```
npm install -g bun
npx create-turbo@latest

create db folder in the packages folder
bun init
cd packages
mkdir db
cd db
bun init
bun install prisma
bunx prisma init
bun add dotenv

defining the schema

model User {
  id               String      @id    @default(uuid())
  username         String
  password         String
  todos            Todo[]
}

model Todo{
    id               String      @id
@default(uuid())
    task             String
    done             Boolean     @default(false)
    userId           String
    user             User        @relation(fields:
                           [userId],references: [id])
}

in this there is one to many relationship a user can
have array of todos
```

```
docker run -e POSTGRES_PASSWORD=mysecretpassword -d
-p 5432:5432 postgres

-d == detached mode
-p == port mapped

change the url
DATABASE_URL="postgresql://
postgres:mysecretpassword@localhost:5432/postgres"

bunx prisma migrate dev
bunx prisma generate
pnpm add @prisma/adapter-pg pg
pnpm add -D @types/pg

in the db folder in the index.ts do this

export { prisma } from "./src/client";
export type { Prisma, User, Todo } from
"./generated/prisma/client";

in the db src folder create client.ts

import { PrismaClient } from
"../generated/prisma/client";
import { PrismaPg } from "@prisma/adapter-pg";

const globalForPrisma = globalThis as typeof
globalThis & {
  prisma?: PrismaClient;
};

const adapter = new PrismaPg({
  connectionString: process.env.DATABASE_URL!,
});
```

```
export const prisma =
  globalForPrisma.prisma ?? new
PrismaClient({ adapter });

if (process.env.NODE_ENV ≠ "production") {
  globalForPrisma.prisma = prisma;
}
```

and then in package.json

```
  "exports":{
    "./client":"./index.ts"
  },
```

bun natively supports the ws server and we can create and run that

Add backend, ws, nextjs routes

Backend

- Create backend folder

```
cd apps
mkdir backend
cd backend
bun init
```

- Add express

```
bun install express @types/express
```

- Add db package

```
"dependencies": {
        "db": "*"
}
```

- do a global pnpm install

```
bun install
```

- Expose some endpoints

```
import express from "express";
import { prismaClient } from "db";
const app = express();
app.use(express.json());

app.get("/users", (req, res) => {
  prismaClient.user.findMany()
    .then(users => {
      res.json(users);
    })
    .catch(err => {
      res.status(500).json({ error: err.message });
    });
})

app.post("/user", (req, res) => {
  const { username, password } = req.body;

  if (!username || !password) {
    res.status(400).json({ error: "Username and password are required" });
    return
  }

  prismaClient.user.create({
    data: {
      username,
      password
    }
  })
    .then(user => {
      res.status(201).json(user);
    })
    .catch(err => {
      res.status(500).json({ error: err.message });
    });
})

app.listen(8080);
```

Websocket

- Create websocket folder

```
cd apps
mkdir websocket
cd websocket
bun init
```

- Add db package

```
"dependencies": {
        "db": "*"
}
```

- Do a global bun install

```
bun install
```

- Add code

```
import { prismaClient } from "db";

Bun.serve({
    port: 8081,
    fetch(req, server) {
      // upgrade the request to a WebSocket
      if (server.upgrade(req)) {
        return; // do not return a Response
      }
      return new Response("Upgrade failed", { status: 500 });
    },
    websocket: {
        message(ws, message) {
            prismaClient.user.create({
                data: {
                    username: Math.random().toString(),
                    password: Math.random().toString()
                }
            })
            ws.send(message);
        },
    },
});
```

in web folder also add the dependecies
"db":"*"

Update nextjs package to use the DB

- Update `package.json` to have a dependency to `db`.

```
"dependencies": {
    "@repo/ui": "*",
    "next": "^15.1.6",
    "react": "^19.0.0",
    "react-dom": "^19.0.0",
    "db": "*"
  },
```

