**Jenkins**

Agile teams produce workable and robust code in each iteration. all that code if bult and evaluated returns lot of conflicts bugs and errors. Developers needs to solve those conflict and issues before moving to next iteration. the more programmers are sharing the code, the more problematic this is. for these reasons, agile teams often therefore choose to use continuous integration. Some terminologies before we begin. Source code All the code that developers write to crate the software is called as source code.

**The Build Process**

It is process by which source code is converted into a stand -alone form that can be run on a computer. For example, a source code written to develop a windows software when built will create a.jar. or ear or .msi file . Another example if a java source code is built it may create A.jar. or ear file. this deployable manally.but there are some build tools that make developers life easy when it comes to building artefact or even deploying it . these are called as build automation tools.

**There are some build tools:**

* Ant
* Maven
* MS build
* Nant
* Gradle

**The unit testing is:**

Unit testing simply verifying the individual unit of code (mostly functions) works as expected. Developer along with the writing the code will writing the code will the test cases that can be executed at the build time. some test cases can be automatically generated. The objective of unit testing is to isolate a section of code (unit) and verify its correctness What is continuous integration Continuous integration (ci) is the process of automating the build and testing of code every time a team member commits changes to version control .ci encourages version control repository after every small task completion. committing code triggers an automated build system to grab the latest code from the shared repository and to build, test, and validate the full master branch (also known as the trunk or main). The problem

Developers will the code and verify locally they push it to their local system. Once developers test the code and verify locally, they push it to the centralised repository like GitHub. Similarly, all the developers would be pushing their code to version control system several times a day. Developers would be pushing code and keep writing the code until they finish a particular task or the project. now all the code which developer have pushed into the version control systems, if built and tested will return lots & lots of conflicts, error due to fail of the build.

**The solution:**

The whenever the developer push the code to the version control system it should be fetched, built & tested by a build sever at the same time This process repeated several times in a week or day or daily once is called as continuous integration. Developers code is continuous integration so any point in time we have a workable software, if there is an issue in the build process the developers will get notified through email and they will fix the problem. Jenkins is a continuous integration server which can be fetch the latest code from the version control system, build it and test it and notify it to developers. Jenkins can do many things apart from just being a continuous integration sever.it was originally known as Hudson, oracle owns Hudson now. Jenkins is an open-source project written by kohuske Kawaguchi. Jenkins is a java-based web application server. AS a prerequisite, we need a java on the machine to run a Jenkins sever

**The feature of Jenkins:**

It is as open source there is a lot of contribution all around the world to the Jenkins software. it has all the latest features and greatest features the developers will integrating into it regularly.

**extensible**

Jenkins comes with a lot of goodies but is just not limited by that, Jenkins main power is its extensibility that can be achieved by installing plugins into it. Jenkins opensource community has written more number of plugins, these plugins can do variety of tasks, like integration with external tools or servers they are:

* 1.version control system plugins: - git, svn, subversion etc
* 2.build plugins: - maven, ant, Ms build etc
* 3.notification plugins: - Email, chat, sms etc
* 4.cloud pliugins: - create cloud instance and deploy the code into the cloud service
* 5.testing piugins: -code analysis will be a unit test cases and static code analysis etc

The list of plugins is very long, whenever we want Jenkins to do some tasks just search for that plugin and most of the time, we will find something. For example, that if we want Jenkins to deploy java artefact to tomcat server, search for the plugin named “deploy to container”.

**Jenkins setup**

Jenkins can be installed on windows Browse to the official Jenkins download page. Under the Downloading Jenkins section is a list of installers for the long-term support (LTS) version of Jenkins. Click the Windows link to begin the download. Once the download is complete, run the jenkins.msi

Prerequisites

* A system running Windows 10
* The latest copy of Java Development Kit or Java Runtime Environment installed
* Access to an account with administrator privileges

