**Gitlab CI CD intro:**

Gitlab CI CD pipeline is the most commonly used pipeline in the development community. It helps developers to concentrate only on developing the application and after that taking **from Build** process to **Deployment** everything is taken care by the pipeline.

Now what is **Gitlab & CI CD pipeline?**

Instead of taking Wiki definition, in layman terms **Gitlab** is a DevOps tool which just like GitHub provides version control, issue-tracking etc. for your projects and along with it is also a great tool to solve the DevOps need for your projects.

**CI CD** is a term used for software development lifecycle which is a combination of multiple practices starting from writing your code to deploying it.

Overall, Gitlab provides developers to solve their DevOps needs by automating the process from Build to Deployment by creating a CI/CD pipeline which is easily achievable using Gitlab.

Deployment can be done through a server(Virtual machine) or through Docker images deployed on Kubernetes or using different cloud services.

**Gitlab CI/CD to server(Virtual machine):**

Now let’s consider a situation where you are a web developer and you are working on a project.

**Localhost deployment:**

So, when you work on the project you use **a build tool** like Maven, Gradle etc. and when you want to run and test your application you first build it and then run it on a local server maybe using **Tomcat, Jetty** etc. and then you see your web app on your browser running on **localhost at some port**.

**Web deployment:**

But our ultimate goal is to deploy it on the web running on something like [WWW.ABC.COM](http://WWW.ABC.COM). So, one way to achieve this is to do a similar process of deploying but instead of our machine(our PC or laptop) we deploy it on a **different machine** with dedicated resources for this which then can be accessed from the internet but by use of **DNS** where the IP of that machine is assigned a **domain** name which we call web URL.

**How Gitlab works:**

While working on our project let’s sometimes we come across a file called “**YAML**”. This file we sometime use for dependencies and sometimes for deployment.

We also have something called **Gitlab runners**(servers) that runs our defined jobs on the file and stores the output for future use.

In Gitlab these two work together to achieve CI/CD pipeline.

Everything we do after committing the code is defined as steps in the .yml file starting from Build to Deployment.

**Docker:**

Docker is an image repository which can be run as containers on different machines like they do on a local machine manually.

An image is a replica of the application environment setup and a running instance of that image is a container.

**Kubernetes:**

Kubernetes is a container orchestration tool. Apart from other multiple usages it helps in distributing the load on the server by running multiple containers on different servers at the time of high load.

To deploy a web application with help of Kubernetes and docker helps to distribute and manage the application easily.

The steps to deploy application using docker and Kubernetes is similar till build step but the deployment of the build will done through containers on Kubernetes instead of machine directly.

1. We should have an image defined in Packages -> Container repository which will contain the environment setup for our project
2. Next we write the .yml file:
   1. First define the image to use that is defined in 1st step
   2. Define stages like build, deploy
   3. Define variables to be used (if any)
   4. Define each stage in detail specifying what it is supposed to do under “script”
   5. Each script under each stage is the steps that the runner has to do in order to finally achieve deployment which is nothing but a shell script

Commit the code and your pipeline will start, solve for any errors and your project is finally deployed.

Example .yml code :

image: registry.abc.com/base: image1

stages:

- build

- deploy

variables:

WAR\_FILE: target/\*. War

build\_war:

stage: build

script:

* mvn clean install

artifacts:

paths:

* $WAR\_FILE

deploy\_code:

stage: deploy

image: image1

script:

* docker login
* docker build
* docker push

The above code only puts the image in the docker registry after building it.

Now that we have the image in the docker repository we open Kubernetes UI and define steps inside it which are easy to do as UI is self-explanatory. Define the image, servers and other configuration variables to finally deploy it.