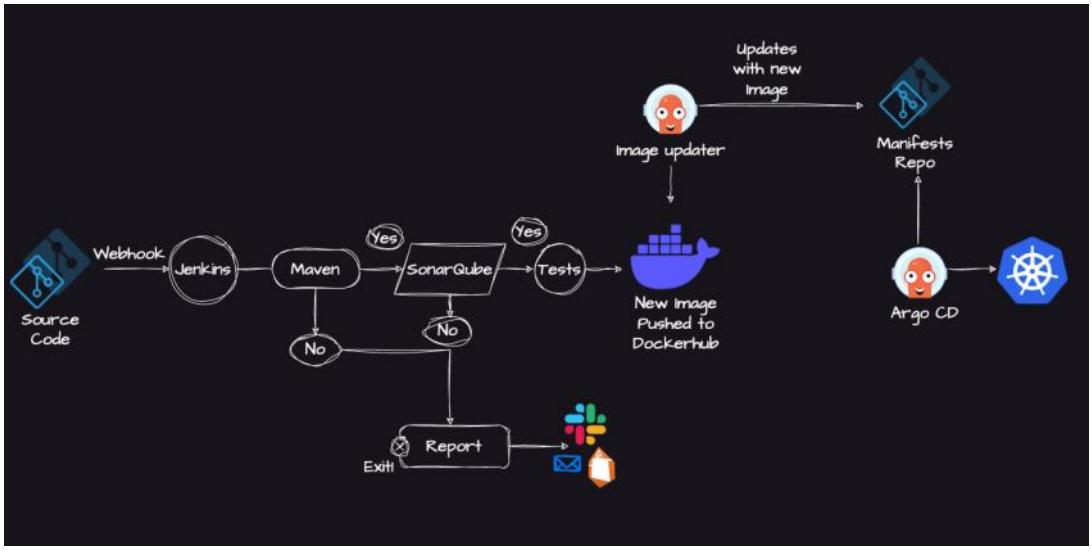
Ultimate CICD Pipeline Explanation of Flow

23 October 2024 11:36



Jenkins Pipeline only doing the continous integration here till New image pushed to dockerhub as after that there is pull mechanism and after part is continous delivery.

CI - Ensures that build is smooth , all test are executed , code quality is maintained and image is created things like these.

CD - Ensures that deployment and delivery process is done.

Jenkins watch the git repository as whenever any changes is occurs jenkins pipeline has to be triggered.

Best way to trigger jenkins by knowing that a change occurred is webhook. Infact github informs the jenkins that a change occurs using webhook.

We can go to jenkins and get the webhook url and put it into the settings and where you can define for which actions webhooks has to be triggered like on commit pull request and on issues.

As first git hub will trigger the change to the webhook and it will be pulled by jenkins then jenkins using maven will build the application.
To configure the maven plugin we go to jenkins and go to configure system and install maven or the other thing we can use a docker agent directly so we don't have to do installation and configuration by ourself as there are already prepared docker agents are available.

As part of build unit test runs and if successful then can perfom static code analysis then on next stage if your build stage goes down or fails therefore in such cases you can configure some alerts using API's via plugins and then it goes to the next stage where we can integrate it with Sonar cube or any other code scanning tools or the security scanning tools so that we can verify what happed in your previous stage and check is it reached a specific threshold of our organisation then if there are no security vulnerabilities then we can proceed with the creation of docker image or we can create a docker plugin here to by writing a shell script then we will send that docker image to a docker registry.

Registry can be anything like docker hub , quay.io , ECR.

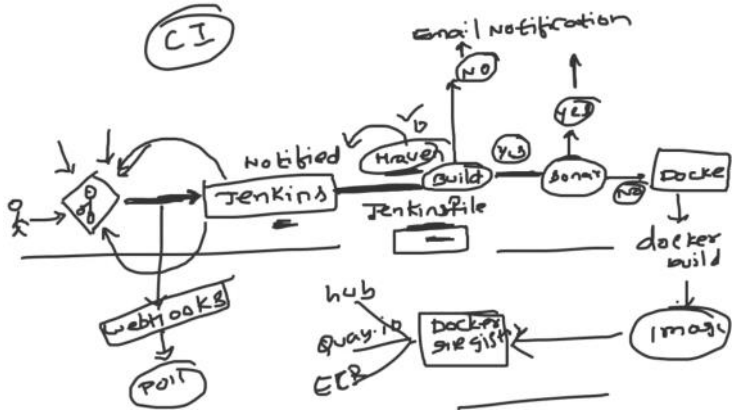
So this was the continousn integration process part.

Jenkins also can talk to jenkins using polling and can also configure some pull triggers into the jenkins.

Declarative pipelines are very easy to write instead of scripted pipelines.

In jenkins there are some plugins like - Email Notification, Slack Notification, AWS Email Notification.

