Phase 4

Testing and deploying the application.

Phase 4

Testing tools.

Jasmine and Karma

Mocha and Chai

Superset testing tool

JEST

Graph QL : Self learning

Docker

Jenkin

Cloud computing

AWS Cloud : EC2, S3 EBS Volume Etc

**Day 1**

Input

Process

Output

MVC

App.js Router, controller, model database

If we want to test controller functionality

Controller must be call from router and router must be call from App.js. We can test the application using some testing tool which verify the application running properly or not.

Testing is use to find the error or defect or bugs in the application or software.

function chckUser(name,pass) {

if(name!=”Raj” && pass !=”123”) {

return “success”;

}else {

return “success”;

}

}

Testing: Testing mainly divided into two types

Black box testing

Input ---🡪 Process -----🡪 Output

A =10 display value

B=20 B is 20

Actual

Expect the output : 30

White box testing

Input---------🡪Process-------------🡪Output

A=10

B=20 coding output put 30

Actual

Expected

Both must be match

Manual testing : we run the testing and manually write the result in testing template.

Automation testing : tool generate the reports. Less interaction from programmer or tester for the loop.

Unit testing: unit testing is testing which help to do the testing for smallest piece for code that can logically verified. Unit also known as function/ method/ class/ modules etc.

Unit is testing is a type of white box testing.

Unit testing is a kind of software testing method is which each individual and independent part of the source code is tested to determine that it is good enough for the user.

When we perform unit testing on the front end (client side) of the software. It is called or known as front end unit testing.

Jasmine : Jasmine is a open source testing framework which help to do the unit testing on front end side technologies.

Using jasmine we can do Client side as well as server side(node JS) testing.

Jasmine provide pre-defined functions which help to do the testing.

1. **Test suite:** Test suite is technically contains more than one test case as well as another test suite. To write test suite testing framework provide pre-pre-definedfunction ie describe.

**Syntax**

describe(“suiteName”,callbackfucntion)

describe(“message”,()=> {

})

1. **Test spec :** Test spec is known as test case which actually help to do the testing. Test spec contains more than one expect which help to check actual and expected output.

Syntax

it(“spectName”,callback)

it(“addition”,()=> {

})

1. **expect :** this construct aids in testing if the expectation from the software are fulfilled or not.

**Testing hook or life cycle functions.**

**beforeEach() : it call again and again before every it functions.**

**afterEach() : it call again and again after every it function.**

**beforeAll() : it call only once before all it functions.**

**afterAll() : it call only once after all it function.**

**fetch() :**

**in react js using axios we call rest API.**

**fetch() is a pre-defined function provide by JavasScript which help to call REST API without depends upon any third party library.**

**Fetch() function return type is promise. Then and catch()**

**Fetch is a pre-defined function part of JavaScript we can use fetch function in Normal JavaScript code.**

**In Fetch we have to use first then() function convert data in json or text format mandatory.**

**Axios is third party library we have to install using node js and we can use in react js application. But in axis it return by default json.**

**Node JS (Server side JavaScript Testing) using jasmine tool.**

**Create folder Server Side JavaScript (Node JS)**

**Create the package.json file**

**Using npm init command**

**We have to install two external modules**

**ie**

**testing dependencies always must be available in development mode.**

**npm install jasmine-node -D**

**npm install jasmine -D**

**or**

**npm install jasmine-node –-save-dev**

**npm install jasmine –save-dev**

**after added dependencies**

**create spec directory using**

**jasmine init command**

**if you get the error jasmine command not recognize.**

**npm install jasmine –g**

**now create src folder**

**Express JS Testing**

**Jasmine with external node Js module ie superTest which help to do the testing for Express JS application.**

**First folder express js testing**

**Package.json file**

**Npm init**

**npm install express**

**npm install jasmine –D**

**npm install supertest –D**

**jasmine init : This command is use to create spec folder**

**create folder src**

**Mocha : Mocha is a feature – rich test framework running on node js as well as client side JavaScript programs.**

**Mocha make asynchronous testing very simple.**

**Jasmine is a big framework has almost everything built into in including assertion/ expectation.**

**Mocha only provide test runner function like describe and it but depends upon third party module like node js assert module or other assertion modules like should.js expect.js or chai.js**

