CI/CD Pipeline Project - Rough Notes

Sample Nodejs App is used. Each section can be mini project but when combined it creates a CI/CD pipeline with the pipeline as code. Sections can be divided into

1. Source Control Management - Git Basics
2. Build Automation (Gradle)
3. Continuous Integration (Jenkins)
4. Continuous Delivery (Jenkins Pipeline)
5. Containerization (Docker)
6. Container Orchestration (Kubernetes)
7. Continuous Deployment (All + Github Webhooks)

1. Git basic setup -

- Steps:
- Install git
- Configure git for SSH authentication via GitHub.com
- Create a remote repo
- Add sample app code to repo
- Clone GitHub repo to local terminal
- To make changes just
- "Add" the change to next commit
- "Commit" the change
- "Push" the change to the remote repo
- "Merge" the "pull request"

2. Creating Build Automation with Gradle

Using Gradle as a build automation tool for the app. The build executes automated tests written in code. If the test passes, the build will create a packaged archive of the app's code and dependencies in a zip archive.

For installation - Gradle Wrapper runs scripts to install Gradle easily.

https://docs.gradle.org/current/userguide/gradle_wrapper.html

Create build task steps:

- Initialize the project with gradle init (creates build file)
- Edit the "gradle.build" file with a editor like vim
- Include the com.moowork.node plugin in the gradle project
- Execute automated tests as part of the build with the npm_test task
- Create a task to generate a zip archive (dist/trainSchedule.zip) of the files that need to be deployed to production
- Make sure task dependencies are set up so tasks execute in the correct order
- Commit and push these changes to GitHub repo

Now the app has build automation set up using Gradle wrapper, and can be built with ./gradlew build command.

3. Automating the build automation with Jenkins Projects

The Gradle build can be automatically triggered by code changes for Continuous Integration.

Steps:

- Configure Jenkins to authenticate with GitHub
- Create job in Jenkins
- In Jenkins job - Configure the project to execute the build file and build the nodejs app
- In Jenkins job - Set up GitHub webhook plugin to trigger the build whenever changes are made to the repo in GitHub
- In Jenkins job - Configure the build to archive ta zip file as a build artifact

The end result is to make a code change in GitHub - commit the change - and Jenkins should trigger a build automatically.

4. Creating Jenkins Pipeline to automate app build (CI)

Turning Jenkins jobs into code using Jenkins Pipelines. The pipeline script will build the code, run the automated tests, and archive the zip artifact that is created by the build.

Steps:

- Create a jenkinsfile
- In jenkinsfile create:
- a "stage" to execute the gradle build
- Archiving of the zip artifact
- Create a multibranch pipeline project in Jenkins and run it

5. Pipeline to automate delivery (CD)

Time to deliver the app being built. I'm making two servers here - one for production and one for staging. I'm going to setup a Jenkins pipeline script to deploy the app to a staging server automatically and then requires manual input to deploy to production server. (Note: Most of the time companies won't implement continuous DEPLOYMENT without have multiple tests set up first that needs to be passed for the app to automatically deploy.) Continuous Delivery still requires a final manual push to production. Which can be safer but requires human interaction, depending on the situation Continuous Delivery can be a better option than Continuous Deployment and vice versa. In further projects I am going to write some tests - possible tests for deployments are through canary testing, smoke tests, and other coded tests etc.

Pipeline to automate delivery (CD) Steps:

- Create both staging and production servers for the publish over SSH plugin
- Give Jenkins credentials to SSH into both servers
- Create a multibranch pipeline set up to build from Github repo

- Create a jenkinsfile and in it create “stages” with steps
- a "stage" to deploy app to staging server
- Run the build to deploy to staging server
- a second "stage" to deploy app to production server
- Run the build and approve to deploy to production server

6. Containerize app using Docker

Making the app able to run in a container. I am going to create a Dockerfile for the application to create the Docker Image. Steps:

- Create a new file to be the dockerfile (vim Dockerfile)
- FROM NODE:CARBON use appropriate built in image for nodejs
- WORKDIR to change to working directory
- COPY json files (dependency files) to current working directory
- RUN npm install
- COPY .. all source code of local folder into working directory
- EXPOSE listening port number (8080)
- CMD (command) to run NPM and Start (note it takes arrays)
- build the docker image and send to dockerhub (docker build [OPTIONS] PATH | URL | -)
- Run and test docker image

7. Deploying containerized NodeJS app in Pipeline

- Install docker plugin in Jenkins
- In Jenkins Configure Jenkins credentials for Docker Hub & environment variable for production server IP address, name the variable (example = prod_ip)
- (In GitHub create API token key for Jenkins access to GitHub)
- Create multi branch pipeline project, add GitHub credentials , link repo
- Edit jenkinsfile to add stage to build Docker image
- Put app into variable - e.g app = docker.build("dockerhub/imagename"
- app.inside (executes command "inside" docker container)
- sh echo curl localhost8000 (smoke test to verify it works)
- Add another stage to push Docker Image to GitHub after making it
- put sample "docker registry”, put dockerhub login credentials to let jenkins use to push image to dockerhub, app.push, add variable for build number for each build gets a new number, app.push(latest) to always push latest dockerimage
- Edit Jenkinsfile stage to remove old docker containers and replace with latest, milestone prevents old version overwriting newer version if there are multiple builds, script to log in via SSH and pull latest dockerimage , stop and remove old dockerimage and restart always container with latest dockerimage thats always running
- When running docker container on production server its usually best practice to have it set to restart always for it to always restart itself anytime it goes down.

8. Setting up Kubernetes (Container Orchestration)

- Create a basic cluster (master and a node) set up. There are many ways, and it can even get a little complex use https://kubernetes.io/docs/setup/
- Quick recap - Initialize master node with kubeadm which is used to bootstrap a cluster, create a config file for kubeadm (in yaml) and then run it. Important things to note are the networking pod subnet and give appropriate CIDR range. example sudo kubeadm init --config kube-config.yaml
- Configure Kubectl - which lets you "talk" to your cluster via command line - After the kubeadm init, follow the steps to start using the cluster as it appears. Copy and paste the three commands in terminal and kubectl should begin to work

9. Deploying to Kubernetes with Jenkins Pipeline

- In Jenkins Install Kubernetes continuous deploy plugin
- Add kubeconfig from Kubernetes master as a credential in jenkins for jenkins to be able to authenticate to Kubernetes master node to carry out deployment.
- ssh into Kubernetes master to get kubeconfig file and paste into jenkins
- Create Kubernetes template file (YAML)
- 2 templates are in file – Node port Service & Deployment
- Node port Service -, Opens port on Kubernetes nodes to access NodeJS app. (not necessarily real world applicable to run on 8080, normally has load balancer)
- Deployment - creates 2 replica pods to be created, selector tells Kubernetes which app this deployment is for. Template to apply app label to pods that are deployed. Specify containers to be created using docker image (docker plug in replaces the $ with the actual image)
- Edit Jenkinsfile to add Kubernetes deployment file at the end of Jenkins file.
- After “milestone” put kubernetesDploy with the kubeconfigID (credential) and configs = config files to deploy , enableConfigSubsitution allows variable for $ dockerimage.
- Run Pipeline and test.