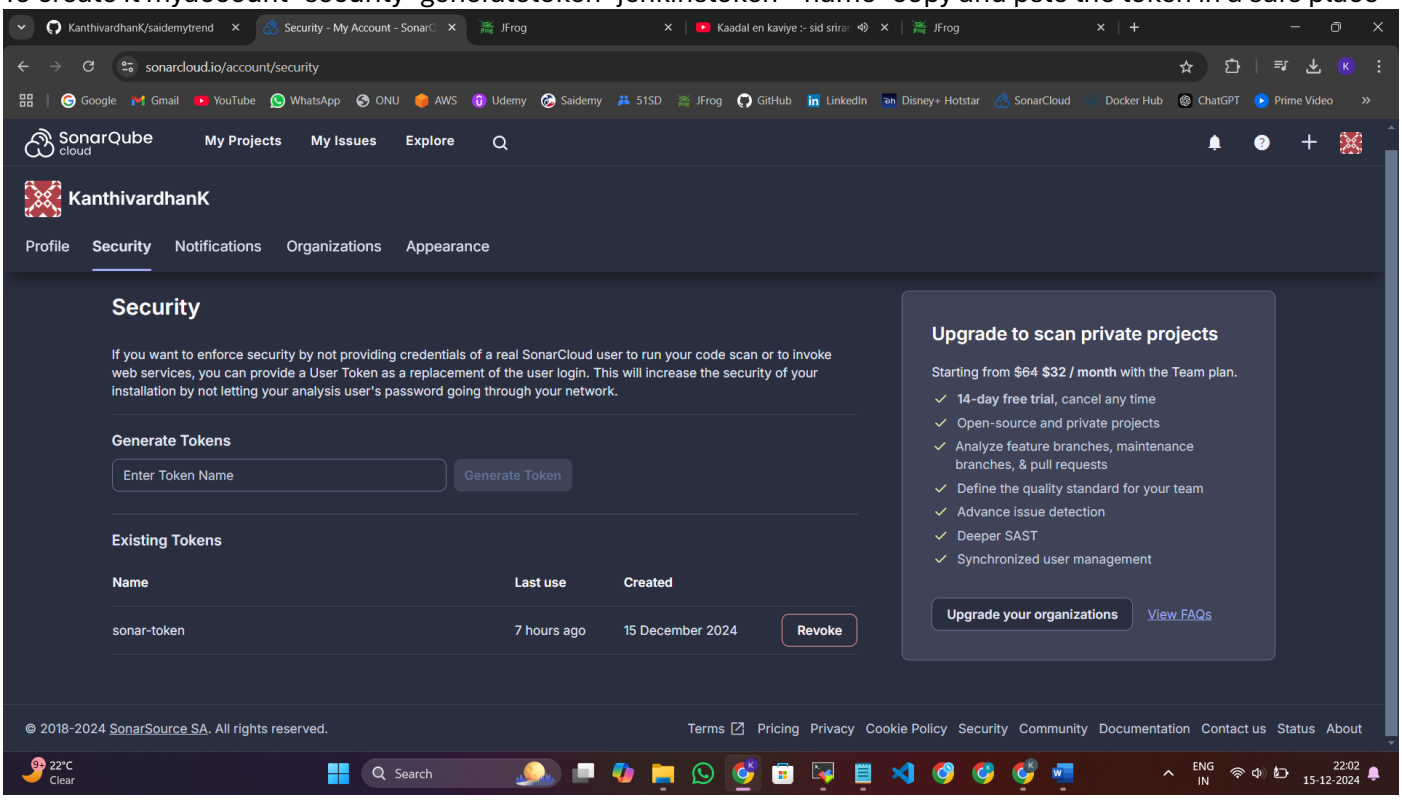


Launch developers ec2 instance
Install git
Configure the user.name and user.email
Clone the repo
Add the code and push the repo to github

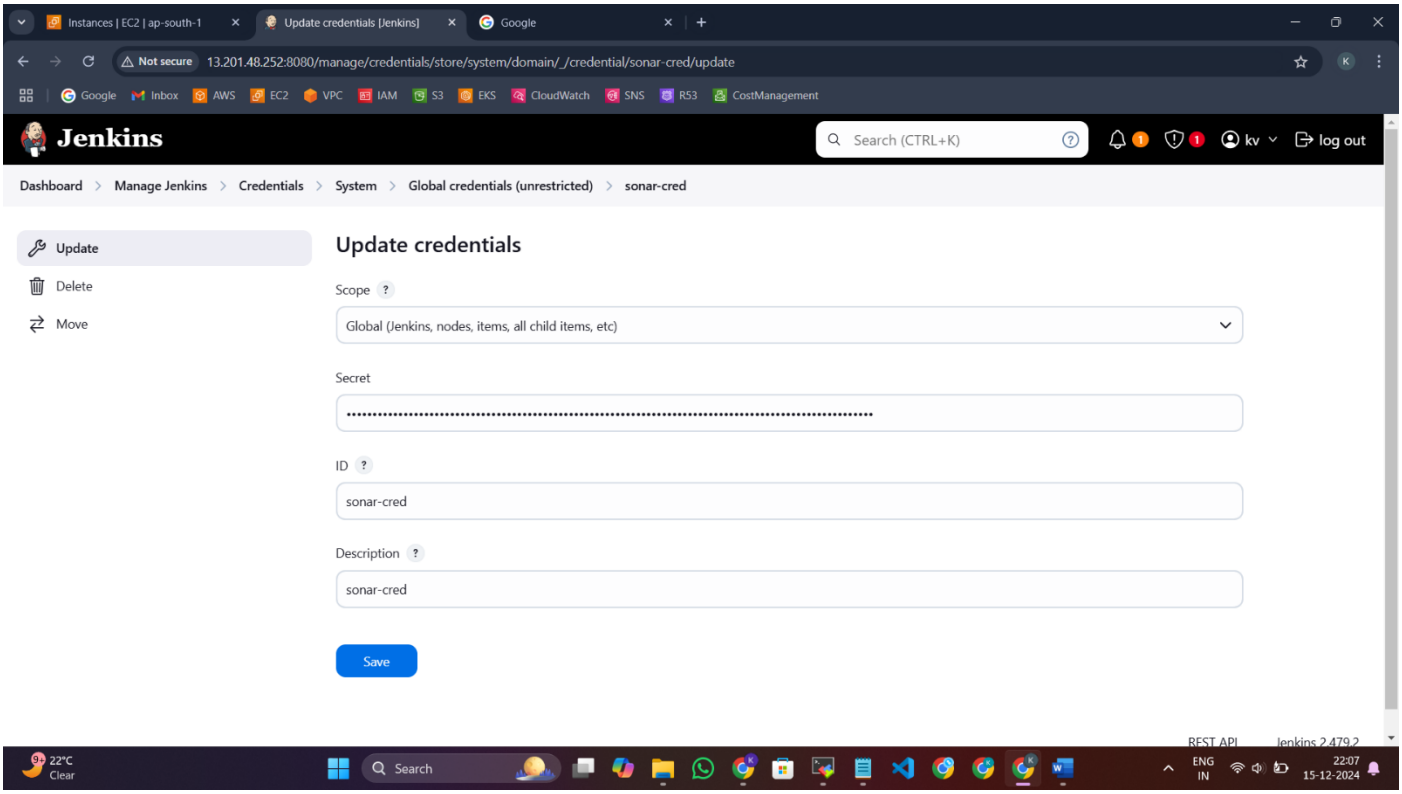
Launch an ec2 instance with t2.medium and 25 Gi for Jenkins
Install java, maven and configure the environment variables in .bash_profile
Install git
Connect to Jenkins dashboard
Install plugins-maven invoker, maven integration, git, github, sonarqube scanner, artifactory
Configure the maven. git and java path in Jenkins>manage Jenkins> tools



Create account in sonar cloud
Create an authentication token which is used to connect with Jenkins
To create it myaccount>security>generatetoken>jenkinstoken->name>copy and pste the token in a safe place