**Mocha with Chai :**

**Mocha with Node JS (node js assertion module)**

**Create folder mocha with node js**

**Create package.json file using npm init**

**npm install mocha –g**

**npm install mocha –D**

**Mocha allow you to use any assertion library you wish,**

**Should.js**

**Expect.js**

**Chai.js**

**Chai.js : chai js is a assertion library which provide set of pre-defined functions which help to check expectation and actual output.**

**Chai with assertion style : The assert style is exposed through assert interface. This provides the classical assert-dot notation similar to assert module in node js.**

**Chai folder**

**Create assert style folder.**

**Create package.json file**

**npm install chai -D**

**Chai with expect style exposed through expect interface. In this scenario you can use natural language assertion.**

**Chai with expect style**

**Create expect style folder**

**Create package.json file**

**Npm install chai –D**

**Chai with should style**

**This style allow for the same chainable assertion as the expect interface. However it extends each object with a should property to start your chain.**

**Create should style folder**

**Create package.json file**

**npm install chai –D**

**Angular -🡪 Jasmine and Karma**

**Client Side JavaScript -🡪 jasmine or Mocha**

**React JS -🡪 JEST Testing framework**

**Node JS (Server sider JavaScript) -🡪Jasmine or Mocha with Chai with style.**

**Node JS with Mocha and Chai**

**Phase 4**

**02-09-2021**

**Create folder mocha with chai**

**Create package.json file**

**npm install mocha –D**

**npm install chai –D**

**create src folder and spec folder**

**Mocha Chai with Express JS Testing**

**Create folder mocha chai with express Js**

**Create package.json file using npm init**

**Npm install mocha –D**

**Npm install chai –D**

**Npm install chai-http –D it is like a supertest**

**Npm install express**

**Create two src and spec**

**JavaScript (Client Side ) ---jasmine**

**Node JS (Server side)**

**Jasmine with supertest for Express**

**Or**

**Mocha and chai with chai-http for express**

**React JS JEST**

**JEST is an open source testing framework build on JavaScript, designed majority to work with React and React Native application.**

**JEST JavaScript testing framework with focus on simplicity. JEST was created facebook.**

**JESt also provide testing runner function like describe, it and expect.**

**Create folder as JEST and insider JavaScript folder**

**Create package.json file**

**npm install jest –D**

**npm install jest –g**

**Create src folder and \_\_tests\_\_ (it is like a spec folder).**

**All testing file must be inside \_\_test\_\_ folder.**

**Create folder React Project**

**Then create react js project using command as**

**create-react-app react-test**

**react internally install jest testing framework.**

**But we required other plugin ie**

**Enzyme : Enzyme is a JavaScript testing utilities for easily testing react component. It helps render react component in testing mode.**

**npm install enzyme –D**

**npm install @wojtekmaj/enzyme-adapter-react-17 -D**

**phase 4 :**

**06-09-2012**

**Docker :**

**Docker : Docker is a advanced OS Virtualization software platform that makes it easier to create, deploy and run application in a Docker container.**

**The Docker container is a very light weighted package that allow the developer to package up an application and deploy is as one with help of inbuilt libraries and other dependencies.**

**In Docker container we can run or deploy any application rather than install software in base machine or virtual machine.**

**Virtualization : virtualization means of employing software (such as Hypervisor) to create virtual version of resources such as a server, data storage(data base) or application.**

**Virtualization lets you divide a system into a series of separate section such as a distinct individual system. This virtual environment is known as virtual machine.**

**VMware software**

**What is a virtual machine : A VM (virtual machine) is a computing environment or software that aids developer to access an operating system via physical base operating system.**

**Base : Window OS**

**16 RAM**

**Virtual machine on base Machine : we have to provide memory and RAM size.**

**Virtual may be Linux or Unix : 4gb**

**Base 12 GB RAM**

**Unix 4 GB RAM.**

**At the same time we have to run the 10 virtual machine.**

**Virtual 🡪 1g RAM**

**Base Machine 🡪 6 GM RAM**

**Docker container : Running instances of Docker images container turn the actual application. A Container includes an application and all of its dependencies.**

