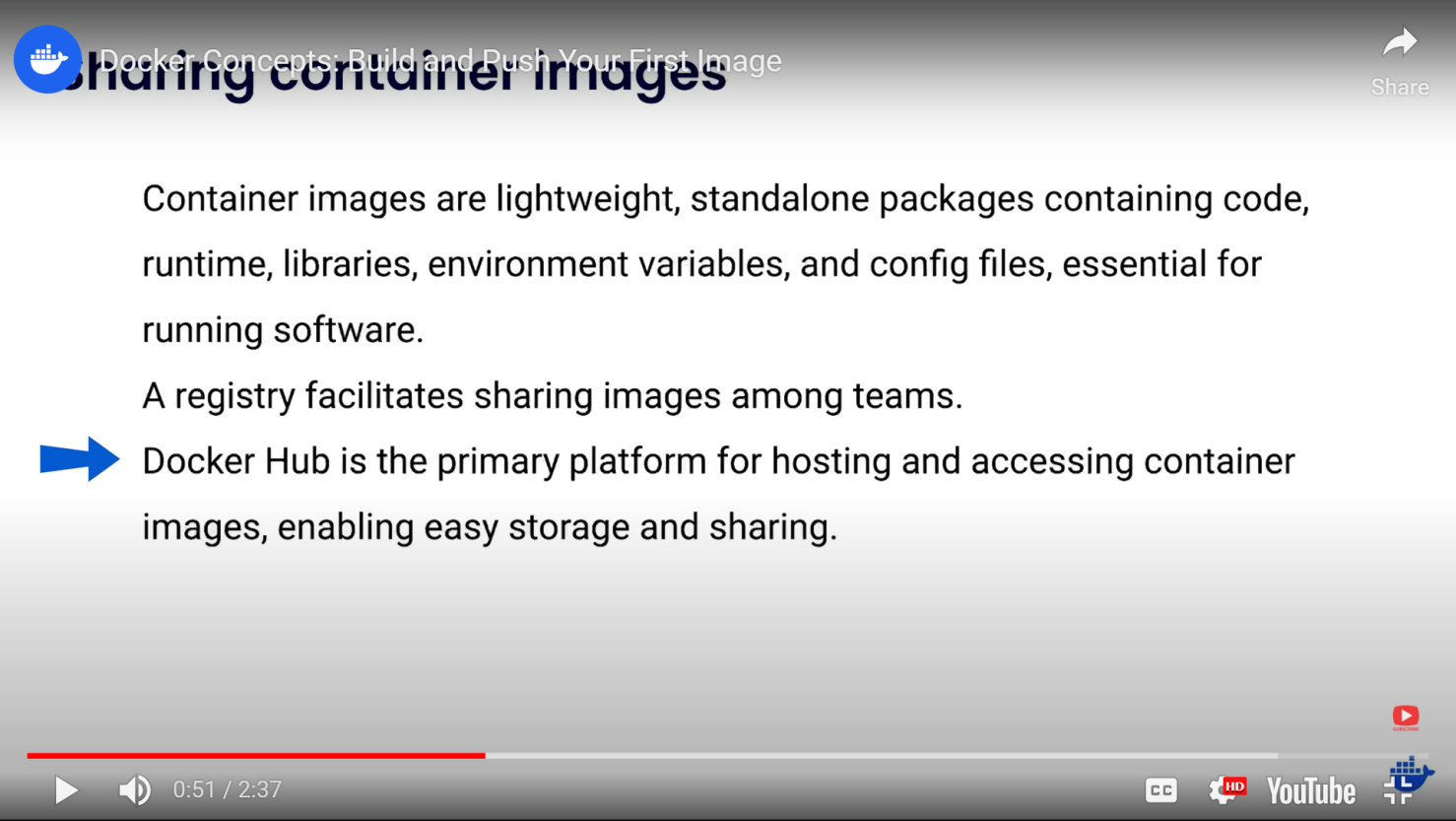
**Docker Daemon –** does all the work(image pull, creation of image, etc)