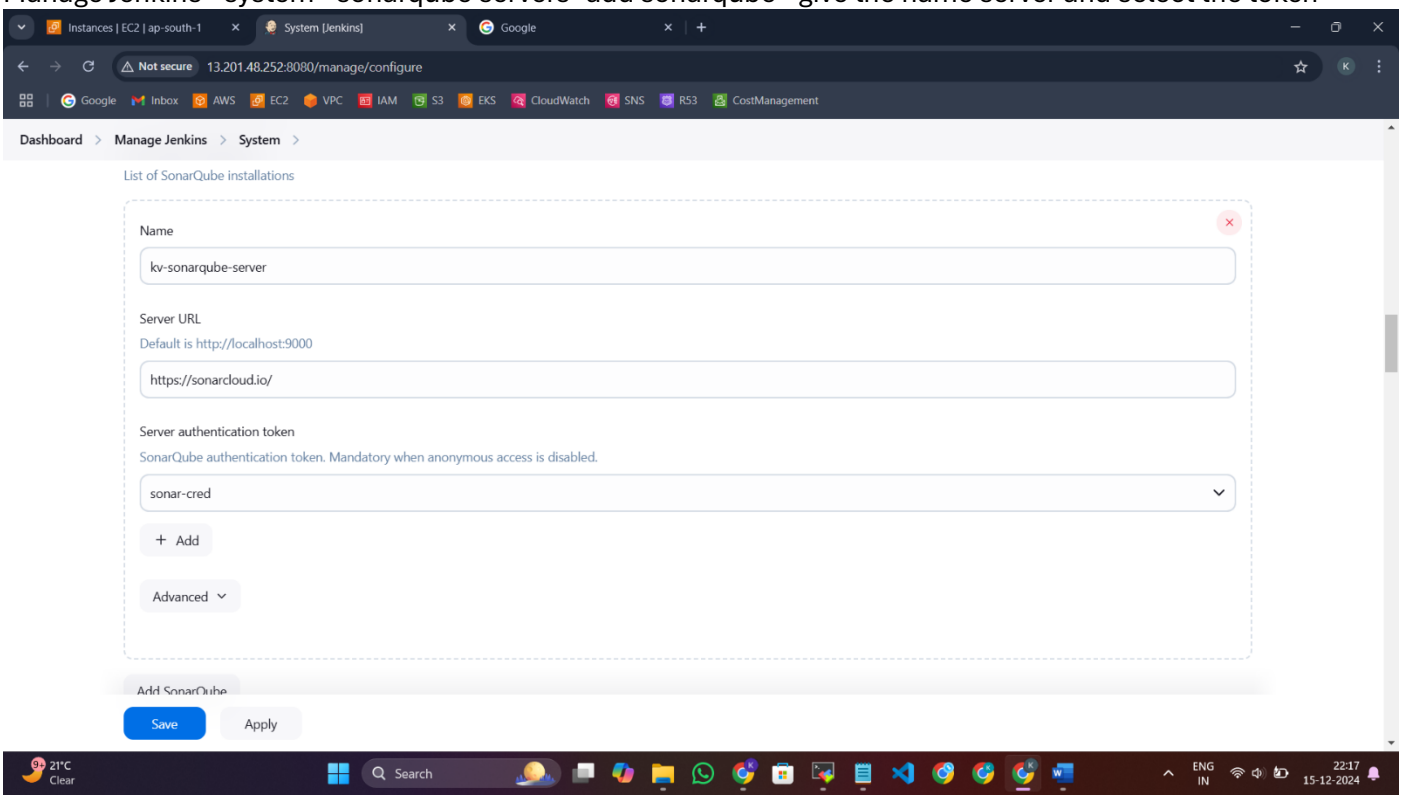


Aad these token in Jenkins credentials

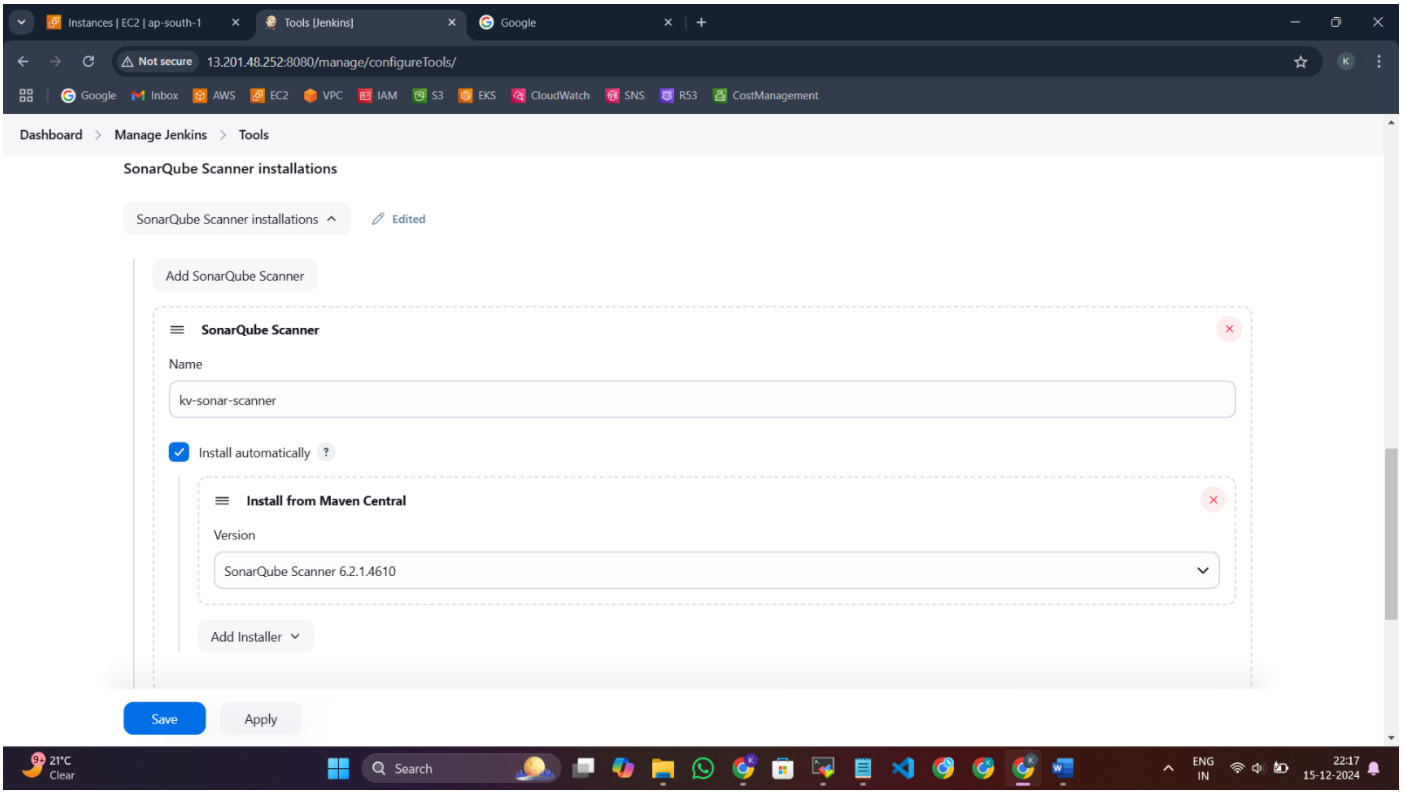


Connecting Jenkins with sq

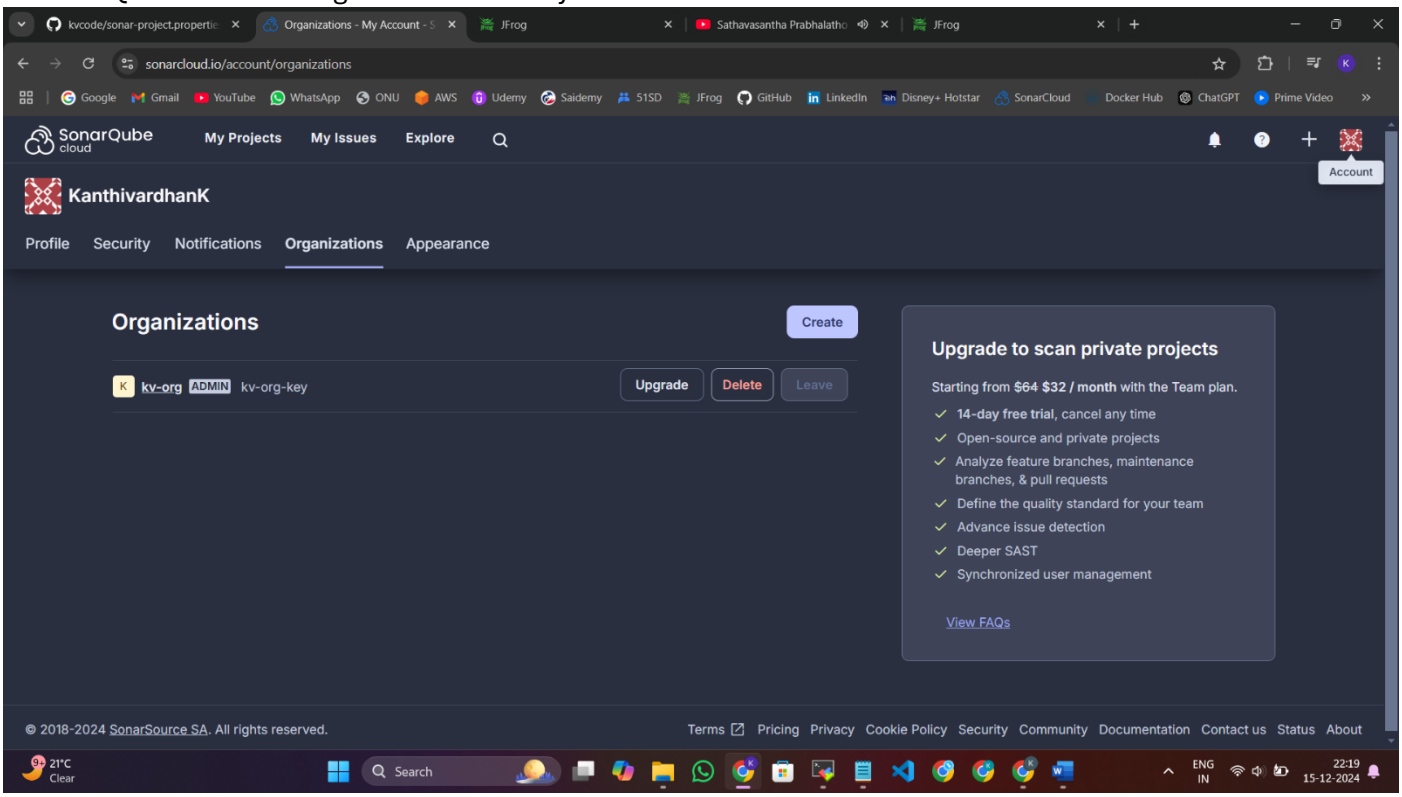
Manage Jenkins> system> sonarqube servers>add sonarqube> give the name server and select the token