**Docker images : The file system and configuration of our application which are used to create the container. Docker image contains everything you need to run our application. It is a template that hold a set of instruction needed to create the working container. Docker images are the source code for our container.**

**Dockerfile : A dockerfile is a blue print / set of instruction that defines how your image built. It is a series of steps that you have defined and that must happen before your image is successfully built.**

**Docker registries : it is use to store the more than one images.**

**Two types of registries**

**Public**

**Private**

**Docker provided public registry ie Docker hub. It is like a github that allows use to host our images and provide access to a wide number of other images than you can pull and run in our machine.**

**Docker commands**

**docker -–version**

**docker images : This command is to display all images present in in our machine.**

**Pull any other image from Docker hub**

**docker pull imagesName**

**Command to run the images**

**docker run imageName**

**busybox images :**

**docker run -it busybox : this command is use to open busybox tiny unix os terminal.**

**Create one folder**

**Alpine images**

**Then create the file with name**

**Dockerfile.txt (without any extension).**

**FROM alpine:latest**

**CMD ["date"]**

**We are creating images base upon alpine images and running simple cmd command to display the date.**

**To create the image open the command prompt in image file location and run the command as**

**docker build –t imageName . –f dockerFileName**

**create the image to run the node js application**

FROM node:latest : load the node js image this image running on alpine os

COPY app.js . copy app.js from source directory to target directory . (in alpine os)

CMD ["node","app.js"] : in alpine server open the command prompt and run the app.js file using node command.

Create the image to run the express js application

First create the folder my express js image

Create the package.json file using command npm init

npm install express

**Dockerfile**

FROM node:latest : pull the node js image

RUN mkdir /app : create folder with name app(it can be anything

WORKDIR /app : move inside app folder

COPY package.json /app/ :copy package.json file in app folder

RUN npm install  : using package.json file install

All node plugin in node image.

COPY app.js /app/ : copy app.js file in node image

Inside app folder

CMD ["node","app.js"] : run the node command to the app.js application.

Then create the image

docker build -t my-express1 . -f Dockerfile

**To run the image if it contains server application means running on port number.**

**To run docker image with contains express js application we have to user command as**

**docker run –p 9090:9090 imageName**

**left side port number is expose port number : right side actual port number**

**docker run –p 9090:9090 my-express1**

**-p port number**

**docker run –d –p 9090:9090 my-express1**

**-d detach mode (background running).**

**then open browser and hit URL.**

**docker ps (process status) :**

**This command is use to display running containers.**

**docker stop containerId**

**This command is use to stop the container.**

**Creating image for the React Application**

**create-react-app my-react-docker**

**create react application with state and props.**

**Then stop the application and create the image using the command as**

**docker build -t my-react1 . -f Dockerfile**

**after created imaged successfully then run the image**

**docker run –d –p 3000:3000 my-react1**

**then run the application on browser using command prompt**

[**http://localhost:3000**](http://localhost:3000)

**What is nginx?**

Nginx (pronounced "engine-x") is an open source reverse proxy server for HTTP, HTTPS, SMTP, POP3, and IMAP protocols, as well as a load balancer,

Nginx is a server which help to deploy our application. It may be normal html page or angular or react or any view technologies.

By default nginx run on port number 80

We create image using nginx and deploy simple html page.

Create folder nginx with html page

**Dockerfile**

FROM nginx:alpine

COPY index.html /usr/share/nginx/html/index.html

**docker build -t my-nginx1 . -f Dockerfile**

**After react js application create we have to build the application.**

**Build phase 🡪 this phase will come after development phase.**

**Develop the application -🡪 build the application 🡪 deploy the application.**

**Please open the command in react js application folder and run the command as**

**npm run build**

**after build successfully inside react js project one folder will create ie build which contains all our build application files.**