Now in the end of CI you have a image ready with new tag and the image artifact with the new version will be pushed into the docker container registry.



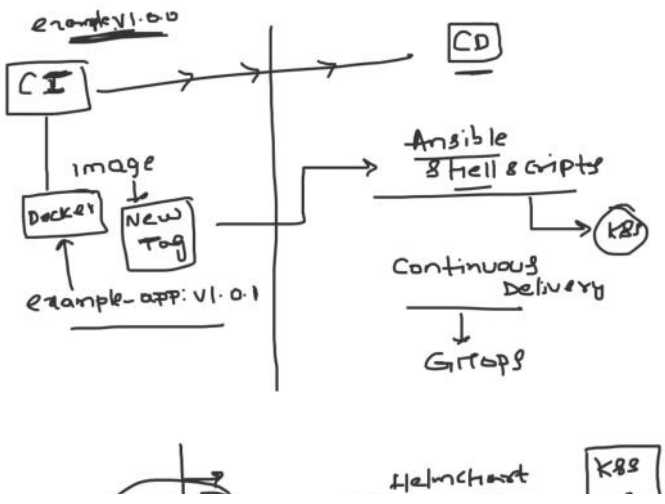
Now the CD part needs to start with continous delivery tools in which gitops based are best because what git ops says and practices is smaller to our source code.

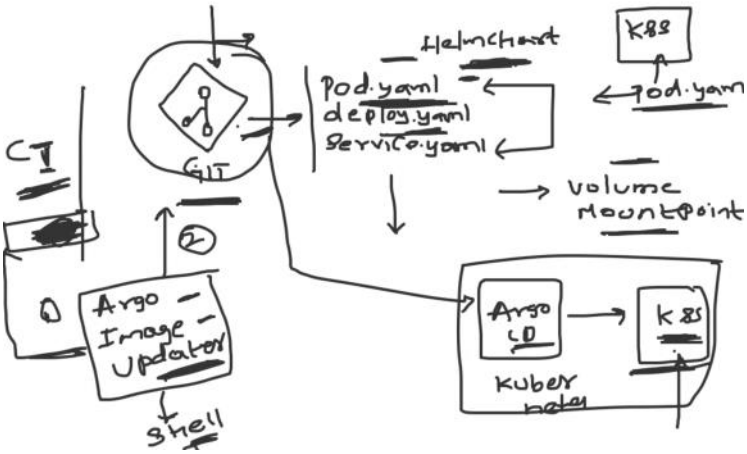
We should use git repository for application manifest too which contains files like pod.yaml , deploy.yaml and service.yaml and we use this so that not anyone without checking can enter into the cluster to update the pod.yaml.

Argo image updater we can use to know whenever an new image version is pushed to the registry it directly update the git repository and there Argo CD is contiously watching the git repositroy and whenever the change arrives then take the new helmchart , pod.yml , deploy.yml and service.yml to the kubernetes cluster.

As soon as the CI process completes and a new image of version is updated to the container registry then.

Conatiner Registry - (Application Management) Git Repository - Kubernetes cluster.