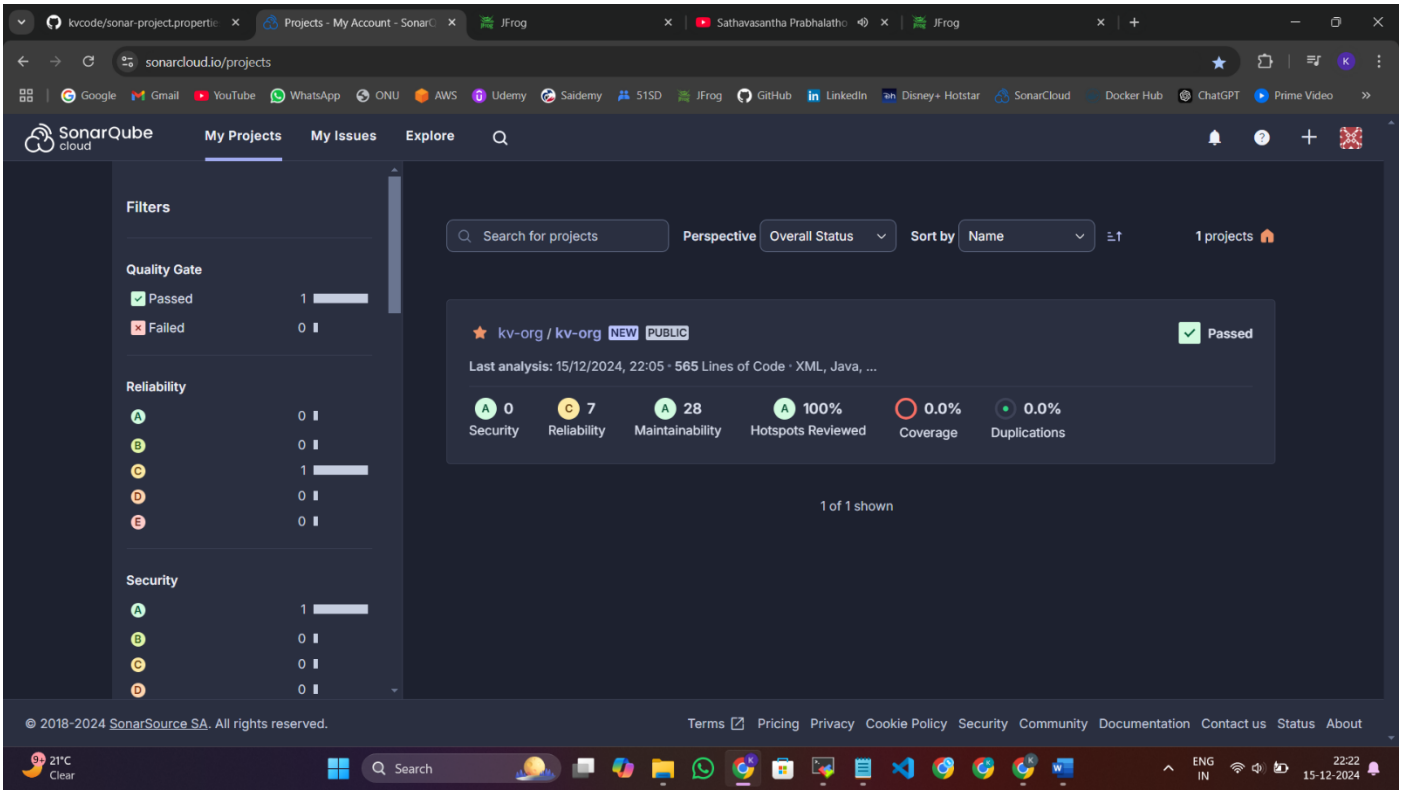
Manage Jenkins tools>sonarqube scanner installation> add ssqscanner



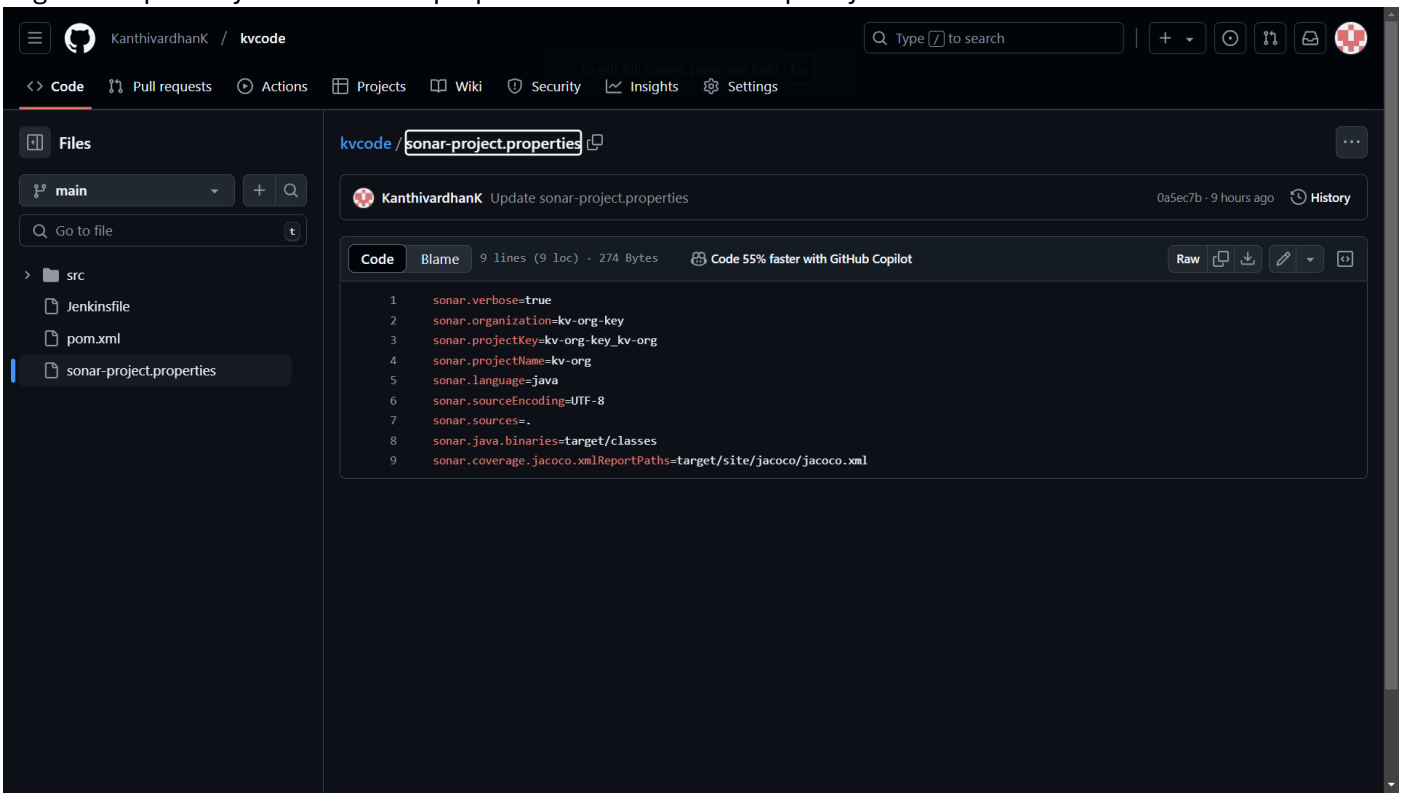
In SonarQube create an organization with key