- Update landing page to hit the `users` table to get back data.

```
import { prismaClient } from "db";

export default async function Home() {
  const users = await prismaClient.user.findMany();
  return (
    <div>
      {JSON.stringify(users)}
    </div>
  );
}
```

if some ts error do bun install
bun run index.ts
run this to test the backend and also copy the env file if some error is shown like scrms errors

try the websocket server using this
ws://localhost:8081 in the postman

for frontend also run the web file and try to test it for that also copy the .env file to the folder

for dockerfile in bun the format is little different so yaa its easier

in monorepo we cannot just copy the package.json first as there are many package.json in this project so to fix that we just copy everything

this was done for layer and image optimization

and when you do COPY . .
create a folder for the .dockerignore which should conhtain node_modules

bun can directly run ts file no need to build the code

we should not migrate the db in ci cd pipeline we can do prisma generate to generate to client just

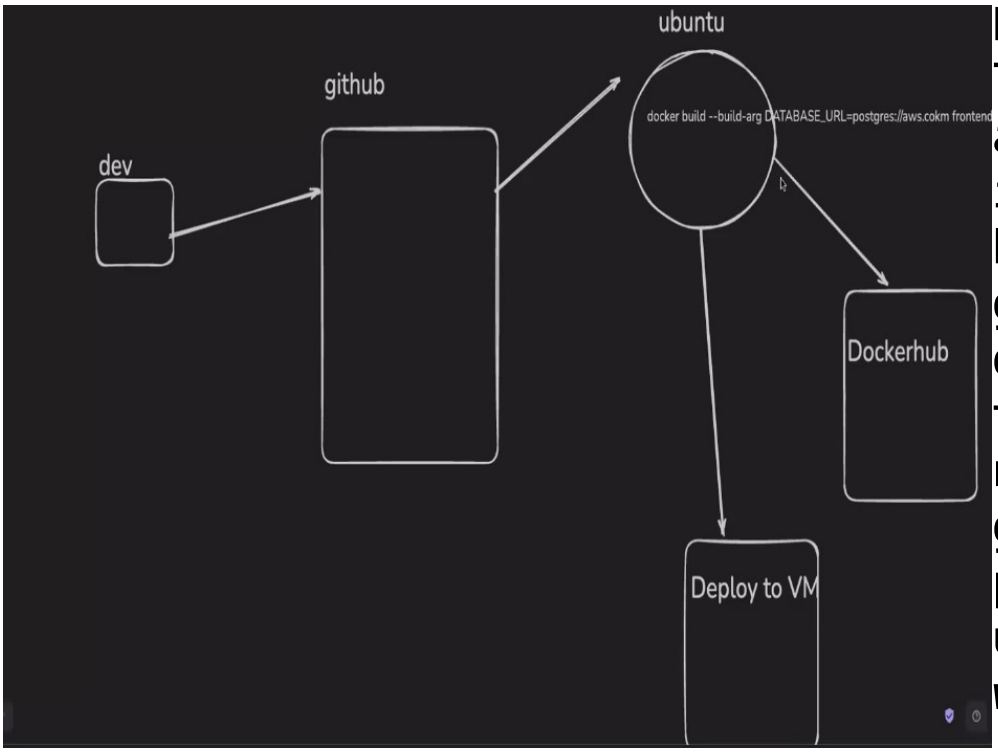
just for the frontend you have to build the project to for the js purpose and also defining some commands in the package.json

"db:migrate": "cd packages/db && npx prisma generate && cd ../..",
"start:web": "cd apps/web && bun run start",
"start:backend": "cd apps/backend && bun run start",
"start:websocket": "cd apps/websocket && bun run start"

the docker file we currently file is ok but kuch gandi chij hai next.js mai we need to fix that which is SSG (Static site generation)
like the frontend part currently we run the
bun run dev
but in production we will run
bun run start
so before doing that we have to
bun run build
and then the bun run start

WHY THE NEXTJS PROJECTS DOCKERFILE BECOMES TRICKY WHEN NEXTJS NEEDS TO TALK TO THE DB DURING THE BUILD PHASE?
Like see when we are in dev and we run bun run dev if any changes are done in the db then it can be updated on the reload but in the production we will first run the bun run build which do static site generation which means when the bun run start is done and if any changes are done in the backend even after restart there will be no changes in the page but the page which will be returned after the bun run start will be very fast but no new data. This can be fixed by rebuidling the project but its a specific usecase like in the notes 100xdevs website whenever harkirat adds the new notes he has to redeploy the website and then the new notes will be visible now

how to fix this??
1. you can force by a command that yeh page static nhi hona chaiye
export const dynamic = ‘force-dynamic’
in the page.tsx file
2. revalidate the page like every 60 seconds check for a data change (This is called incremental side generation)
export const revalidate = 60