**Install Jenkins on Windows**

1. Browse to the official Jenkins download page. Under the Downloading Jenkins section is a list of installers for the long-term support (LTS) version of Jenkins. Click the Windows link to begin the download.
2. Once the download is complete, run the jenkins.msi installation file.
3. The setup wizard starts. Click Next to proceed.
4. Select the install destination folder and click Next to continue.
5. Under the Run service as a local or domain user option, enter the domain username and password for the user account you want to run Jenkins with. Click Test Credentials to verify the login data, then click Next to proceed. Using the Run service as Local system option doesn’t require you to enter user login data. Instead, it grants Jenkins full access to your system and services. Note that this is a less secure option and is thus not recommended. Note: When selecting the Run service as local or domain user, the user in question must have the required permission to log on as a service. If this is not the case, an error message stating the account cannot be verified appears. In the event of this error, perform the following steps to resolve the issue:

* Make sure you are logged in as a user with administrative privileges.
* In the Administrative Tools, open Local Security Policy.
* In the Local Security Policy window, expand Local Policy in the left-hand panel and select User Rights Assignment.
* In the right-hand side panel, right-click Log on as service and select Properties.
* Click the Add User or Group… button.
* Use the Select Users or Groups dialogue to add the current user and click OK.
* Save changes by clicking the OK button in the Properties window.

1. Enter the port number you want Jenkins to run on. Click Test Port to check if the selected port is available, then click Next to continue.
2. Select the directory where Java is installed on your system and click Next to proceed.
3. Select the features you want to install with Jenkins and click Next to continue.
4. Click Install to start the installation process.
5. Once the installation is complete, click Finish to exit the install wizard. Unblock Jenkins After completing the installation process, you must unblock Jenkins before you can customize and start using it.
6. In your web browser, navigate to the port number you selected during the installation using the following address:
7. Navigate to the location on your system specified by the Unblock Jenkins page Note: The Unblock Jenkins page may point to a hidden directory. In case you cannot find the directory, make sure you enable viewing hidden items.
8. Open the initialAdminPassword
9. file using a text editor such as Notepad.
10. Copy the password from the initialAdminPassword: \*\*\*\*\*@#
11. instances faster, lighter.

**Docker:**

**That first we know about virtualization**

from now, deploying a service was a process that was both slow and painful. The process involved in writing of code by the development team and then its deployment by the operations team on metal machines. The operations team used to have their work cut out as they had to look for language compilers, libraries, and patches to make the code work. If the process had any errors or bugs, it would have to start all over again – the development team would fix the bugs or errors, and the operations team would again begin deploying the code. Things got a little better when Hypervisors were developed. So, what are Hypervisors? These are a collection of virtual machines (VMs) that may continuously be running or switched off at regular intervals, especially when not in use. Virtual machines definitely helped to a by accelerating the process of the fixing of errors and the deployment of code, but they still had a few issues. Docker containers came as the real game-changers. They even addressed the issues that existed in virtual machine.

**What is Docker**

It is an open-source platform that is used by developers across the globe to run, package, and distribute applications. Docker makes the process of the encapsulation of applications from the first step to the last, very easy and efficient. To understand Docker in a better manner, you will have to understand what containers are and how they work.

**The Container**

A container is lightweight, and executable package of a part of the software that comes with everything that is required to run it. Containers are in no way dependent on platforms. So, Docker is compatible both with Windows and Linux-based machines. Also, you can even run Docker on a virtual machine The basic objective that Docker aims to achieve is to let developers use distributed architecture to run microservice applications.  
Unlike virtual machines that used to perform the abstraction of hardware, Docker goes a level up and performs abstraction of a different set of resources at the OS level. This provides several benefits, including separation of infrastructure and portability of applications amongst others. In other words, unlike virtual machines that used to abstract the hardware server, Docker’s container-based approach works by abstracting the OS core. This is a great alternative to virtualization that leads to the faster creation of lightweight instances.

**Docker is available in two versions:**

* Enterprise Edition (EE):

This version is specifically designed for IT teams and enterprise development. This version is used to develop, ship, and run applications.

* Community Edition (CE):

This version is used by individuals and small teams that are exploring container-based apps or getting started with Docker.  
Docker Workflow we will focus on the Docker Engine as well as its different components. This will help us in better understanding how Docker works before we move on to Docker architecture. Docker Engine is the power that enables developing to perform various functions using this container-based app. You can use the components that are listed below to build, package, ship, and run applications.

1. Docker Daemon

It is the background process that continuously works to help you manage images, storage volumes, networks, and containers. It is always looking for Docker API requests to process them.