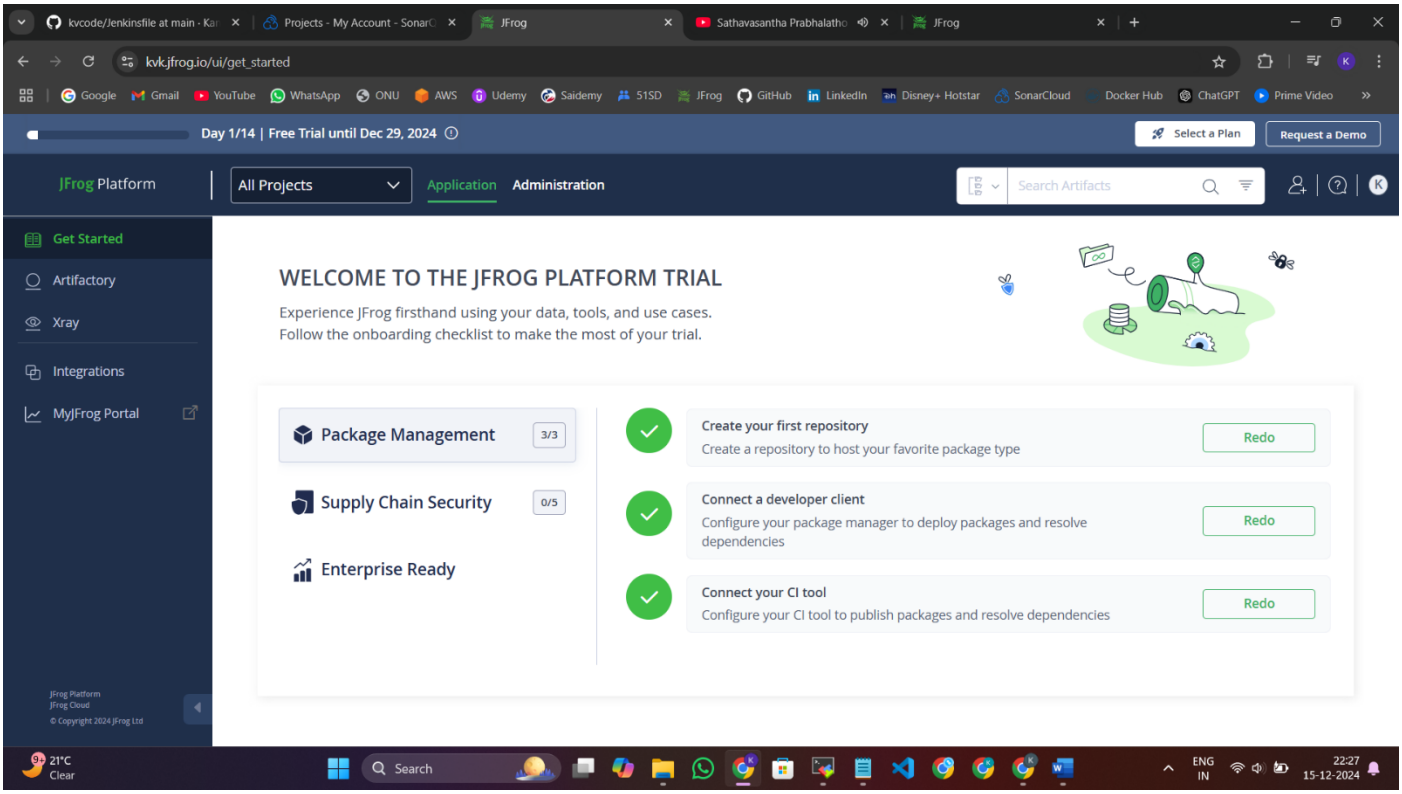
Create a project in that organization



In github repository it should have properties file to check the quality of code

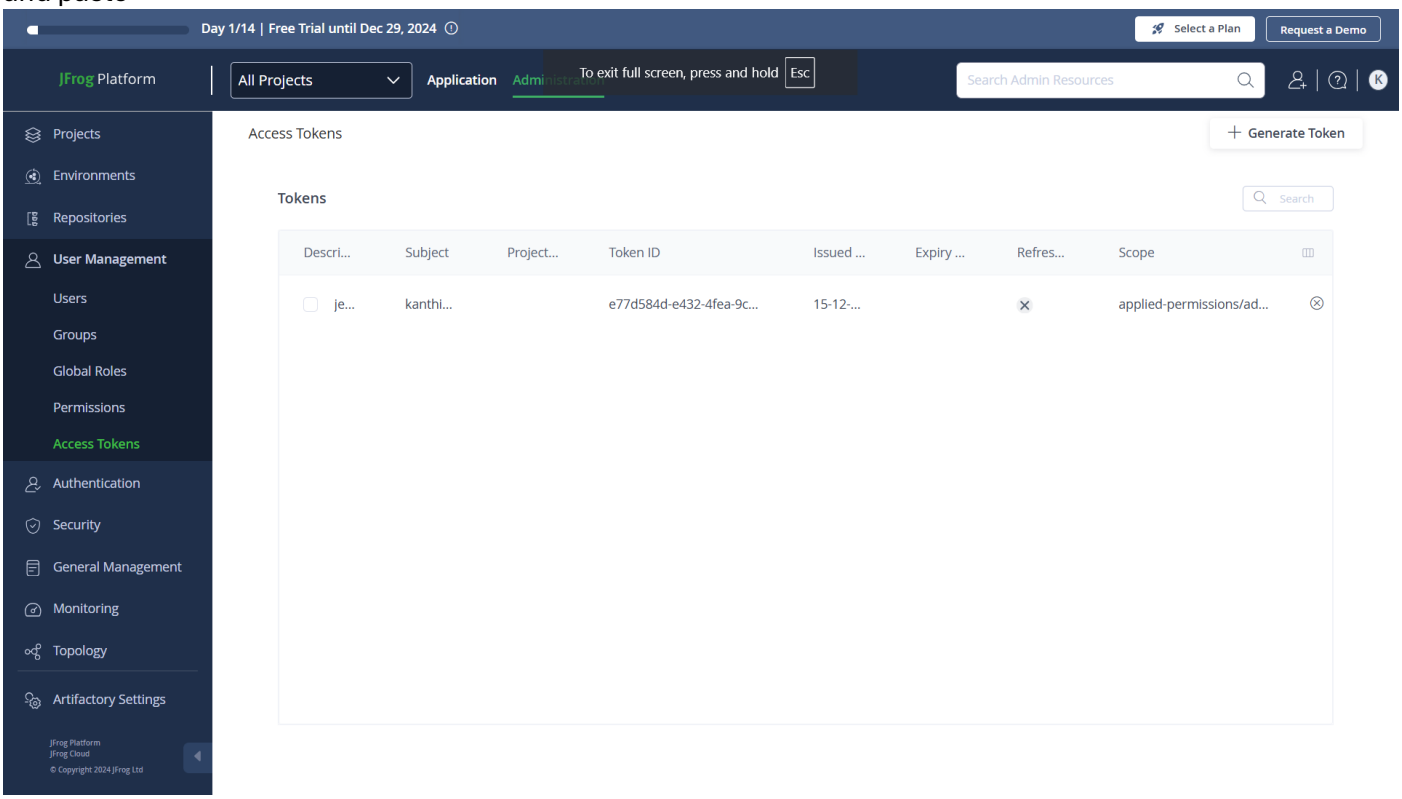


Create jfrog account

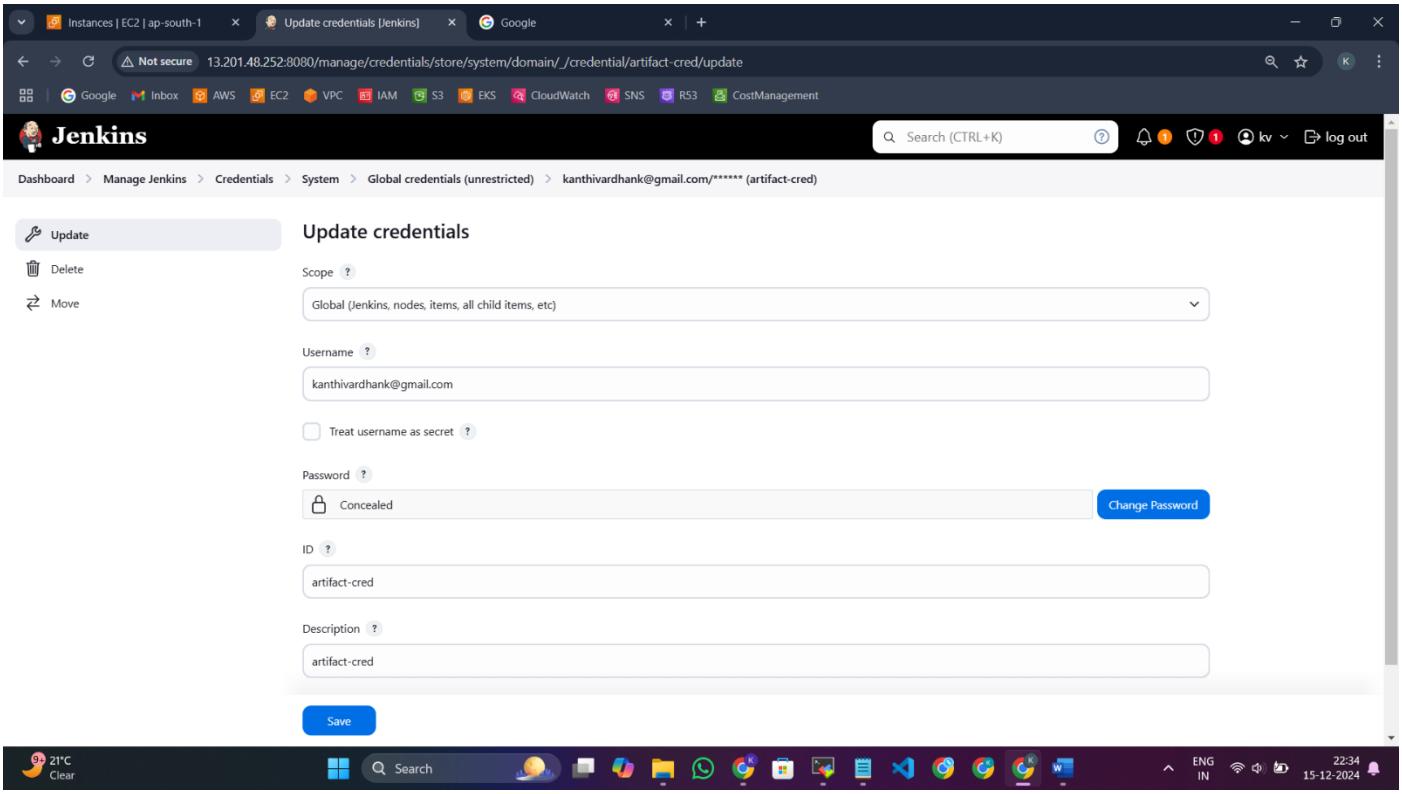


Create a maven repository in jfrog by default local,virtual and remote repository will get created
