WebFront

All of you have shown great progress till now and it's time to go one level beyond with this week 3 assignment. Api's are a very powerful tool and all the things we developers use like GitHub, Slack, Aws, Docker etc. work because of them. In this assignment we have combined these daily use tools and Api's to build something useful, something to take your knowledge to the next realm.

This project will be completely local, with multiple running services and extensive use of docker.

All of you must be familiar with tools like Vercel, Netlify, GitHub Pages, they are used to build and deploy our frontend web apps. Our goal is to build something similar.

Below is the complete journey of how we will collect the source code and deploy it via docker.

Getting Source Code

Our first micro service is "`wf-collect-client`", and it will expose the following api's

API	**Request Body**

/api/v1/collect	`{`
	`  "project_github_url": "", // public url`
	`  "build_command": "npm run build",`
	`  "build_out_dir": "out" // or dist`
	`}`

`project_github_url`	`A public github repo url (Assume we will only support react apps for now)`
`build_command`	`Command to generate the build when executed in the root folder of the cloned repository. Usually its` `` `npm run build` or `yarn build` ``
`build_out_dir`	`Name of the folder generated after the build`

Once this api is called, it will allocate a unique build id to the request and transfer this information along with the ***BUILD_ID*** to another microservice (`wf-code-builder`) via kafka (running via docker) and return the ***BUILD_ID*** to the user.

`wf-client-collect` will expose one more API to show the build events (If you convert this to websockets or SSE to get real time updates, it will be an added advantage).

API	Response
/api/v1/build/{build_id}	{ "build_id": "WF01792BSA2", "project_github_url": <github_url>, "events": { "BUILD_QUEUED": { "timestamp": <>, }, "BUILD_STARTED": { "timestamp" <>, }, "BUILD_FAILED": { "timestamp": <>, "reason": "GITHUB_PRIVATE" // or BUILD_COMMAND_FAILED, INVALID_OUT_DIR }, "BUILD_PASSSED": { "timestamp": <> }, "DEPLOY_PASSED": { "timestamp": <>, "branded_access_url": "https://localhost:3000/BUILD_ID", "url": "https://localhost:7234", }, "DEPLOY_FAILED": { "timestamp": <>, "reason": "Any reason which prevents the build from deploying" } } }

When this API is called it should only show the event's occurred till that time.

Building Source Code

Once the build event is received via kafka on "`wf-code-builder`", it will be added to a queue (You can use Redis here) - we will be processing only **3** concurrent builds at a time. If the queue is free we can directly process it.

Once a build event is selected, you have to utilise the Docker API to generate a build with the given build command. All build events, success or error should be saved to a Postgres DB (Running via docker), so they can be delivered to the user via "/api/v1/build/{build_id}".

Once the build is completed verify the out folder is generated and proceed to deploying the build.

Deploying Source Code

Convert the build to a docker image and serve it via docker only. When the image starts running on docker it will for obvious use a PORT (Allocate the ports efficiently). That same port should be exposed on the host machine (port forwarding). One more thing that needs to be taken care of here is, the build should be accessible via the main API also.

If I go to http://localhsot:3000/BUILD_ID, it should redirect me to the port where the app is running, if the build is still in progress and the user tries to access it, show an error message.

Notes

- Make sure to validate all user input.
- Add verbose logging.
- Maintain a proper README doc to show your execution process.