What is CI and CD Pipeline:

CI --- Continuous Integration

CD --- Continuous Delivery and Continuous Deployment

Build a web application which is going to deployed on live web servers

- Need to have set of developers who are responsible for writing code which will go on and build the web application
- When this code is committed into version control System(Such as Git, SVN) by the team of developers
- Next its goes through the build phase which is first phase of the pipeline where developers put their code and then again code goes to the version control system having a proper version tags.

Build Phase:

Suppose we have a Java code and it needs to be compiled before execution. So, through the version control phase, it again goes to build phase where it gets compiled. You get all the features of that code from various branches of the repository, which merge them and finally use a compiler to compile it.

Testing Phase:

Once the build phase is over, next **testing phase**. In this phase, we have various kinds of testing, one of them is the *unit test* (you test the unit of software or for its sanity test).

Deploy Phase:

When the test is completed, next **deploy phase**, where you deploy it into a staging or a test server. you can view the code or you can view the app in a simulator.

Auto Test Phase:

Once the code is deployed successfully, you can run another set of a sanity test. If everything is accepted, then it can be deployed to production.

Deploy to Production:

if there is some error, you can shoot a mail back to the development team so that they can fix them. Then they will push it into the version control system and goes back into the pipeline.

Once again if there is any error reported during testing, again the feedback goes to the dev team where they fix it and the process re-iterates if required.

Measure+ Validation:

lifecycle continues until we get a code or a product which can be deployed in the production server where we measure and validate the code.

Importance of CI and CD Pipeline:

Our task is to automate the entire process, from the time the development team gives us the code and commits it to the time we get it into production.

Our task is to automate the pipeline in order to make the entire software development lifecycle on the dev-ops mode/ automated mode. For this, they would need automation tools.

we have a git repository where the development team will commit the code. Then Jenkins takes over from there which is front-end tool where you can define your entire job or the task. Our job is to ensure the continuous integration and delivery process for that particular tool or for the particular application.

From Git, Jenkins pulls the code and then moves it to the **commit phase**, where the code is committed from every branch. Then Jenkins moves it into the **build phase** where we compile the code. If it is Java code, we use tools like maven in Jenkins and then compile that code, which we can be deployed to run a series of tests. These test cases are overseen by Jenkins again.

Then it moves on to the staging server to deploy it using **docker**. After a series of Unit Tests or sanity test, it moves to the production.

This is how the delivery phase is taken care by a tool called **Jenkins,** which automate everything. Now in order to deploy it, we will need an environment which will replicate the production environment, i.e., **Docker**.

- ***Jenkins*** provides us with various interfaces and tools in order to automate the entire process.
- ***Docker*** is just like a virtual environment in which we can create a server. It takes a few seconds to create an entire server and deploy the artifacts which we want to test.

Building CI CD Pipeline using Docker and Jenkins

1. Start Jenkins and Docker using the commands "**systemctl start jenkins**", "**systemctl enable jenkins**", "**systemctl start docker**
2. Open your Jenkins on your specified port. Click on **New Item** to create a Job.
3. Select **freestyle** project and provide the item name (here I have given test) and click OK.
4. Select **Source Code Management** and provide the **Git** repository. Click on **Apply** and **Save** button.
5. Then click on **Build->Select Execute Shell**
6. it will build the archive file to get a war file. After that, it will get the code which is already pulled and then it uses maven to install the package. So, it simply installs the dependencies and compiles the application.

```
#!/bin/bash
echo " Starting CI CD Pipeline task"
#.Build
echo " "
echo "Build phase Started :: Compiling Source Code"
cd java_web_code
mvn install

#. Build (Test)
echo " "
echo  " Test phase Started:: Testing Via Automated Scripts"
cd ../integration-testing/
mvn clean verify -P integration-test
```