We store end product in local repo

Connecting jfrog with Jenkins
Administration>user accoubt> access token generate token> scoped token-username- generate-copy token and paste

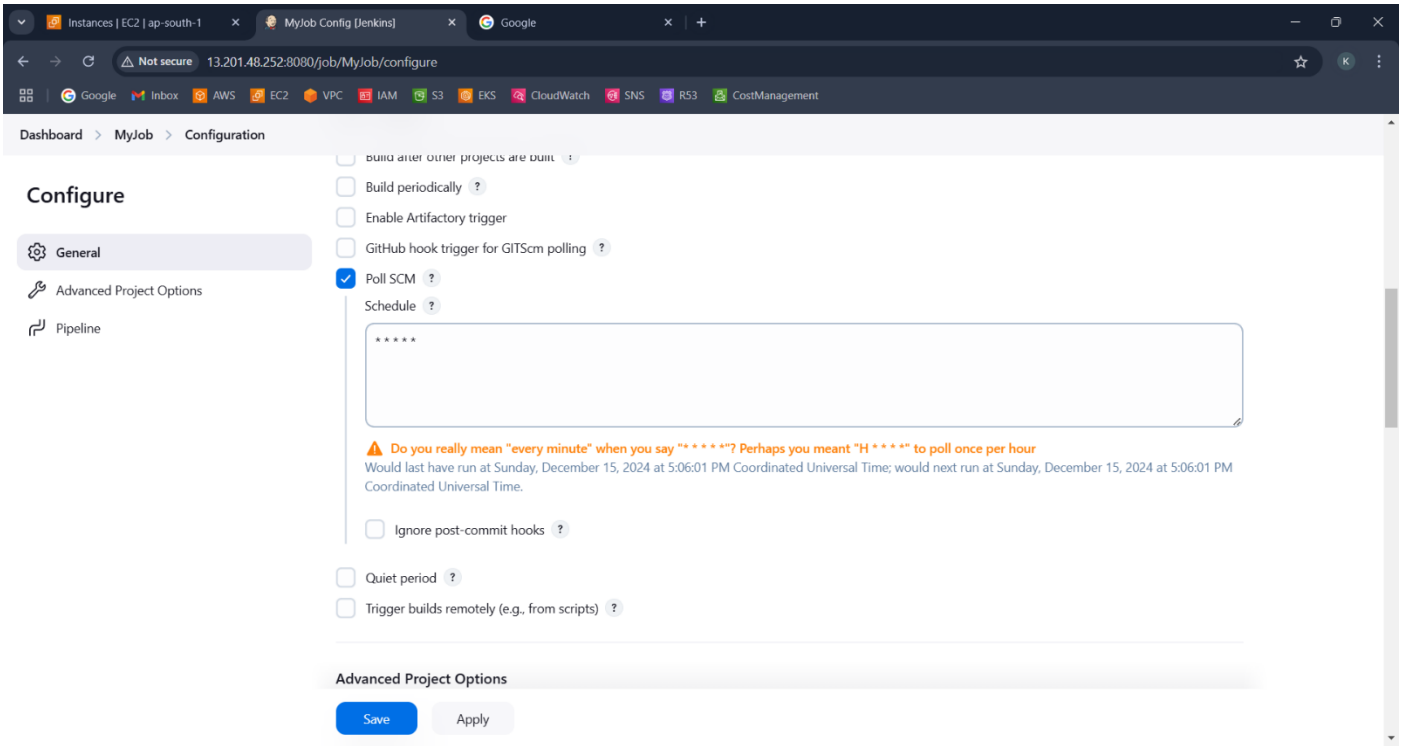


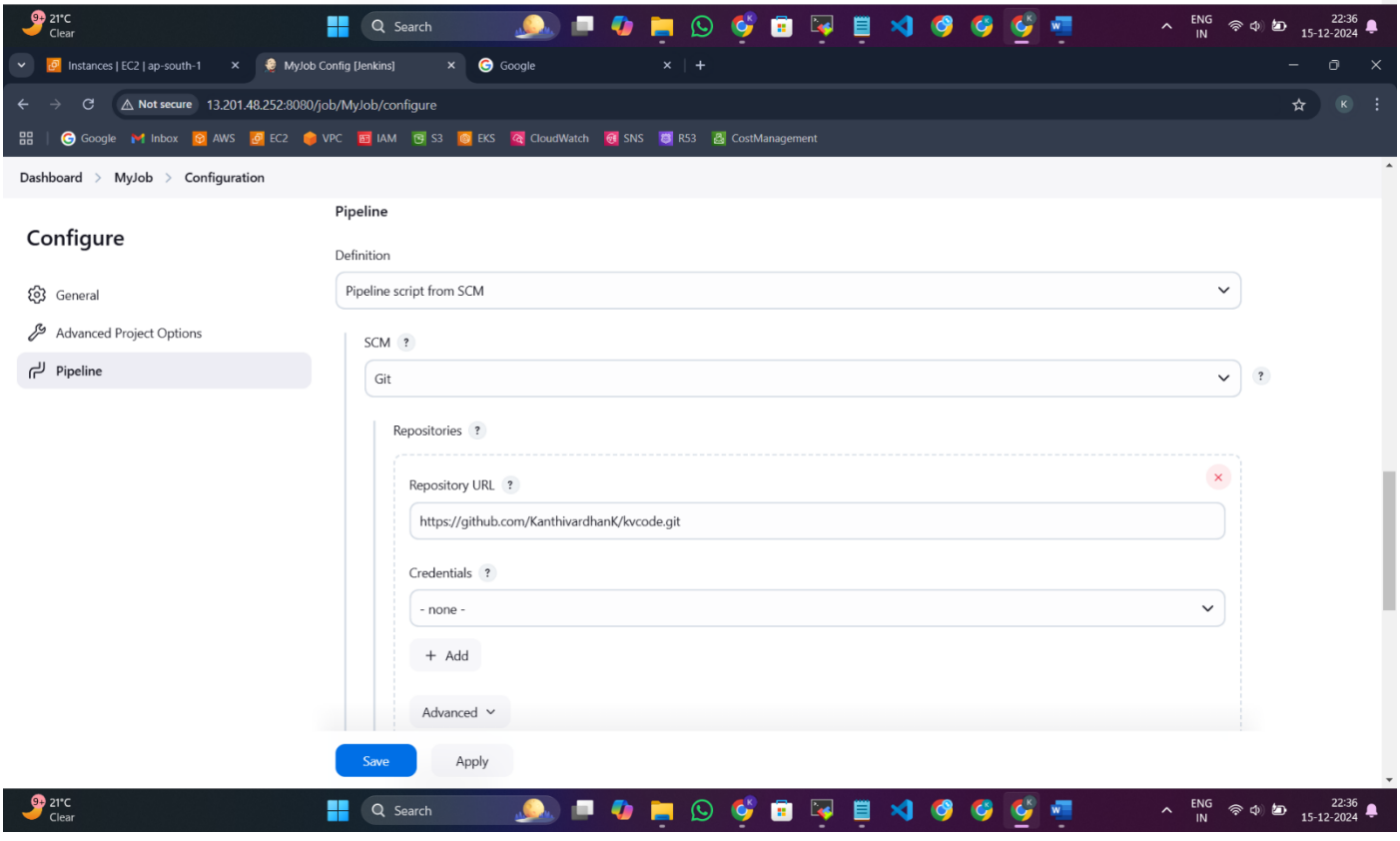
Adding these credentials in Jenkins
Jenkins>manage Jenkins> credentials

