chrome runtime for js coding

node js is a js runtine, it runs js, front end chrome v8, backenfd change to node js, framework and libraries

guy who made node js his course

node js is a server/runtime, libraries like ecpress, django of js

route post get

next js

react is a library, use a lot of other libraries, in next it is built in, install router in react

server side rendering and client side

react fotr vclient side, if js off in browser react side wont load, google bots get empty page they only execute and read, load time more,

in react empty html

in next js we populate on server side with js , crawller get populated page, if populatedd, usev dpes need to ru js on his side, so faster

async wait

dynamic ui is client side, static is server side

mix page

static rendered server side

promises in async wait

no code block

oop in js

clouser in js

stack queue

dom react documet objact model, object of each page, in form of a object

get propertoes of html

dom.get elemt by id

react virtual dom

light weight

only loads the changed objects

use state

js is single sided, how code is not blocked?

if 10 ficnvtion anfd one taking time why not blocked, its in one thread, async

js has so many issues, safety is gard to control, , o<null, js is dynamic type language, wors without variable type, to avoid this issue, type script, every function is typed, varuables also firced, errotrs on compile time nnot runtime

type safety in type script

paxkage.json, to build project dependencies

apis, rest ful api

web socket, framework, socket with server, handshake, diffrreence from

server less functions, too many people issue solution, functions defined on aws etc on dwerver it costs even if no one, on server less it doesnt

/////"Serverless" is a buzzword, and I got stuck on it, too. But... it does ultimately make sense.

You dont pay for a cloud server, myserver.azuresites.com, to sit around doing nothing all day until midnight when it runs a script, then resumes doing nothing while you pay for it to be available around the clock.

Instead, you tell Azrue, or AWS, whoever, "here's a script". Just before midnight, Azure loads the script onto whatever server (which they manage) isn't doing much at the time. The script runs, then gets unloaded. You don't know or care which server it ran on, only that it ran. "Serverless".

For a web service, it's one step more complex. You define the web service and let azure know you want them to host it. They put the webservice on one of their disks, in case someone ever calls it. The first time someone does call it, they move it from disk onto a server that isn't doing much, then run your code, which answers the request. This is a 'cold start'. It takes a few seconds.

But now, the server is instructed to keep your code on hand, in memory, for, say, one hour. If your web service is hit several more times soon after that first request, they happen fast because your code is still in memory. If not, your code goes back into cold storage.

Now, you pay (a few cents) per web request, not for an always on server.

erverless is just a way of saying you don't have to manage the server. You tell the host, "please run this code for me real quick," and the host says "sure, just shove it in my black box and we'll have results for you shortly".

Cold starts can be a problem (that can be resolved with the proper application of money), but I also believe that the ideal use case for Serverless is generally async and not exceptionally time sensitive. If you need always-on infrastructure, you are better served by having... infrastructure./////

docker why? platform independt, hybper visor used

containers creating by itself when more load, ubernaties, even duplicates databases when too much load, for distributed computing, decides how to gives resources to containers