7. Create the new **Job** by clicking on New Item.
8. Select **freestyle** project and provide the item name (here I have given Sample test) and click on OK.
9. Select **Source Code Management** and provide the **Git** repository. Click on **Apply** and **Save** button.
10. Then click on **Build->Select Execute Shell**.
11. Provide the shell commands. Here it will start the integration phase and **build** the Docker Container.

```
 #!/bin/bash
#.POSTBUILD (PROVISING DEPLOYMENT)
echo “ ”
echo “ Integration phase started:: Copying Artificats
cd java_web_code/
/bin/cp target/wildfly-spring-boot-sample-1.0.0.war  ../docker/
echo “ “
echo “ Provisioning phase started :: Building Docker container”
cd ../docker/
sudo docker build -t Devops_pipeline_demo .
```

12. Create the new **Job** by clicking on New Item.
13. Select **freestyle** project and provide the item name (here I have given sample-test1) and click on OK.
14. Select **Source Code Management** and provide the **Git** repository. Click on **Apply** and **Save** button.
15. Select **Source Code Management** and provide the **Git** repository. Click on **Apply** and **Save** button.
16. Provide the shell commands. Here it will check for the Docker Container file and then deploy it on port number 8180. Click on Save button.

```
RUNNING=$(sudo docker inspect  --format=”{{ .State.Running }}” $CONTAINER 2> /dev/null)
If [ $? -eq 1]; then
  echo “ ‘$CONTAINER’ does not exist”
else
      sudo docker rm -f $CONTAINER
fi
      echo “ “
      echo “ Deploying phase started :: Building Docker container”
      sudo docker run -d -p 8180:8080  -- name devops_pipeline_demo  devops_pipeline_demo
echo “ ________________________________________”
echo “ View app deployed here: https://server-ip:8080/sample.txt”
echo “ -----------------------------------------------------------”
```

17. Now click on **test -> Configure**.
18. Click on **Post-build Actions -> Build other projects**.
19. Provide the project name to build after test (here is Sample test) and then click on **Save**.
20. Now click on **Sample test -> Configure**.

21. Provide the project name to build after Sample test (here is sample-test1) and then click on **Save**.
22. Now we will be creating a Pipeline view. Click on '+' sign.
23. Select **Build Pipeline View** and provide the view name (here I have provided CI CD Pipeline).
24. Select the **initial Job** (here I have provided test) and click on OK.
25. Click on **Run** button to start the the CI CD process.
26. After successful build open **localhost:8180/sample.text**. It will run the application.

Sample.text

Hello
Welcome to my world

Docker:

- It is mainly used in the local development process, so when your developing a software application you would use docker containers for different services that your applications depends on like database message brokers etc.
- It is also used in the CI process to build your application & package into isolated container environment
- Once built that container gets stored or pushed into a private repository

Kubernetes:

- If you have a development server that is made up of multiple virtual or physical servers basically install kuberentes on top of server
- Once Kubernetes is running you will create a cluster that could actually run your docker container on top of it.
- You have kuberentes engine that spend multiple virtual & physical servers to create one cluster
- Where docker containers are actually running & you can distributes the no of docker containers across those physical or virtual servers
- Each containers will be its own application and the Kubernetes service that actually enables docker to run in the cluster is Kubelets so each node in the Kubernetes cluster will actually have one cluster and the technology that is actually comparable with Kubernetes in docker swarm.
- Docker swarm is basically an alternative to Kubernetes which is a container orchestration tool
- Instead of kubletes—you have services called docker daemons that run on each nodes
- Instead of Kubernetes engine you would just have docker that actually spend those multiple nodes that makeup the cluster same eith docker container with the same applications running on the cluster setup

	kuberentes	Docker Swarm
Installation & Cluster config	Setup is very compilated but once installed cluster is robust	Installation is very simple but the cluster is not robust
GUI	GUI is the Kubernetes Dashboard	There is no GUI