BUILD ARGS
This can be taken as a input and used in github during building and getting the initial data
this variable will not be pushed to github this will be passed in the ubuntu machine and will be used

RUN DATABASE_URL=${DATABASE_URL} bun run build

to check the docker files are working or not
docker build -t apps -f .\docker\Dockerfile.backend .

For frontend
docker build --build-arg DATABASE_URL="postgres" -t apps -f .\docker\Dockerfile.frontend .
In this add a proper url currently just dummy

NOW COPIED THE DOCKER COMPOSE FILE

NOW SETTING UP THE CI CD PIPELINE

create .github /workflows in that create cd_backend.yml

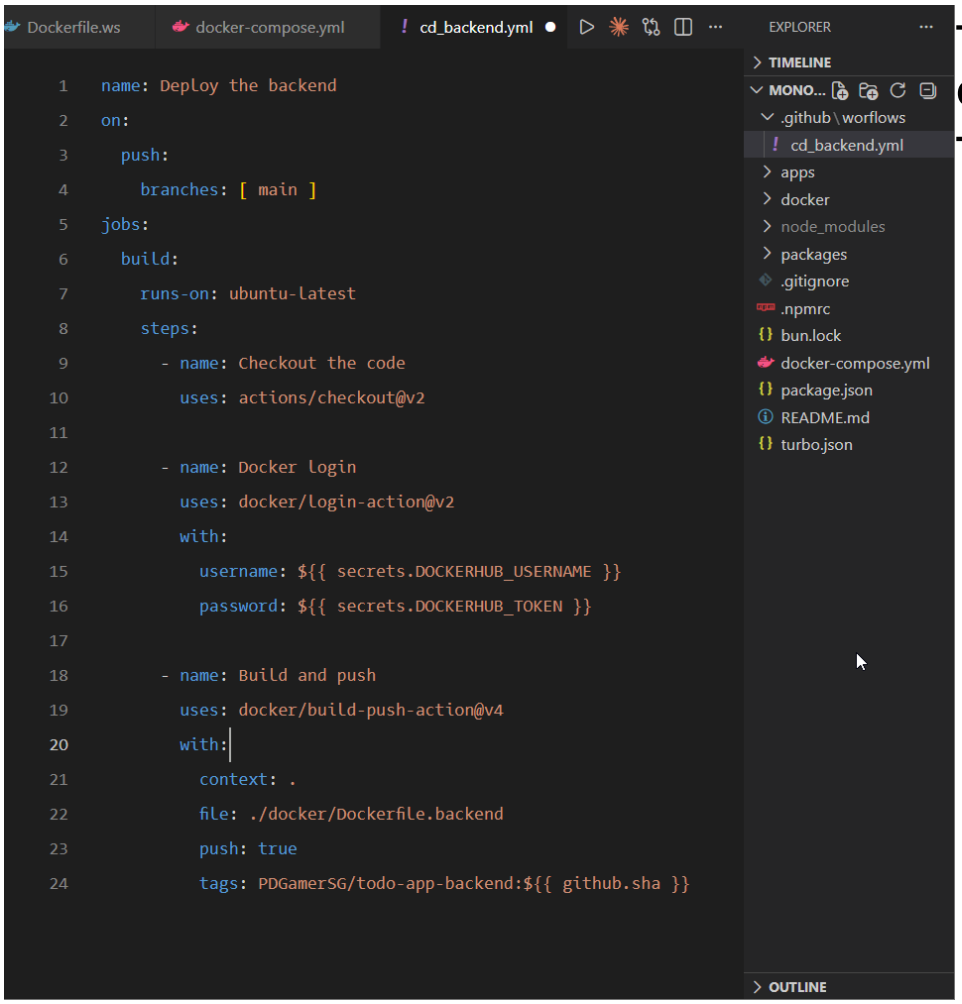
##build the docker image
##Push the docker image to docker hub
##SSH into our VM and start the new image