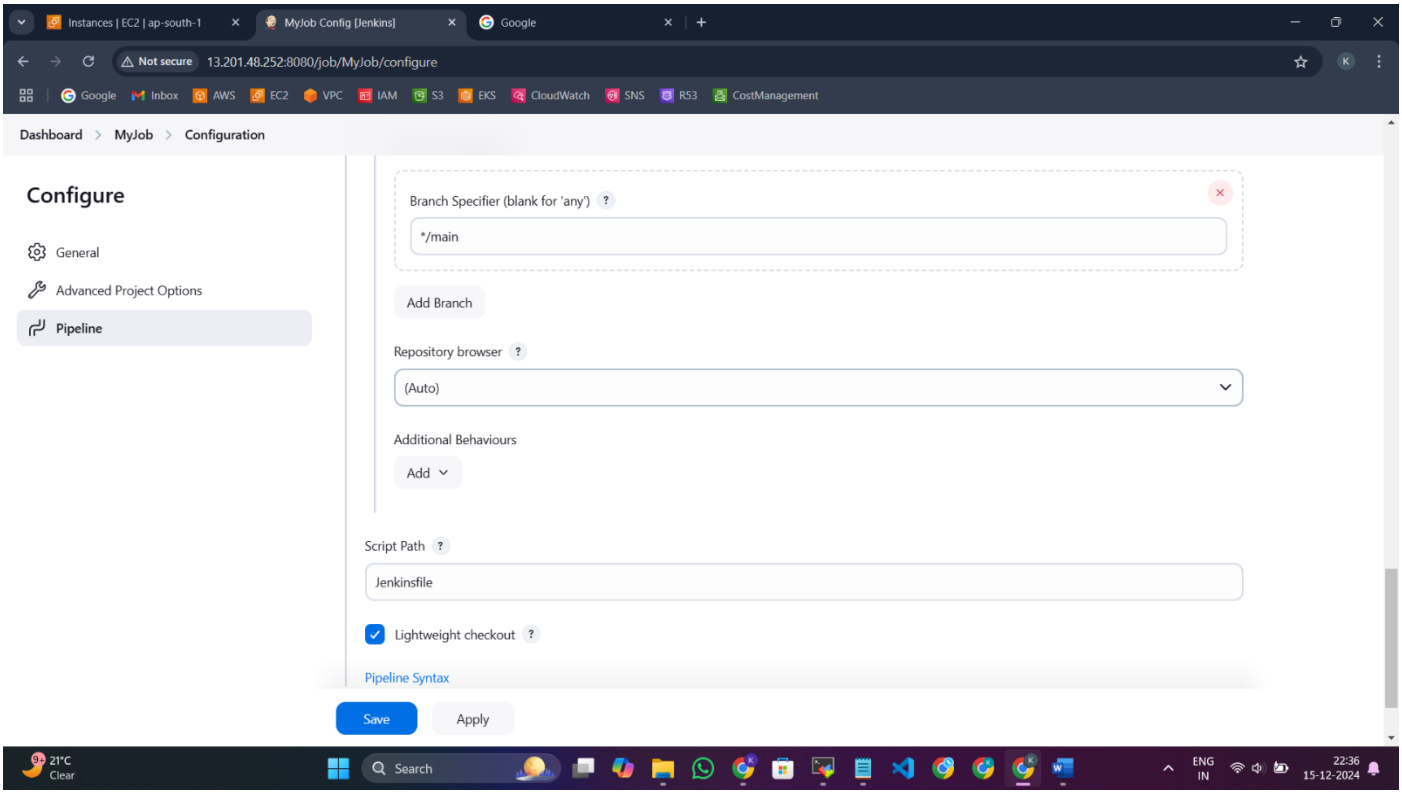


Add jfrog related script in jenkinsfiles in github repo

Create a pipeline job in Jenkins
Give poll scm * * * *
Select pipeline script from scm , give git repo url and branch


Instances | EC2 | ap-south-1
MyJob Config [Jenkins]
Google
Not secure
13.201.48.252:8080/job/MyJob/configure
Google Inbox AWS EC2 VPC IAM S3 EKS CloudWatch SNS R53 CostManagement
Dashboard > MyJob > Configuration
Configure
General
Advanced Project Options
Pipeline
Build periodically
Enable Artifactory trigger
GitHub hook trigger for GITScm polling
Poll SCM
Schedule
* * * * *
Do you really mean "every minute" when you say "* * * * *"? Perhaps you meant "H * * * *" to poll once per hour
Would last have run at Sunday, December 15, 2024 at 5:06:01 PM Coordinated Universal Time; would next run at Sunday, December 15, 2024 at 5:06:01 PM Coordinated Universal Time.
Ignore post-commit hooks
Quiet period
Trigger builds remotely (e.g., from scripts)
Advanced Project Options
Save
Apply


21°C Clear
Search
ENG IN
22:36 15-12-2024
Instances | EC2 | ap-south-1
MyJob Config [Jenkins]
Google
Not secure
13.201.48.252:8080/job/MyJob/configure
Google Inbox AWS EC2 VPC IAM S3 EKS CloudWatch SNS R53 CostManagement
Dashboard > MyJob > Configuration
Configure
General
Advanced Project Options
Pipeline
Pipeline
Definition
Pipeline script from SCM
SCM
Git
Repositories
Repository URL
https://github.com/KanthivardhanK/kvcode.git
Credentials
- none -
+ Add
Advanced
Save
Apply
21°C Clear
Search
ENG IN
22:36 15-12-2024