Scalability	Highly Scalable and scales fast	Highly scalable & scales 5x faster thank Kuberentes
Auto Scaling	Kuberenetes can do auto scaling	Cannot do auto scaling
Load Balancer	Manual intervention needed for LB traffic between different containers and pods	Auto LB of traffic betwwn containers in the cluster
Rolling updates & Rollbacks	Can deploy rolling updates & does automatic rollbacks	Can deploy rolling updates but not automatic rollback
Data Volumes	Can share storage volumes only with the other containers in the same pods	Can share storage volumes with any other containers
Logging & Mointoring	In-built tools for logging &monitoring	3rd party tool like ELK stack should be used

Docker	Kuberentes
Container Technology means which creates isolated environment for application	Infrastructure for managing multiple containers
Automated Building & deploying application -CI/CD process(Before & when Deploying)	Automated scheduling and managing of application containers(After container deployment)
Containers platform for configuring building & distributing conatiners	Ecosysytem for managing a cluster of Docker containers

Microservices:

- Way of breaking large software projects into loosely couples modules which communicates with each other through simple application programming interfaces(API)
- Microservices are really more than another architectural solution for designing complex—web based applications
- Example: Suppose I need to build a classic web application using java. 1st thing is presentation layer(user interface) followed by an application layer which handles all business logic . this is followed by an integration layer to enables loose coupling betwwn various components of the application layers.Finally I will design database layer that will be accessible to the underlying persistence system.
- To run entir application , I will create either EAR or WAR package and deploy it on an application server(JBoss,Tomcat) Because I have packages everything as an EAR/WAR my application becomes monolithic in nature which means that even through we have separate and distinguishable components all are packaged together.

Advantages of Microservices:

1. Improves fault isolation
2. Eliminate vendor or technology lock-in
3. Ease of understanding
4. Smaller and faster deployments
5. Scalability

Disadvantages of Microservices:

- Communication between services is complex
- More services equal more resources
- Global testing is difficult
- Debugging problems can be harder
- Deployment Challengers
- Large VS small product Companies

What is GitFlow

- Git-Flow is a branching model and release management strategy for Git.
- It defines a well formulated path for the project development life cycle and ensure that the development team adheres to the processes
- It offers a set of extensions over GIT to provide a high level repository operation
- It also help the developers and management to keep track of all features manage release flow and deal with parallel hot fixes with its strictly typed branching conventions
- Developers create feature branches from master branch and work on them. once done, they create pull requests
- In pull requests, each developer comment on changes
- The pull request is completed and ready to be merged to the main branch.

Advantages:

1. Project Type: open source
2. Team size: no limit(small,medium,large)
3. Developer Experiences: Suitable for team with a lot of junior developer
4. Suitable for product development that already established before.

Truck Based Development:

- It is source control branching model, where developers collaborates on code in a single branch (Trunk). Resist any pressure to create other long-lived development branches by employing documented techniques.
- Developers start work on main branch or create shortlisted feature branch and start work on them.
- Once they are done with compiling and all testing they push all changes to trunk directly.

Advantages:

1. Project-type: Closed Source
2. Suitable for starting up development
3. Support fast iteration development
4. Developer experiences: Suitable for tea with a lot of senior developer
5. Flow Visualization looks very simple

GIT VS TFS(team Foundation Server):

- TFS is a centralized version
  Git is distributed as everyone has a full copy of the whole repo and its history
- TFS has its own lang: Check-in/Check-out is a different concept
- TFS users Check-in which invokes file locking
  Git users do commits based on distributed full versions with differences checking

- TFS provides a shelf to hold local changes temporarily
  Git provides a stash area away from items being committed
- Shelve-sets in TFS are stored in the central Server
  Stashed items in Git remain local machine
- TFS groups changes in sequentially numbered changesets
  Git assigns a 32-byte hash to each commit

SVN(Subversion)	GIT
It is centralized repository, directly we involved in the centralized repository	Git is distributed repository, we are working in our laptop after we are transferring the code from our laptop to centralized repository. Git has 3 phases workspace,staging/index,localrepo
We work on SVN if we are facing any networking issues we cant work on SVN because of we are directly involve into centralize repository	In Git, we working on local system only so no need internet connection whenpushing the code from 1 system to centralized repository at that time we need n/w repository
Developers directly interact with the centralized repository	Developers not directly interact with centralized reository

What is a merge conflict?