creating your own actions (GOOD TASK)

adds secret in the github actions for the docker helps in login with the docker to use the docker file which gets created whenever a update is pushed to the repo

whenever a docker new file is made we attach a tag which is like a version number, so instead of providing a value ourself github himself provide a sha value we can directly use that and balle balle

till the github.sha every step remains the same just the last part changes based on the thing we are using such as docker, gitops, kubernetes etc…



this is a docker compose file for the backend

while taking the token from the security use the read and write option to give access to the token

now after running the ci cd pipeline new dockerfile is shown in the website hub.dockerhub.com damnnnn

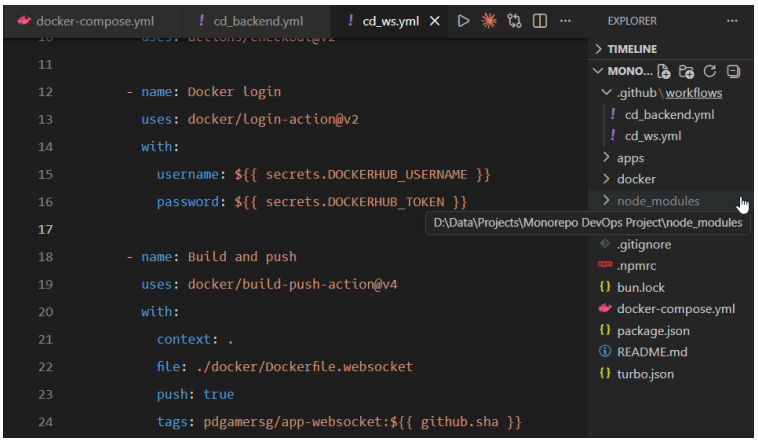
while pushing the code to the github check properly in the docker file it should be main or master as I was using master for this

in tags section in the repo it will show all the versions made also

its simply like
you make changes to the code → the code is pushed and the docker file is made → now the person can use the file and do the work

AND ALSO KEEP IN MIND IF THE REPO IS CREATED BY THE CI CD PIPELINE MAKE IT PRIVATE

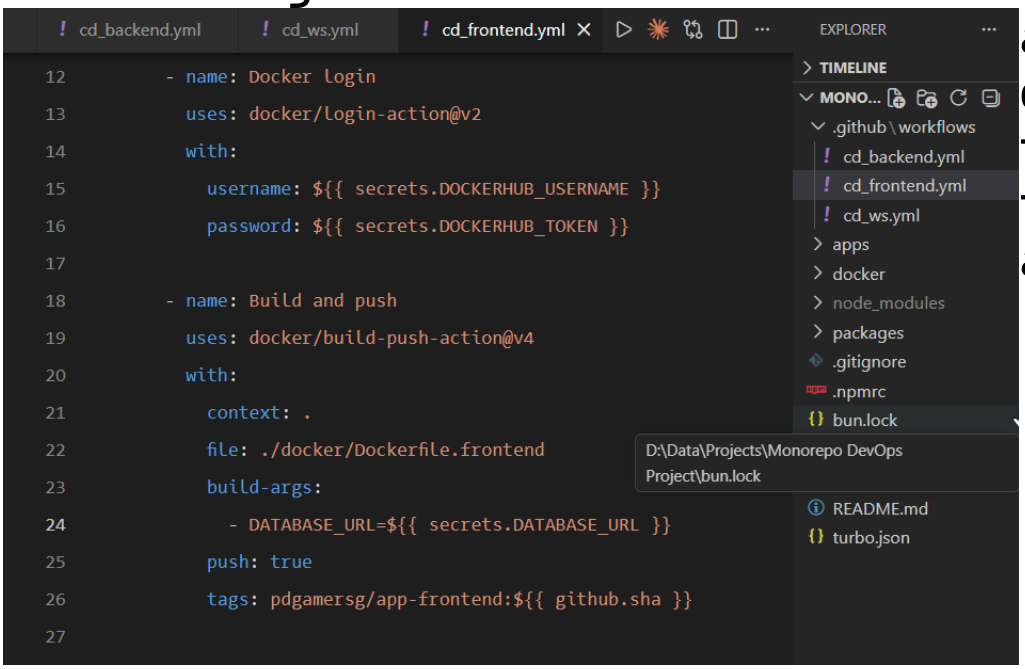
for other two files just the names will change
the tag name and the file name



