Parameters in Pipelines

- Parameter is a user input while running the jenkins job Refer Here
- using parameter to pass the maven goal

```
pipeline {
  agent { label 'spc' }
  parameters {
    choice(name: 'MAVEN_GOAL', choices: ['package', 'clean package'], description:
'This is maven goal')
  }
  stages {
    stage('git') {
          steps {
            git url: 'https://github.com/spring-projects/spring-petclinic.git',
branch: 'main'
          }
    }
    stage('build') {
      steps {
        mail bcc: '', body: 'Build started', cc: '', from: '', replyTo: '',
subject: 'Build started', to: 'all@learnigthoughts.io'
        sh "mvn ${params.MAVEN_GOAL}"
        junit testResults: '**/surefire-reports/*.xml'
        archive '**/target/spring-petclinic-*.jar'
        mail bcc: '', body: 'Build completed', cc: '', from: '', replyTo: '',
subject: 'Build completed', to: 'all@learnigthoughts.io'
      }
    }

  }

}
```

Environmental variables

- Jenkins injects environmental variables before the job is executed and these variables will contain information about
  - build id
  - Git commit details
  - location details
- Lets make the emails more elaborate by changing the pipeline

```
pipeline {
  agent { label 'spc' }
  parameters {
    choice(name: 'MAVEN_GOAL', choices: ['package', 'clean package'], description:
'This is maven goal')
```

```
  }
  stages {
    stage('git') {
          steps {
            git url: 'https://github.com/spring-projects/spring-petclinic.git',
branch: 'main'
          }
    }
    stage('build') {
      steps {
        mail bcc: '', body: 'Build started', cc: '', from: '', replyTo: '',
subject: 'Build started', to: 'all@learnigthoughts.io'
        sh "mvn ${params.MAVEN_GOAL}"
        junit testResults: '**/surefire-reports/*.xml'
        archive '**/target/spring-petclinic-*.jar'
        mail bcc: '', body: 'Build completed', cc: '', from: '', replyTo: '',
subject: 'Build completed', to: 'all@learnigthoughts.io'
      }
      post {
        failure {
            mail bcc: 'all@learningthoughts.io',
                 from: 'jenkins@learningthouths.io',
                 to: "dev@learningthoughs.io",
                 subject: "Build of ${JOB_BASE_NAME} with Build Id ${BUILD_ID} is
failed",
                 body: "Refer to ${RUN_DISPLAY_URL} for more info"

        }
        success {
            mail bcc: 'all@learningthoughts.io',
                 from: 'jenkins@learningthouths.io',
                 to: "dev@learningthoughs.io",
                 subject: "Build of ${JOB_BASE_NAME} with Build Id ${BUILD_ID} is
success",
                 body: "Refer to ${RUN_DISPLAY_URL} for more info"
        }
      }
    }

  }

}
```

Static Code Analysis

- Static code analysis tools help in ensuring
  - Best Practices are following
  - Code Coverage:
    - line coverage
    - symbol coverage

        - branch coverage
    - Code Analysis (Technical debt)
        - Errors
        - Warnings
        - Critical
        - Info
    - Security Scanning
- Generally organizations create Quality Gates. This defines the acceptance criteria for code to be allowed to be considered
    - Code coverage
    - Code Analysis issues (Technical debt)
- We will be using sonarqube Refer Here
- Lets create a Sonar Cloud account Refer Here
- Jfrog: for artifact repository Refer Here