Sometimes two developers will change the same line of code in two different ways; in such a case, Git can't tell which version is correct

Resolving merge conflicts can take a minute or they can take days (if there are a lot of files that need to be fixed)sync your code multiple times a day by committing, pushing, pulling, and merging often.

How do you resolve a git merge conflict?

Create a new Git repo, add a file, make a branch, make some conflicting edits
Start with an empty directory and run git init:

```
$ ls -l
$ git init
```

create a README file and commit**t**

```
echo "This is a new README file" > README.md
$ cat README.md
This is a new README file
$ git add README.md
$ git commit -m "README file added"
$ git status
On branch master
```

Create a new branch:

```
$ git checkout -b "branch_to_create_merge_conflict"
$ git branch
* branch_to_create_merge_conflict
master
```

Make an edit:

```
This is a new README file

This is an edit on the branch
```

commit that edit:

```
$ vim README.md
$ git add README.md
$ git commit -m "Edits made to README on the branch"
```

Return to the master branch, edit the README on line 3

```
$ git checkout master
```

Edit the README:

```
This is a new README file

This is an edit on the master branch
```

Commit the edit:

```
$ git add README.md

$ git commit -m "Edits made to README on the master branch"
```

Merge the branch into master to see the error:

```
$ git branch
  branch_to_create_merge_conflict
* master
$ git merge branch_to_create_merge_conflict
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

go into the README file, as Git

```
This is a new README file
<<<<<<< HEAD
This is an edit on the master branch
=======
This is an edit on the branch
>>>>>>> branch_to_create_merge_conflict
```

<u>Git added some syntax including seven "less than" characters, <<<<<<< and seven "greater than" characters, >>>>>>>, separated by seven equal signs, =======.</u>

That there are two sections within this block:

- The "less than" characters denote the current branch's edits and the equal signs denote the end of the first section.
- The second section is where the edits are from the attempted merge; it starts with the equal signs and ends with the "greater than" signs.

You can run git status to see details

```
$ vim README.md
$ git status
```

```
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add ..." to mark resolution)
  both modified: README.md
no changes added to commit (use "git add" and/or "git commit -a")
```

you can abort the merge by running git merge --abort to abort the merge.