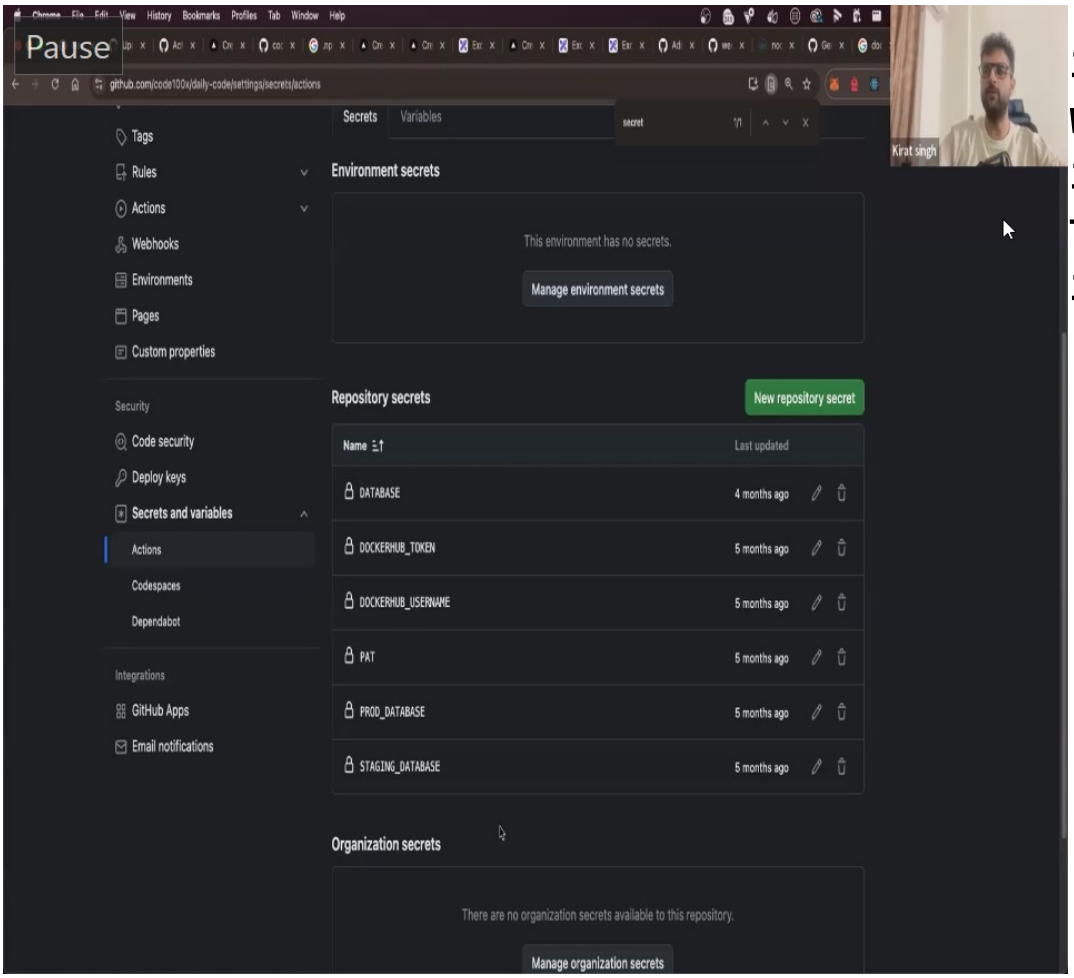
now next stuff is pushing to vm part

and in frontend we need to pass some args also

add the args like this



also store the database url in the secrets of the github actions settings



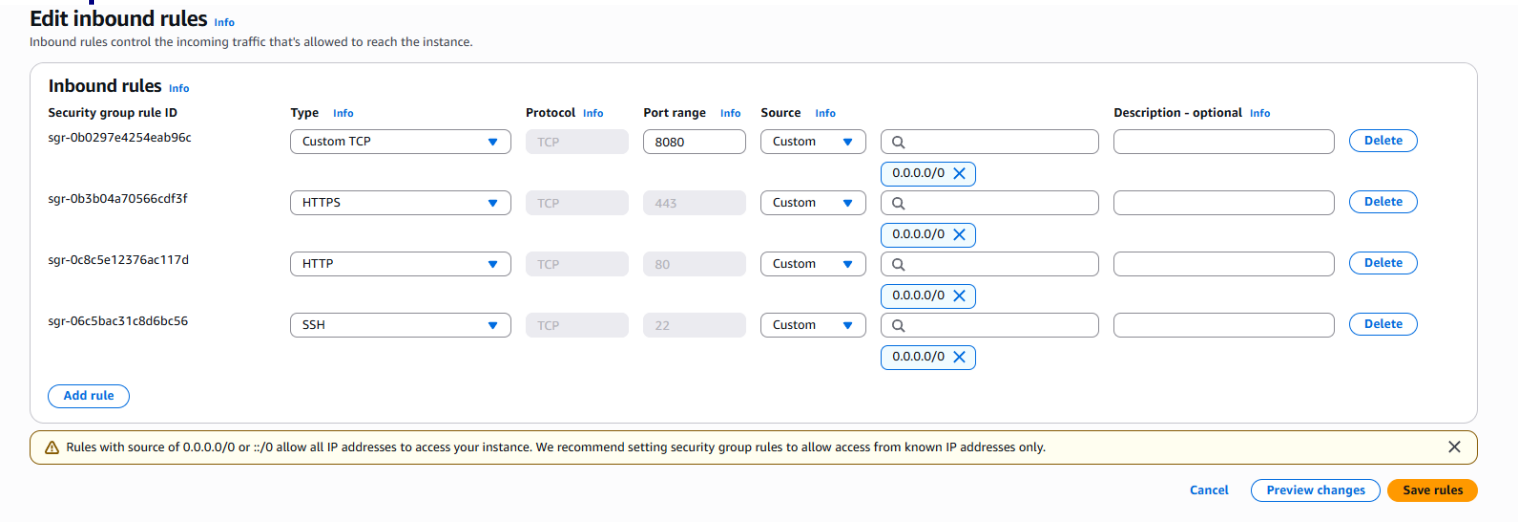
in actual websites also its done like this only see its soooo cool

in the keys folder use this
ssh -i pdkey.pem ubuntu@13.60.60.83
while creating check properly the os

just install docker in the ubuntu

from any website install and then copy the repo name and then use
docker run -p 8080:8080 pdgamersg/app-backend:tag_number

CHANGE THIS PORT ALSO TO MAKE IT WORK ON THE http://13.60.60.83:8080/




```
docker run -d -p 8080:8080 pdgamersg/app-backend:aaa86de3d36c81dcb82eeba9014eda07eed7268d
```

in this the -d helps in running the process in detaches mode which means it can run in the background also

ssh key can be created using
ssh-keygen

for ubuntu
cd ~/.ssh/
cat authorized_keys
to see who are are allowed
now to create a new one to give access
first in the .ssh folder create using
ssh-keygen
do cat name the .pub one and copy to the authorized keys in the ubuntu machine
then copy the private one and paste in the github actions secrets

