***Deployment Life Cycle***

**Idea**: We are trying to deploy our spring boot application in cloud using tools like github, jenkins, docker, dockerhub, kuberbetes, AWS, EC2.

CONTINOUS INTEGRATION: *CI focuses on the process of integrating code changes from multiple contributors into a shared codebase frequently and automatically.*

**Key Practices:**

- Developers commit their code changes to a version control system (e.g., Git) multiple times a day.

- Automated build and test processes are triggered whenever code changes are committed.

- If the automated tests fail, the team is notified immediately, and the issues are addressed.

- The goal is to catch integration problems, bugs, and conflicts early in the development process.

**Benefits**:

- Early detection of integration issues and bugs.

- Faster feedback to developers.

- Improved code quality and stability.

- Enables collaboration among team members.

**Continuous Integration Tools**:

Jenkins, Travis CI, CircleCI, GitLab CI/CD, and many others.

Continuous Delivery (CD):

**Purpose**: CD builds upon CI and focuses on automating the delivery and deployment of code to production or staging environments with minimal manual intervention.

**Key Practices**:

Code that passes CI tests is automatically deployed to staging or pre-production environments.

Automated tests, including integration tests and user acceptance tests, are run in staging environments.

Deployment to production is automated but typically requires manual approval or a triggered release process.

The goal is to ensure that the software is always in a deployable state, and releases can be done quickly and reliably.

**Benefits**:

Faster and more reliable releases.

Reduced deployment risk.

Consistent and repeatable deployment processes.

Improved collaboration between development and operations teams (DevOps).

Continuous Delivery Tools: Jenkins, GitLab CI/CD, AWS CodePipeline, Travis CI, and others.

In summary, while CI focuses on the frequent integration of code changes and automated testing to catch issues early in development, CD extends this by automating the deployment process and ensuring that code is always in a deployable state. CD emphasizes reducing manual intervention in the deployment process and aims for reliable and efficient releases to production.

It's worth noting that there is another related concept called "Continuous Deployment" (CDeploy), which takes CD a step further. Continuous Deployment automatically deploys code changes to production without manual approval, as long as the automated tests pass. This approach is common in organizations that prioritize rapid and frequent releases.

**How we can integrate our Github to Continous Integration tool with Jenkin and how to Automate the Build. (Comes under *CI* only)**

**Step: 1**

Create a spring boot application with dummy apis and push it your github repository.

[*https://github.com/vishalkumar392392/end-to-deployment*](https://github.com/vishalkumar392392/end-to-deployment)

**Step: 2** Jenkins server on AWS EC2

Open your AWS account and create a EC2 instance using Amazon Linux type. Download the key value pair. Now we have to install the Jenkins on the EC2 instance. Follow the below URL to install the jenkins in your EC2.

[*https://www.jenkins.io/doc/tutorials/tutorial-for-installing-jenkins-on-AWS/*](https://www.jenkins.io/doc/tutorials/tutorial-for-installing-jenkins-on-AWS/)

**Step: 3** Jenkins Server boot up

Start the Jenkins server and download the default plugins. Create a user with password.

**Step: 4** Installing missing dependencies.

Install the Git and maven in EC2 instance as root user sudo -i

**Step: 5** Process of creating a new

1. Click on the New Item.
2. Enter a item name, select the FreeStyle Project and click ok.
3. We will go to the configuration page.
4. Give the Description and check the Github Project option and give the project url and display name.
5. In Source code Management section select Git Radio button.
6. Give Repository url, if you are getting the unable to fetch error msg then it means you don’t have the git installed in your EC2 or you need to set the git path in the manage jenkins/Tools Section.
7. Next Add jenkins credentials by creating.
8. Next in the Branches to build section give branch name as “/main”.
9. In build Triggers section Select the Poll SCM and give the corn expression as “\* \* \* \* \*”. It means build after each commit and push to the repo.
10. In the Build Steps, select the Invoke top-level Maven targets. In Goals write “install”.
11. Click on Apply and Save.
12. Then click on build. The build will start.

Building and Pushing Docker Image to Docker Hub using Jenkins Pipeline

**Step: 1**

Create a spring boot application with dummy apis and push it your github repository.

[*https://github.com/vishalkumar392392/end-to-deployment*](https://github.com/vishalkumar392392/end-to-deployment)

**Step: 2** Jenkins server on AWS EC2

Open your AWS account and create a EC2 instance using Amazon Linux type(t2.large). Download the key value pair. Now we have to install the Jenkins on the EC2 instance. Follow the below URL to install the jenkins in your EC2.

[*https://www.jenkins.io/doc/tutorials/tutorial-for-installing-jenkins-on-AWS/*](https://www.jenkins.io/doc/tutorials/tutorial-for-installing-jenkins-on-AWS/)

**Step: 3** Jenkins Server boot up

Start the Jenkins server and download the default plugins. Create a user with password.

**Step: 4** Installing missing dependencies.

Install the Git and maven in EC2 instance as root user sudo -i

**Step 5**: Install Docker in EC2 instance. Follow the below url.

<https://www.cyberciti.biz/faq/how-to-install-docker-on-amazon-linux-2/>

Execute this below commands other wise we cannot connect to docker from jenkins server.

***sudo usermod -aG docker jenkins***

***sudo systemctl restart jenkin****s*

**Step 6**: **Maven Config**

1. Click on Manage Jenkins/Tools, configure the maven.

2. Under Maven Installation, give name as 3.8.4, select the version as 3.8.5. Click apply and Save.

**Step 7**: Create the script for the build trigger and run the pipeline. Refer below link

<https://youtu.be/PKcGy9oPVXg?si=9NysjeaCM2LrS7eq>

Install kubernetes:

sudo vi /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

exclude=kubelet kubeadm kubectl

sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes

sudo systemctl start kubelet

sudo systemctl enable kubelet

sudo kubeadm init --pod-network-cidr=192.168.0.0/16

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

export KUBECONFIG=/etc/kubernetes/admin.conf

kubectl apply -f <https://docs.projectcalico.org/manifests/calico.yaml>