I can provide you with a general outline for a Jenkins CI/CD pipeline that incorporates automation for building, testing, and deploying using Git, Maven, SonarQube, Artifactory, Docker, and Kubernetes

1.Setting up the environment:

- **Install and configure Jenkins on a server or a virtual machine.**
- **Install and configure Git, Maven, SonarQube, Artifactory, Docker, and Kubernetes on the Jenkins server.**

2.Creating a pipeline:

- **In Jenkins, create a new pipeline project and specify the pipeline script location (e.g., from a Git repository).**
- **Define the stages for the pipeline:**
- **Build: This stage compiles the code and creates a build artifact using Maven.**
- **Test: This stage runs automated tests to verify that the build artifact works correctly.**
- **Static Code Analysis: This stage uses SonarQube to analyze the code for bugs, vulnerabilities, and code smells.**
- **Package: This stage creates a Docker image from the build artifact and pushes it to the Artifactory repository.**
- **Deploy: This stage deploys the Docker image to a Kubernetes cluster.**

3.Defining the pipeline script:

- **Define the pipeline script that implements the stages defined in step 2.**
- **Use Jenkins plugins to integrate with Git, Maven, SonarQube, Artifactory, Docker, and Kubernetes.**
- **Use conditional statements to handle errors and failures in each stage.**
- **Use environment variables to store configuration information (e.g., Git repository URL, Artifactory credentials).**

4.Running the pipeline:

- **Trigger the pipeline to run automatically whenever code changes are pushed to the Git repository.**
- **View the pipeline logs and status to monitor the progress of the pipeline.**
- **Use email notifications or other alerts to notify the team of pipeline failures or errors.**

This is a high-level overview of a Jenkins CI/CD pipeline that incorporates automation for building, testing, and deploying using Git, Maven, SonarQube, Artifactory, Docker, and Kubernetes. The specifics of the pipeline will depend on the details of your project and the tools and technologies you are using.

```
pipeline {

  agent any


  stages {

    stage('Checkout') {

      steps {
        git 'https://github.com/your-repo.git'
    }
}
stage('Build') {
    steps {
        sh 'mvn clean package -DskipTests'
    }
}
stage('Static Code Analysis') {
    steps {
        withSonarQubeEnv('SonarQube') {
            sh 'mvn sonar:sonar'
        }
    }
}
stage('Deploy to Artifactory') {
    steps {
        script {
            def server = Artifactory.server 'Artifactory'
            def buildInfo = Artifactory.newBuildInfo()
            def rtMaven = Artifactory.newMavenBuild()
```

```
          rtMaven.deployer releaseRepo: 'my-release-repo', snapshotRepo: 'my-snapshot-repo', server: server

          rtMaven.resolver releaseRepo: 'my-release-repo', snapshotRepo: 'my-snapshot-repo', server: server

          buildInfo.env.capture = true

          buildInfo.env['BUILD_NUMBER'] = env.BUILD_NUMBER

          buildInfo.env['GIT_COMMIT'] = env.GIT_COMMIT

          buildInfo.env['BUILD_URL'] = env.BUILD_URL

          rtMaven.run pom: 'pom.xml', goals: 'clean deploy -DskipTests', buildInfo: buildInfo

        }

      }

    }

    stage('Build Docker Image') {

      steps {

        sh 'docker build -t my-image .'

      }

    }

    stage('Push Docker Image to Registry') {

      steps {

        withCredentials([usernamePassword(credentialsId: 'docker-registry-credentials', usernameVariable: 'DOCKER_REGISTRY_USERNAME', passwordVariable: 'DOCKER_REGISTRY_PASSWORD')]) {
          sh "docker login -u $DOCKER_REGISTRY_USERNAME -p
$DOCKER_REGISTRY_PASSWORD"
        }
        sh 'docker push my-image'
      }
    }
    stage('Deploy to Kubernetes') {
      steps {
        withKubeConfig([credentialsId: 'kubeconfig-credentials', serverUrl:
'https://kube-api-url']) {
          sh 'kubectl apply -f deployment.yaml'
        }
      }
    }
  }
}
```

In this pipeline:

- The first stage checks out the source code from the Git repository.
- The second stage builds the code using Maven.
- The third stage performs static code analysis using SonarQube.

- **The fourth stage deploys the built artifacts to Artifactory.**
- **The fifth stage builds a Docker image using the Dockerfile in the source code repository.**
- **The sixth stage pushes the Docker image to a Docker registry.**
- **The final stage deploys the application to Kubernetes using a deployment manifest.**

Note that this pipeline assumes you have set up the necessary credentials in Jenkins for Artifactory, Docker registry, and Kubernetes. You will need to replace the placeholders in the pipeline with your actual credentials and configuration information.