**Then create the folder ngix with react build file**

**Then copy and paste all build file inside this folder.**

**Dockerfile**

FROM nginx:alpine

COPY . /usr/share/nginx/html/

**Create the image**

**docker build -t nginx-react . -f Dockerfile**

**run the image**

**docker run -d -p 8686:80 nginx-react**

**creating image using nginx deploy html page in ngnix and publish this docker image in docker hub.**

**Create html page (index.html)**

**Then create Dockerfile**

FROM nginx:alpine

COPY index.html /usr/share/nginx/html/index.html

**Then create the image**

**docker build -t my-web-sep-batch . -f dockerfile**

**before the publish the image we have to create tag for that image ie latest.**

**docker tag imagename dockerhubaccountName/imageName:latest**

**after created tag successfully**

**to push**

**docker push dockeraccountname/imageName:latest**

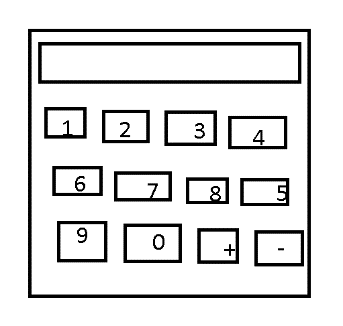
**if you get denies plese write**

**docker login : it ask uername and password. Please do the login through command prompt.**

**Phase 4 phase end project**

**Create react application.**

**The create calculator**

****

**After created simple or complex calculator.**

**npm run build**

**Then it will create the build folder which contains all build files.**

**Then create separate folder and paste all build files inside a folder.**

**Using nxinx create image and build the image.**

**Please test the image running in our machine or not.**

**After that add the tags for your image**

**Docker tag imageName accountName/imageName:latest**

**If you get any error for login please do the login through Docker command**

**Docker login : it ask username and password. Please enter it.**

**The push the images.**

**Jenkin :**

**CI and CD : Continuous Integration and Continuous delivery.**

**Online shopping**

**1 developer**

**2 developer git**

**3 developer**

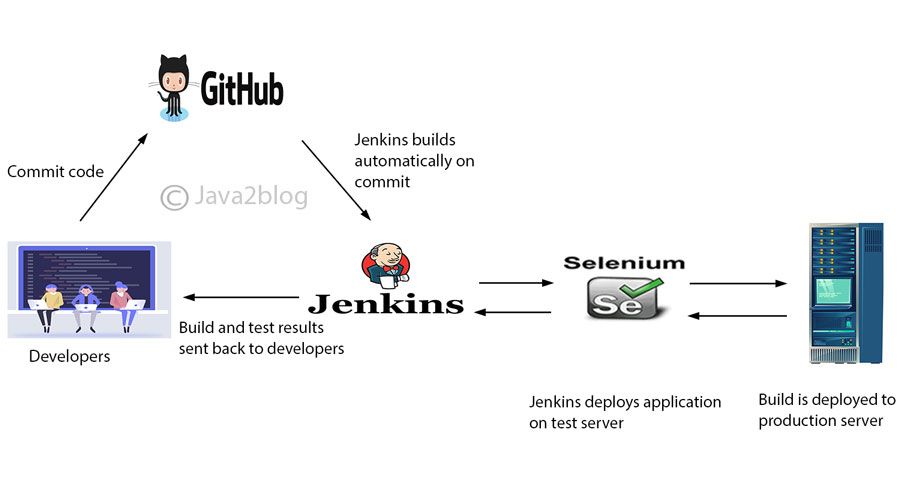
**Old days**

**code changes made by individual team members are merged together into working software. This phase is known as Integration phase.**

**Integration phase was a hard work which often result in code conflict. Hard to find the defects or error.**

**CI and CD : Continuous Integration and Continuous delivery or deploy the application.**

**Jenkin is type of CI Server.**



**Jenkin is open source automation server for continuous integration. It is written using Java technologies. So with Jenkin the integration phase run very smoothly. Jenkin detects changes in sub version control system ie git whenever if any client push the code in git hub.**

**Jenkin perform the task ie build, test, deploy, package and integrate automatically.**

**First pull the image using docker**

**docker pull jenkins/jenkins**

**After pull successfully to run the Jenkin we have run the command as**

**docker run –p 8080:8080 –p 50000:50000 jenkins/jenkins**

**after run the Jenkin open the browser and write the url as**

[**http://localhost:8080**](http://localhost:8080)