```
$ git add README.md
$ git status
On branch master
All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)
Changes to be committed:
modified: README.md
$ git commit
[master 9937ca4] Merge branch 'branch_to_create_merge_conflict'
```

Concept of Branching in GIT:

Git is the most popular version control system (VCS) in the world for keeping track of text-based file changes. It also provides a distributed collaborative work environment.
Many VCS is the Branching feature which enables us to open a new channel for a specific set of changes like a bug fix, new feature implementation or an experiment without interfering with the mainstream or parallel changes, in git, the branching is a very lightweight operation.

- The software projects have two mainstream branches as **master** and **dev**.
- The master branch is used for ready for production changes and the dev branch is used for testing purposes. When new changes are tested on the dev branch, eventually it will be merged into the master branch and finally, we will take a deployment on the master branch

- I already have a GitHub repository. Go to the homepage of that repository and click the “1 branch” section.
- I have only the “master” branch which is created by default. Also, it is the default branch that indicates that repository homepage content is loaded as the master branch is selected.
- As I said before most software projects have common branches as **master** and **dev**. By typing “dev” and clicking the appeared “Create branch: dev” section, and I can create the dev branch on my remote repository.

Create a Branch on Local Repository

- $git branch
- $git branch --list
  If you want to exit from that branch list page, just hit the q key from your keyboard.
  I couldn’t see the dev branch. If you want to see your remote repository branches
- $git branch -r

Fetching Changes from Remote Repository

I need to fetch all up to date changes from my remote repo. **git fetch** we retrieved the latest remote metadata without any copy operation

- $git fetch –all
  In order to retrieve and copy all the remote changes, we should use the **git pull** command.
- $git pull origin dev
  origin keyword indicates the remote repository
  To see both remote and local branches
- $git branch -a
  we need to change our current branch from master to dev
- $git checkout dev

create a new branch from the dev branch

- $git branch my-new-faeture

you can still rename any of your branches without checking out

- $git branch -m my-new-faeture my-new-feature

check our local repository with **git status** command.

<u>Git Flow :</u>

The Git Flow is the most known workflow on this list

Advantages

- Ensures a clean state of branches at any given moment in the life cycle of project
- The branches naming follows a systematic pattern making it easier to comprehend
- <u>It has extensions</u> and support on most used git tools
- It is ideal when there it needs to be multiple version in production

Disadvantages

- The Git history becomes unreadable
- The master/develop split is considered redundant and makes the Continuous Delivery and the Continuos Integration harder
- It isn't recommended when it need to maintain single version in production

<u>GitHub Flow:</u>

The GitHub Flow is a lightweight workflow.

Advantages

- it is friendly for the Continuous Delivery and Continuous Integration
- A simpler alternative to Git Flow
- It is ideal when it needs to maintain single version in production

Disadvantages

- The production code can become unstable most easily
- Are not adequate when it needs the release plans
- It doesn't resolve anything about deploy, environments, releases, and issues
- It isn't recommended when multiple versions in production are needed

<u>GitLab Flow:</u>

It combine feature-driven development and feature branches with issue tracking.

The most difference between GitLab Flow and GitHub Flow are the environment branches having in GitLab Flow (e.g. staging and production) because there will be a project that isn't able to deploy to production every time you merge a feature branch

Advantages

- It defines how to make the Continuous Integration and Continuous Delivery
- The git history will be cleaner, less messy and more readable (prefers squash and merge, instead of only merging, on this article)
- It is ideal when it needs to single version in production

Disadvantages

- It is more complex that the GitHub Flow
- It can become complex as Git Flow when it needs to maintain multiple version in production

<u>Different Phases in Devops:</u>

<u>Continuous Development</u> → *Plan application objectives and Code the requirements*

- you should plan your application objectives that must be delivered to the customer.
- It includes activities like code generation and putting the same to the next phase(application objectives, start with the project development)
- the continuous development approach, work may carry out on the existing code by using continuous feedback in the development and operation

<u>Continuous Integration --→</u> Plan Tests and Build the product

- automatically starts after development
- steps like the planning of tests that will be carried out in the next phase, understanding the code to produce the desired outcome as needed in the initial project documentation.
- Continuous integration is the seamless process in DevOps that leads to the next phase in an efficient manner

<u>Continuous Testing -→</u> Verify the product for actual usage in a live environment

- Testing process checks the actual use of an application in the DevOps
- The testing process gives more information about different aspects of an application that in turn is sent to the development process to improve the application.

<u>Continuous Monitoring</u> --→ Improvise the current product and helps to release new versions quickly

- the operational phase in DevOps where key information about application usage is recorded and carefully processed to find out trends and identify the problem areas.
- It enhances the operational efficiencies of a software product that may occur in the form of documents or produce massive data about application parameters when the application is in a continuous use position.

<u>Continuous Feedback</u> → Improvise the current product and helps to release new versions quickly

- The application performance is improved consistently by analyzing the final outcome of the product.
- The continuous feedback is an important phase of the software application where customer feedback is a big asset to improve the working of the current software product and release new versions quickly based on the response.

<u>Continuous Deployment</u> → Ensures product is deployed with maximum accuracy

- The <u>deployment</u> process is performed in such a way that any changes made in the code should not affect the functioning of high traffic website

Continuous operations →Automate release process with shorter development cycles

- operations are based on continuity with complete automation of the release process and allow organizations to accelerate the overall time to market on an ongoing basis.
- It is clear from the discussion that continuity is the critical factor in DevOps removing the abundant steps that often distract the development, take it longer to detect issues, and producing a better version of the product after several months.
- you can make any software product more efficient and increase the overall count of interested customers in your product