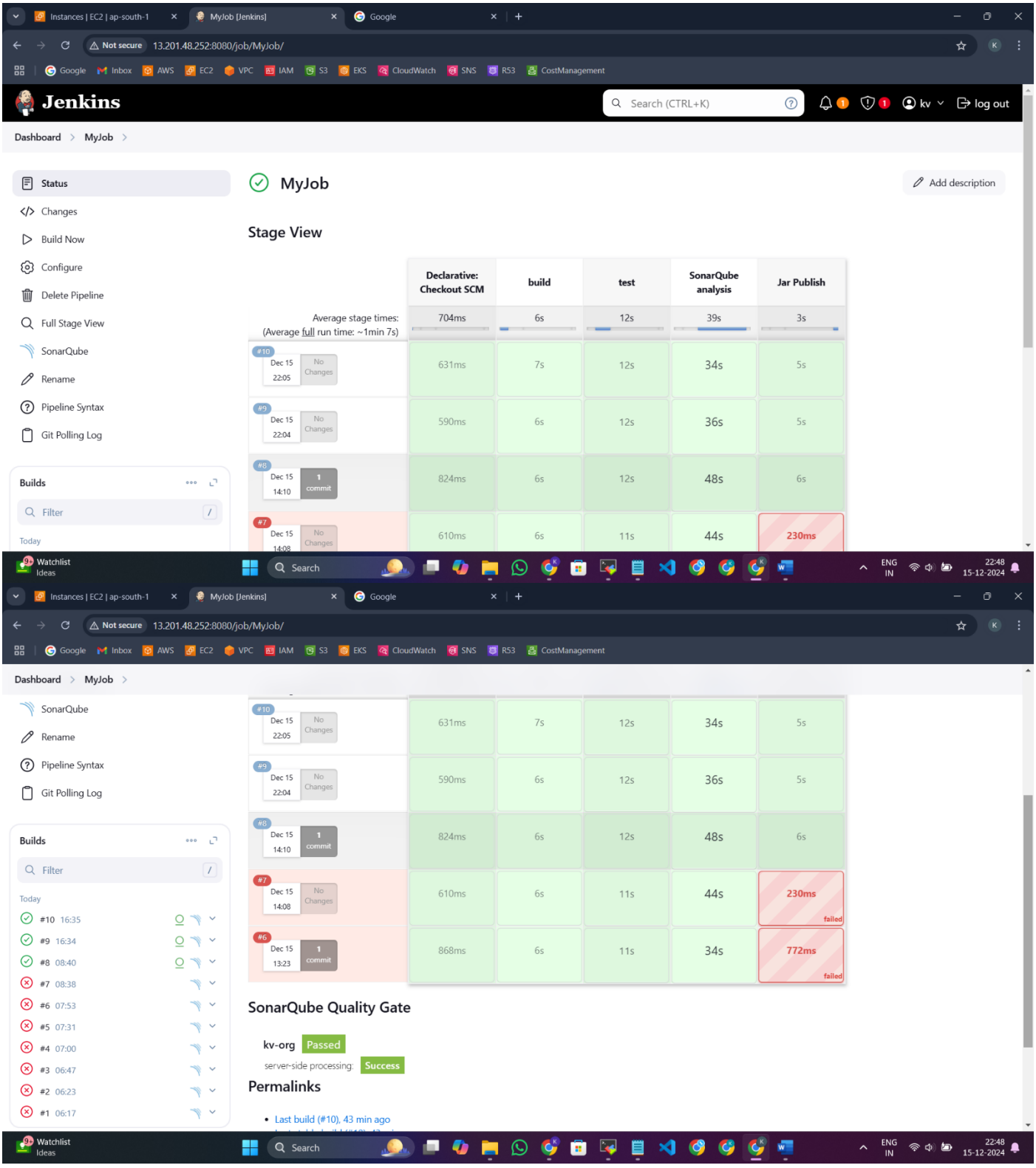
Jenkinsfile

```
// Define the URL of the Artifactory registry
def registry = 'https://kvk.jfrog.io'

pipeline {                          // 1  // Defines the start of the Jenkins pipeline block

  agent any                         // Specifies the pipeline can run on any available agent

  environment {                     // 2  // Defines environment variables for the pipeline
    PATH = "/opt/maven/bin:$PATH"       // Adds Maven's path to the system's PATH variable
  }                                 // 2  // Ends the environment block

  stages {                          // 3  // Defines the stages block where multiple stages are declared

    stage("build") {                // 4  // Creates a stage named 'build'
      steps {                       // 5  // Defines the steps that will be executed in this stage
        echo "----------- build started ----------"
                                    // Logs a message indicating the start of the build
        sh 'mvn clean deploy -Dmaven.test.skip=true'
                                    // Runs Maven clean and deploy commands, skipping tests
        echo "----------- build completed ----------"
                                    // Logs a message indicating the build completion
      }                             // 5  // Ends the steps block for 'build' stage
    }                               // 4  // Ends the 'build' stage

    stage("test") {                 // 6  // Creates a stage named 'test'
      steps {                       // 7  // Defines the steps that will be executed in this stage
        echo "----------- unit test started ----------"
                                    // Logs a message indicating the start of unit tests
        sh 'mvn surefire-report:report'
                                    // Runs the Maven Surefire report to execute unit tests
        echo "----------- unit test completed ----------"
                                    // Logs a message indicating unit test completion
      }                             // 7  // Ends the steps block for 'test' stage
```

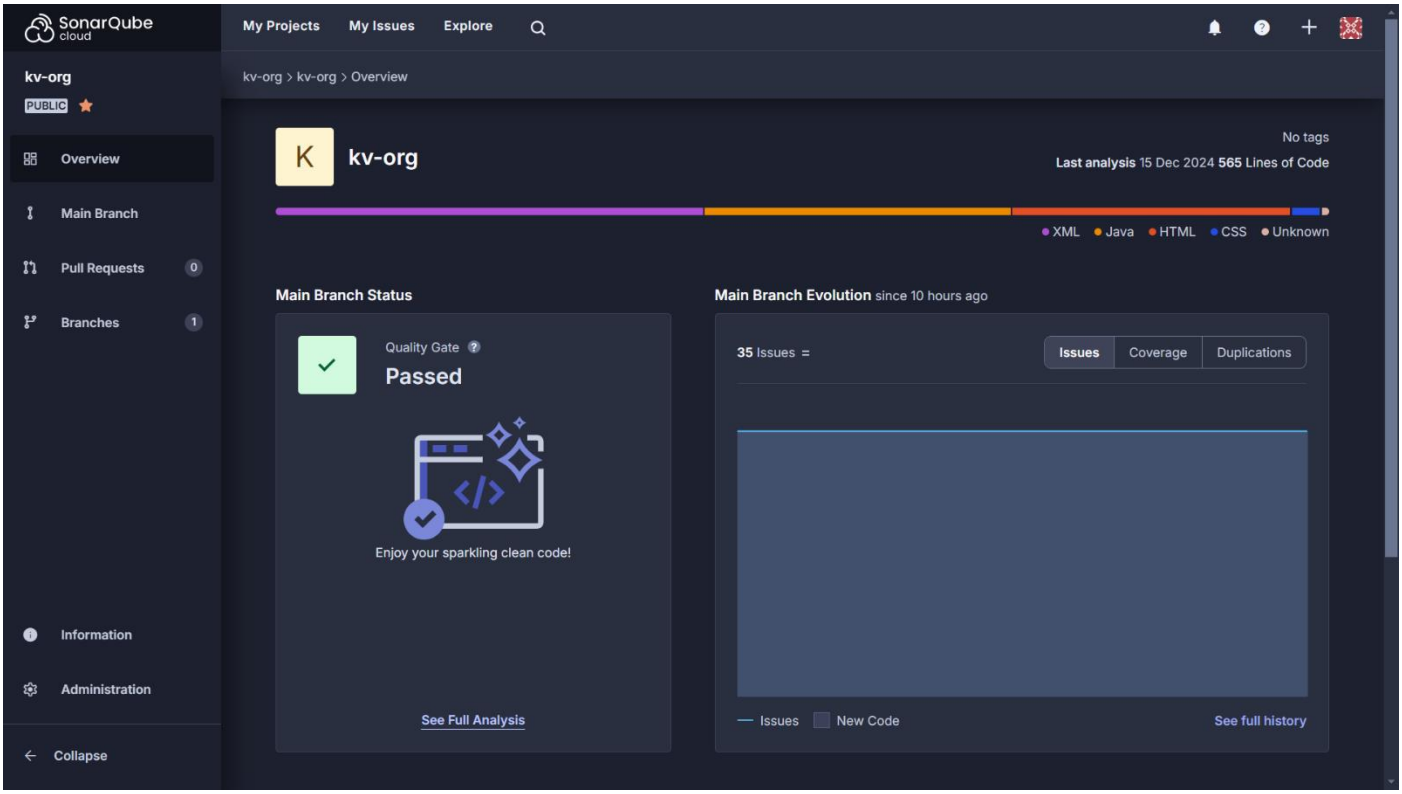
```
}                       // 6  // Ends the 'test' stage

stage('SonarQube analysis') {       // 8  // Creates a stage named 'SonarQube analysis'
  environment {               // 9  // Defines environment variables specific to this stage
    scannerHome = tool 'kv-sonar-scanner'
                        // Sets the SonarQube scanner tool
  }                     // 9  // Ends the environment block for this stage

  steps {                   // 10  // Defines the steps that will be executed in this stage
    withSonarQubeEnv('kv-sonarqube-server') {
                        // Executes the SonarQube analysis within the SonarQube environment
      sh "${scannerHome}/bin/sonar-scanner"
                        // Runs the SonarQube scanner tool
    }                   // Ends the withSonarQubeEnv block
  }                     // 10  // Ends the steps block for 'SonarQube analysis' stage
}                       // 8  // Ends the 'SonarQube analysis' stage

//stage("Quality Gate") {           // 11  // Creates a stage named 'Quality Gate'
 //  steps {                  // 12  // Defines the steps that will be executed in this stage
  //   script {               // 13  // Allows running custom Groovy script inside the pipeline
   //    timeout(time: 1, unit: 'HOURS') {
    //                    // Sets a timeout of 1 hour for the quality gate check
     //     def qg = waitForQualityGate()
      //                  // Waits for the quality gate result from SonarQube
        //if (qg.status != 'OK') {
         //                // Checks if the quality gate status is not OK
        //      error "Pipeline aborted due to quality gate failure: ${qg.status}"
       //                 // Aborts the pipeline if the quality gate fails
     //      }
    //     }
  //   }                    // 13  // Ends the script block for the Quality Gate stage
 //  }                      // 12  // Ends the steps block for 'Quality Gate' stage
//}                        // 11  // Ends the 'Quality Gate' stage

stage("Jar Publish") {            // 14  // Creates a stage named 'Jar Publish'
  steps {                   // 15  // Defines the steps that will be executed in this stage
    script {                // 16  // Allows running custom Groovy script inside the pipeline
      echo '<--------------- Jar Publish Started --------------->'
                        // Logs a message indicating the start of JAR publishing
      def server = Artifactory.newServer url: registry + "/artifactory", credentialsId: "artifact-cred"
                        // Defines the Artifactory server with the specified URL and credentials
      def properties = "buildid=${env.BUILD_ID},commitid=${GIT_COMMIT}"
                        // Sets properties like build ID and Git commit ID for the build
      def uploadSpec = """{
          "files": [
           {
            "pattern": "jarstaging/(*)",
            "target": "kv-libs-release-local/{1}",
            "flat": "false",
            "props": "${properties}",
            "exclusions": [ "*.sha1", "*.md5"]
           }
          ]
       }"""
                        // Defines the upload specification for uploading JAR files to Artifactory
      def buildInfo = server.upload(uploadSpec)
                        // Uploads the files to Artifactory and collects build info
```

