**Week 12**

**12.1 Deploying Frontends on AWS**

* Objects are anything that are not stored on the database like mp4, html, css, jpeg files etc.
* We store objects in object stores. S3 is one such object store. Simple Storage Service – S3
* At scale, deploying frontend on AWS is much cheaper than Vercel.
* CDNs – Content Delivery Networks. For distribution. Cloudfront.
* CDNs manage a lot of pops (point of presence) so that when a request goes for a file, the files get cached in that specific pop for a certain time period. Pops are live smaller servers.
* All requests to object stores go through CDN.
* Usually distribution cost is higher, so you ask for discount on CDNs.
* For backends, edge networks are used which is deploying backend applications over multiple servers. The do not use caching as each user requests a unique data.
* We do not use edge networks for storage as the files are very large.
* npm run build to get HTML, CSS and JS files from the react project.
* serve serves all of your file over a port.
* We want to block public access because we don’t want our files to be distributed through S3 as it is expensive. We want it be distributed through Cloudfront.
* Generate the policy from Cloudfront and edit the bucket policy in S3 to allow Cloundfront to access S3.
* Find cheap domains on namecheap.com
* You can add alternate domain names through Cloudfront.
* You can also generate free SSL certificate for your new domain name through AWS or put one bought from GoDaddy etc. Amazon would verify by giving you a specific domain to put in your DNS
* Finally you need to put you new domain in the DNS pointing to the Cloudfront URL.
* You can also add error pages if the user tries to hit an invalid page through Cloudfront.
* To remove all caches from the Cloudfront pops, you can go to Invalidations and add /\*
* CI/CD pipeline whenever you push your code to GitHub it automatically deploys to S3.
* **Summary:** Deployed frontend on AWS and connected custom domain – Only HTML/CSS/JS files were uploaded to S3. Distribution through CDN – Cloudfront. SSL for custom domain requested through AWS, verified, new domain pointed to Cloudfront URL, added error page.

**12.2 Advance Typescript APIs**

* Pick – lets you select a set of properties from an existing type or interface.
* Partial
* Readonly and readonly.
* Record and Map.
* Exclude.
* Zod - export type finalUserSchema = z.infer<typeof userProfileSchema>;
* Objects and maps are two ways to do key value pairs.

**Deploying NPM Packages and Intro to Monorepos**

* Zod inference so that we can verify inputs at the frontend but the types is inferred from the backend.
* You never publish the src folder and you exclude it by adding “src” to .npmignore using vi.
* .d.ts file just contains the declarations, inferences, and types. No JS logic.
* Whenever you publish your package, you also need to publish .d.ts file so that the user using your package can get the types. Add “declaration”: true in tsconfig.json and compile again to get the .d.ts file. Imagine this file as a header file that shows all the return types, the input types, interfaces etc.
* No one know publish npm packages to get code from backend to frontend or vice versa. We use mono repos now.
* Turborepo is one mono repo. Just create components in ui folder and you can import them in your frontend because frontend’s package.json has “ui”: “\*” as a dependency.

**12.3 Actionable Docker**

* Container is like a minicomputer with their own networks and file system. running inside your computer.
* Images are like code/package that people put out in the world for other people to use.
* Docker hub contains all the images.
* A container is an image in execution.
* Mongo runs on port 27017.
* docker run -p 27017:27017 mongo
* docker run -d -p 27017:27017 – for running in detach mode, so that it runs in the background and does not show any logs.
* docker ps – for checking which containers are running.
* docker kill container\_id for killing the container.
* docker run -e POSTGRES\_PASSWORD=mysecretpassword -d -p 5432:5432 postgres – e stands for environment variable. The container that starts has this password. You always have to pass this environment variable. However, POSTGRES\_USER and POSTGRES\_DB are optional and take the default value of postgres when not passed.
* postgresql://postgres:mysecretpassword@localhost:5432/postgres
* You have both postgresql locally installed as well as container running, when you try to connect through psql, psql will try to connect to postgresql running locally if both are running. Turn off local through brew services stop postrgresql to connect to container.

**12.4 SQL Relationships and Joins**

* psql is a CLI that lets you connect to the database.
* docker exec -it container\_id /bin/bash – ssh into the container and once you’re in the container you can run psql commands without installing psql locally on your computer.
* On delete restrict won’t let you delete the user until you have deleted their addresses. On delete cascade deletes the addresses of the user when you delete the user.

**12.5 Prisma Recap and Relationships in Prisma**

* Prisma lets you connect to any database while you write the same syntax, gives you types when you run the query, provides you automatic migrations file.

**12.6 Connecting Pooling in Serverless envs**

* Workers connect to the connection pool and then the connection pool connects to the database because database restricts the amount of connections.
* Engine is one such dependency of prisma that only works with NodeJS. So, prisma requires you to connect to a connection pool.
* [Deploy to Cloudflare Workers & Pages | Prisma Documentation](https://www.prisma.io/docs/orm/prisma-client/deployment/edge/deploy-to-cloudflare) and [Getting started with Prisma Accelerate | Prisma Documentation](https://www.prisma.io/docs/accelerate/getting-started)
* npm create cloudflare@latest
* npm install --save-dev prisma
* npx prisma init
* Ideally, you should have your connection pool created very near to the database so you select the same location as the database.
* Go to prisma accelerate, set up a new project and enable accelerate, and generate API key. This API key should be put in your wrangler.toml file so that it connects to the connection pool when deployed. The normal neon db url should be in your env file as DATABASE\_URL and DIRECT\_URL and schema.prisma should have url and directUrl.
* All environment variables you need in CLI should be in .env while all environment variables needed by index.ts should be in wrangle.toml.
* npm install @prisma/extension-accelerate
* npx prisma generate --no-engine. No engine basically means to skip things that we don’t need in the cloud environment. Decreases the size of the bundle.
* npm run deploy