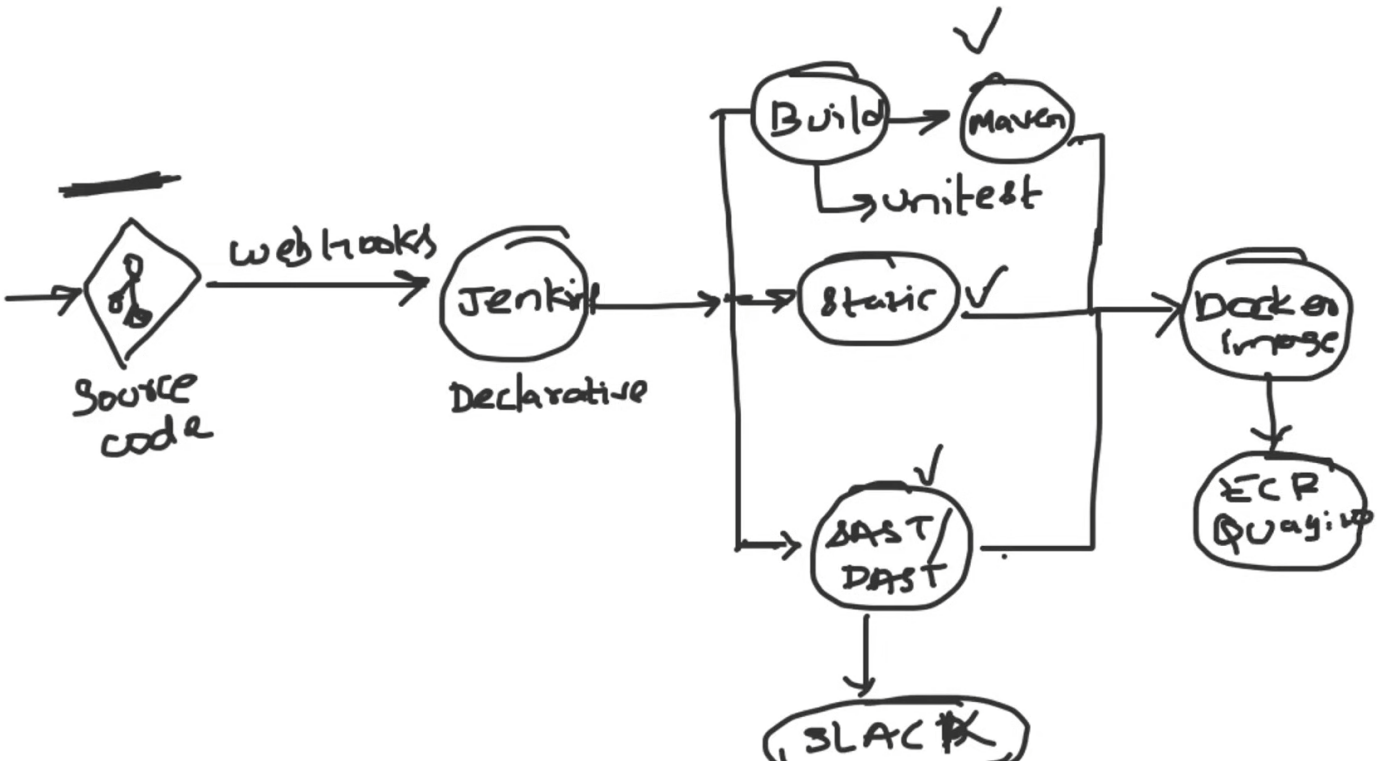




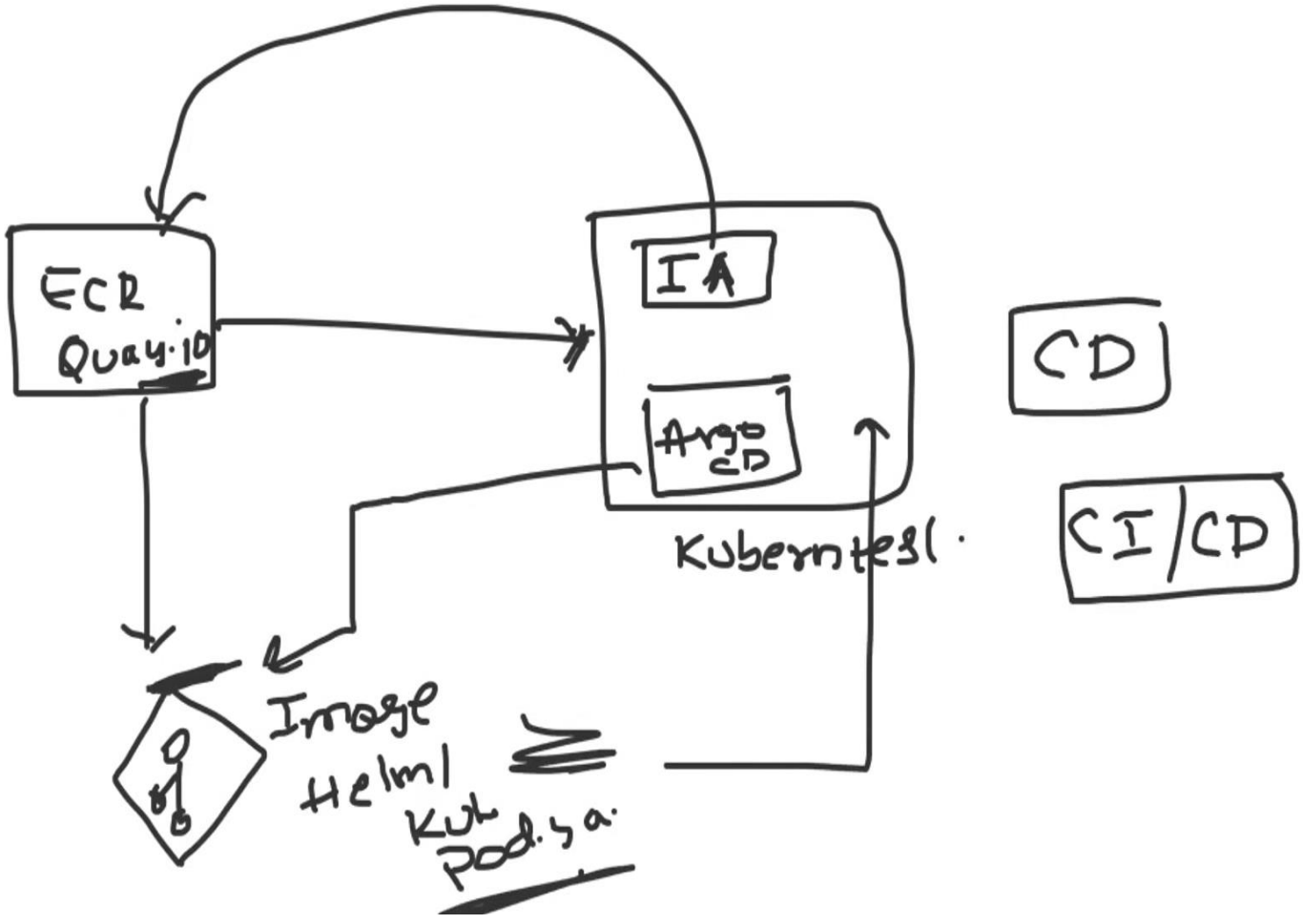
CICD COMPLETE EXPLANATION -

We have a git repository where we have application source code as soon as a developer raises a pull request to this git repository we have webhooks using it we triggers jenkins pipelines. We have used declarative jenkins pipeline because they are easy to write and collabrate and as part of this pipeline we run multiple stages some of the stages are first is build stage using it maven as our build language we build the application and if its successful with the unit test the next stage we would perform static code analysis using this we verify its not exposed to any static codes and after that we have slash and dast tools using these we verify security if it has any security vulnarabilites if it fails we failed notifications like slack , email if everything is good then we forward for create a docker image. We are running simple shell target to create a simple docker image out of the docker file which is stored in the git repository itself and when its create we pushed it to the conatiner registry like in ECR in AWS or docker hub. It completes CI process once it pushed to coantiner registry we have a kuberntes cluster in which we deploy two CD tools one is argo image updater and other is argo cd. First monitor the image registry as soon as a new image is created it will pick it in another git repository that we have which is purely for the image manifest that is helmchart , customize , Pod.yml, Deployment.yml so as soon as this git repository is updated with the new image than the other kubernetes controller we have which is argo cd it takes the new image and it deploys on the kubernetes cluster.

CI PROCESS -


Build
Maven
unitest
webhooks
Jenkins
Declarative
Source code
Static
Docker Image
ECR
Quay.io
SAST/
DAST
SLACK

CD PROCESS -


ECR
Quay.io
IA
Argo CD
CD
CI/CD
Kubernetes
Image
Helm
Kub
Pod.yaml

z

1. Install the necessary Jenkins plugins:
   1.1 Git plugin
   1.2 Maven Integration plugin
   1.3 Pipeline plugin
   1.4 Kubernetes Continuous Deploy plugin

2. Create a new Jenkins pipeline:
   2.1 In Jenkins, create a new pipeline job and configure it with the Git repository URL for the Java application.
   2.2 Add a Jenkinsfile to the Git repository to define the pipeline stages.

3. Define the pipeline stages:
   Stage 1: Checkout the source code from Git.
   Stage 2: Build the Java application using Maven.
   Stage 3: Run unit tests using JUnit and Mockito.
   Stage 4: Run SonarQube analysis to check the code quality.
   Stage 5: Package the application into a JAR file.
   Stage 6: Deploy the application to a test environment using Helm.
   Stage 7: Run user acceptance tests on the deployed application.
   Stage 8: Promote the application to a production environment using Argo CD.

4. Configure Jenkins pipeline stages:
   Stage 1: Use the Git plugin to check out the source code from the Git repository.
   Stage 2: Use the Maven Integration plugin to build the Java application.
   Stage 3: Use the JUnit and Mockito plugins to run unit tests.
   Stage 4: Use the SonarQube plugin to analyze the code quality of the Java application.
   Stage 5: Use the Maven Integration plugin to package the application into a JAR file.
   Stage 6: Use the Kubernetes Continuous Deploy plugin to deploy the application to a test environment using Helm.
   Stage 7: Use a testing framework like Selenium to run user acceptance tests on the deployed application.
   Stage 8: Use Argo CD to promote the application to a production environment.

5. Set up Argo CD:
   Install Argo CD on the Kubernetes cluster.
   Set up a Git repository for Argo CD to track the changes in the Helm charts and Kubernetes manifests.
   Create a Helm chart for the Java application that includes the Kubernetes manifests and Helm values.
   Add the Helm chart to the Git repository that Argo CD is tracking.

6. Configure Jenkins pipeline to integrate with Argo CD:
   6.1 Add the Argo CD API token to Jenkins credentials.
   6.2 Update the Jenkins pipeline to include the Argo CD deployment stage.

7. Run the Jenkins pipeline:
   7.1 Trigger the Jenkins pipeline to start the CI/CD process for the Java application.
   7.2 Monitor the pipeline stages and fix any issues that arise.