1. Docker CLI

It is an interface client that interacts with Docker Daemon. It helps developers simplify the process of managing container instances. It is one of the primary reasons why developers prefer Docker over other similar applications.

1. Docker Engine Rest API

It facilitates interactions between Docker daemon and applications. An HTTP client is usually required to access these APIs. Docker Architecture Docker architecture is a client-server-based architecture. It has three major components that are mentioned below:

1. Docker host
2. Docker client
3. Docker registry
4. Docker objects

In the initial phase, the Docker client interacts with the daemon, which is responsible

for performing much of the work that goes into developing, running, and distributing Docker containers. Docker daemon and the client can either run on a single system or the developer can use a remote daemon to connect it with a local Docker client. Rest API is used to establish communication between Docker daemon and client. This can be either done over a network interface or UNIX sockets.

1. Docker Host

A Docker host is responsible for running the Docker daemon. Docker Daemon entertains API requests, including docker build and docker run amongst others. It also manages images, networks, containers, and other Docker objects. Daemons can communicate with each other to manage different Docker services.

1. Docker Client

It is nothing but the method that users use to interact with Docker. The Docker client sends our requests, such as docker run, and Docker builds to Docker daemon. A very important feature of Docker client is that it can communicate with several daemons.

1. Docker Registry

A registry is a server-side application that is scalable and stateless. It not only stores Docker images but lets developers distribute them as well. Docker provides us with f //RFFFCc the flexibility to create our own images, or there are public registries available that we can make use of. These registries include Docker Cloud and Docker Hub amongst others. Docker’s configuration is such that it always turns to Docker Hub and other public registries to look for images. However, we have the option of creating our own registry. So, we can pull out the required images using our own registries with the help of docker run and docker pull commands. Docker push command pushes the required image to the registry that we created.

1. Docker Objects

We use and create several objects while using Docker. These objects include containers, images, plugins, volumes, networks, and others.

1. Docker Images

A Docker image is nothing but a read-only template that provides us with the instruction required to create a container. On many occasions, one image has a connection with another image. What differentiates two images is the added layer of customization. To put it differently, an image can also be defined as an immutable snapshot of a container. Images are small, lightweight, and fast. 6. Docker Containers Let’s follow a different approach to understand Docker containers. So, if an image could be used for representing a class, a container could be its instance. In other words, a container is a runtime object. We can create, start, move, stop, or delete containers with the help of Docker CLI or API. Containers can also be attached to storage and connected to one or more than one network. Depending on the current state of a container, we can also create a new image. Conclusion Now that you know what Docker architecture and its components are, you are in a better position to understand the rise in its popularity. It simplifies infrastructure management and helps make instances faster, lighter.

# Middleware

middleware is firewall or plugin that takes the incoming request for the web browser are from a client before of our view this middleware can manipulate those request and response and similarly the middleware will the takes request and that will send it to the view and similarly it also works on the template and it returns the responses as the required and they send to the view

## first will go to project:

we go to project go to setting s.py go to middleware:-

'django.middleware security .security Middleware'

it is a security middleware is reposed for https or sslp sequent and other security futures

'django.contrib.session.middleware.session middleware'

that session middleware is that do all the session management that mean when we use the session API internally is response and that incoming request making that is a session still active when the response go back to the web browser this will add the cookies

'dajango.middleware.common.common,middleware

'

that will works with URLs etc. it makes sure that is a forward slash() and the end points.

'djano.middleware.csrf.csrfviewMiddleware'.

that check every post request when from is submitted it does have a csrf token or else it throwback error

'django.contrib.auth. middleware.AuthenticationMiddleware'.

the authenticate is respectable for the authentication the end user request and it also make sure that there is a user object available for the view for the available for the authentication template one's the view for the temple one's the authentication done

'django.contrib.message.middleware,Message middleware,'

,django.contrib.middleware.click jacking.xFrame.options Middleware',

custom middleware my Middleware

\_init\_(self,get\_reponse,

\_call\_(self,reguest)

process\_view (self,request,view\_kwargs) process\_exception(self,exception) process\_template,reponse(self,request,repsonse)

'django.middleware.crsfviewmiddlware', that in a crsf view that