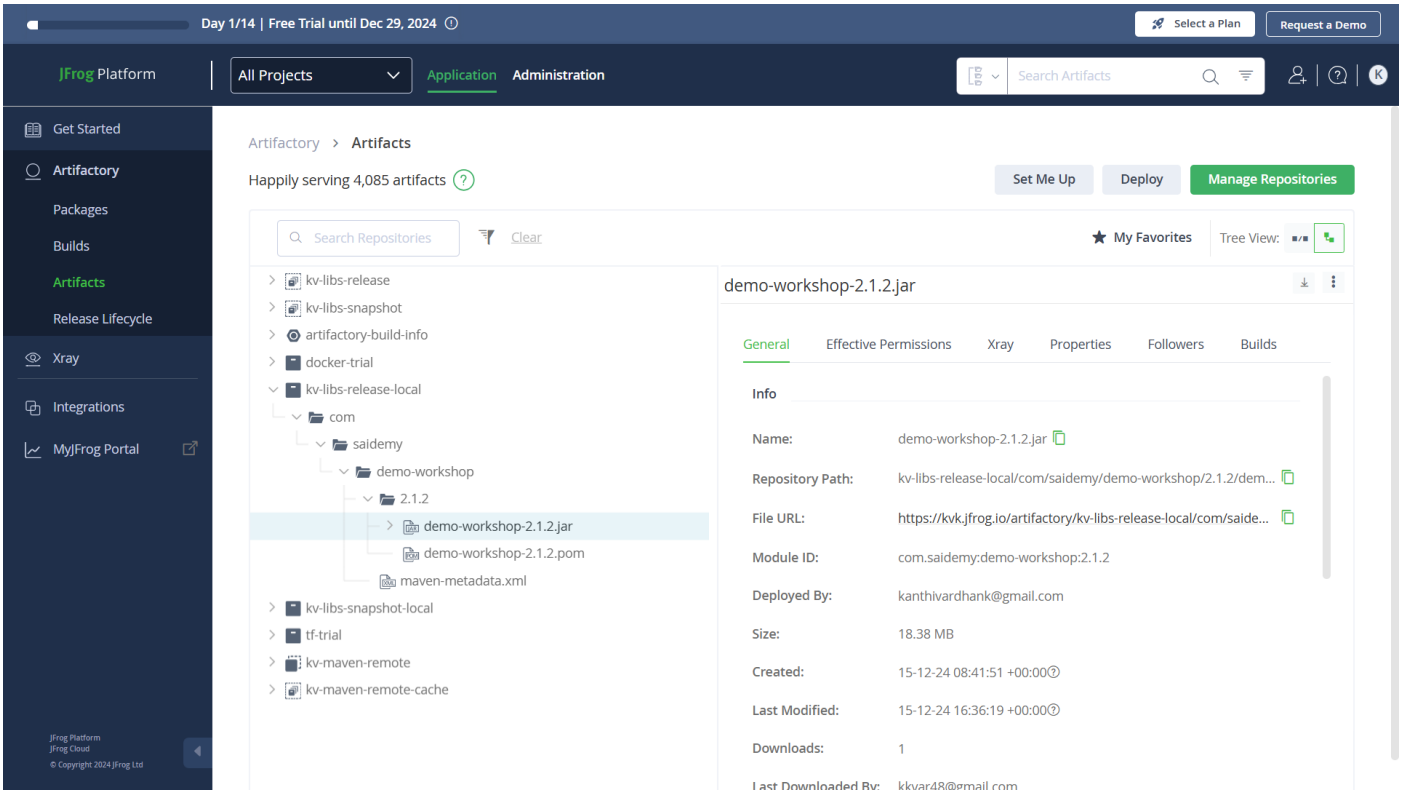
```
            buildInfo.env.collect()
                        // Collects environment variables as part of the build info
            server.publishBuildInfo(buildInfo)
                        // Publishes the build information to Artifactory
            echo '<--------------- Jar Publish Ended --------------->'
                        // Logs a message indicating the end of JAR publishing
          }             // 16  // Ends the script block for 'Jar Publish' stage
        }               // 15  // Ends the steps block for 'Jar Publish' stage
      }                 // 14  // Ends the 'Jar Publish' stage

    }                   // 3  // Ends the stages block
}
```

Now trigger the job

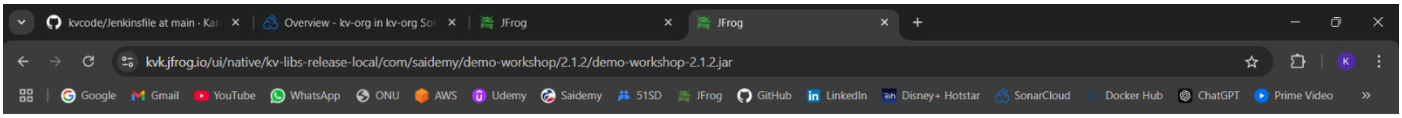


If the code satisfies the quality gate conditions the code will be stored in jfrog and pipeline continues





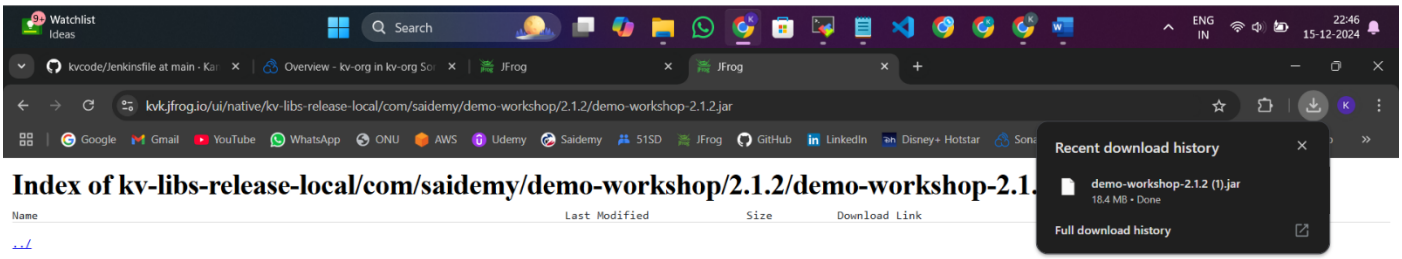
To share the artifacts with others like client
Copy the file url show in the above image
And share it. When they paste in browser automatically the end product (war/jar) file will be downloaded in their local machine



Index of kv-libs-release-local/com/saidemy/demo-workshop/2.1.2/demo-workshop-2.1.2.jar

Name	Last Modified	Size	Download Link

../



Index of kv-libs-release-local/com/saidemy/demo-workshop/2.1.2/demo-workshop-2.1.

Name	Last Modified	Size	Download Link

../