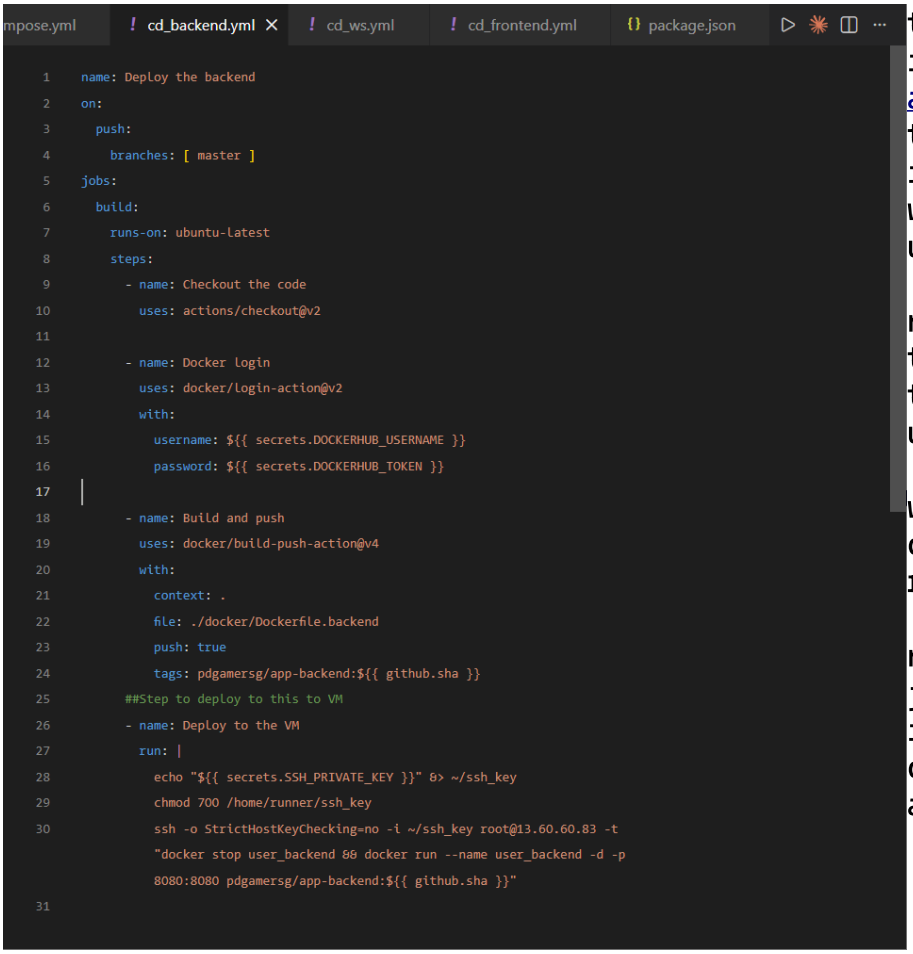
now the ci cd pipeline can access the key also
sudo docker run --name user_backend -d -p 8080:8080 pdgamersg/app-backend:31c9f731fd25d89e6a050cb77096b5133a719a8d

name this also because whenever next time the new docker file will run it should have the same name so that it can stop that

HOW TURBOREPO MAKES MONEY?? USING REMOTE CACHING (they make it safe)

ITS VERY DIFFICULT TO VALIDATE THE YML FILE
USE SOMETHING CALLED YML VALIDATOR
yml to json convertor (for easy view)

EXPLANATION



this checkout the code will be seen in the github actions the actions/checkout@v2
this code helps in cloning the data in the ubuntu machine this code is written by someone we are just using that

now this docker login is done by the creds which we have stored in the github actions secrets it can use that and login

we can also write code for this ourself but why if already written reuse it just

now the build and push this its just building and then using that image and asking if we should push or not and then tag it like giving a version name

what is this StrictHostKeyChecking?
This is like whenever we first time try to connect to system it does like ask for yes or no to skip that we are using this thing as no

now the last line is just ssh into the machine and stoping the old task and then running the new task with the same name before and then the website is up again lets goooooooooooo

FOR STATIC PAGES WE SHOULD NOT USE DOCKER AND SHOULD NOT DEPLOY ON VM
DO IT ON A CDN DIFFERENCE SHOWN IN S3 AND CLOUDFARE

STATIC MEANS NEVER CHANGE YOU SHOULD CACHE THEM VERY HEAVLY ON A CONTENT DELIEVERY NETWORK (CDN) , STATIC SHOULD BE DEPLOYED ON A CDN MOSTLY

NOW WHENEVER YOU RUN THIS AGAIN YOU HAVE TO CHANGE THE IP MULTIPLE PLACES
FIRST IN THE YML FILES, THEN MAYBE YOU HAVE TO ADD AGAIN THE SSH KEYS I HAVE EXPLAINED ABOVE

CREATE A GOOD PROJECT AROUND THIS YOU CAN I BELIEVE YOU

NOTHING ELSE ALL GOOD