Images & instances( containers ) 🡪 One can create multiple instances/ containers of an image(can be considered as class & object)



Containers( these containers can be created with the help of images )



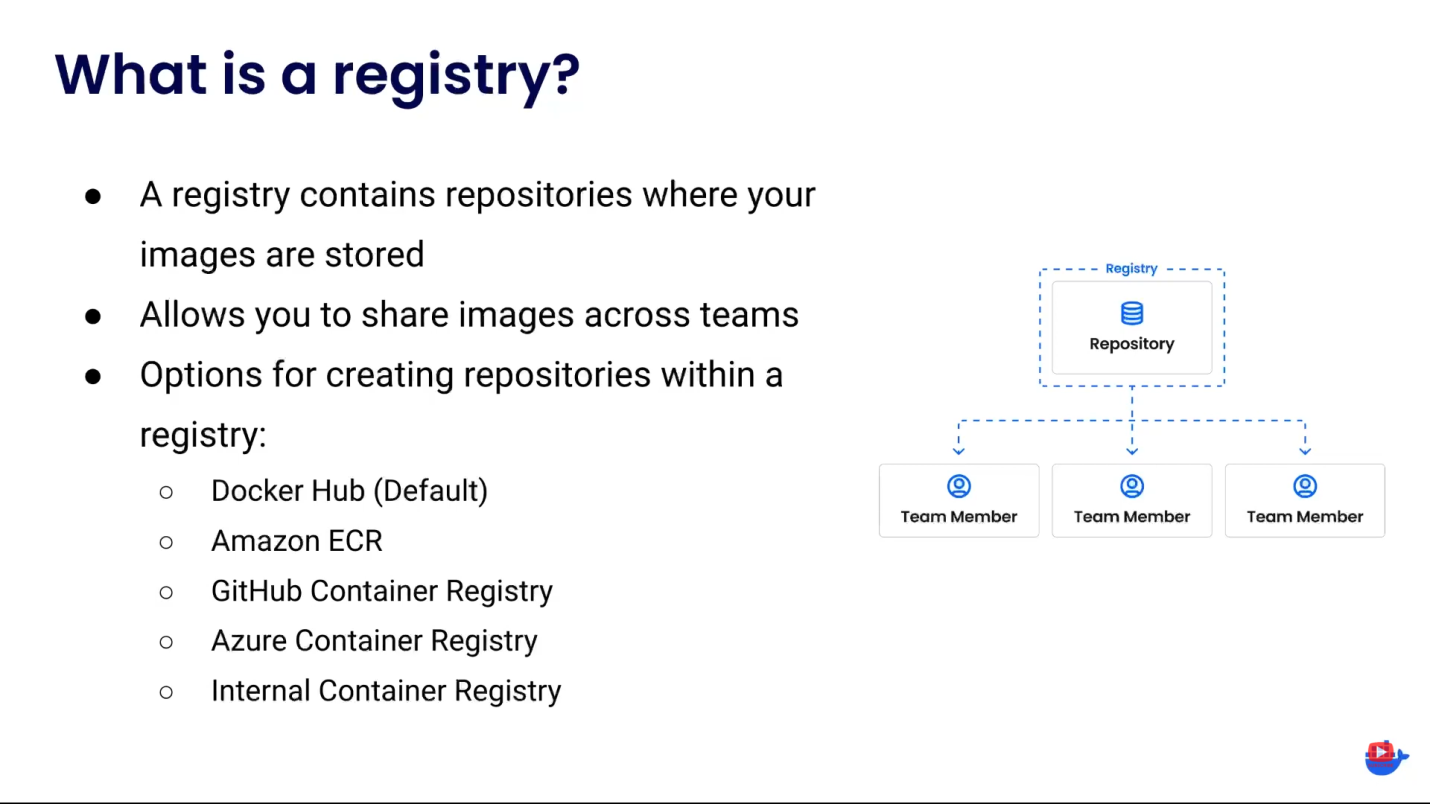
**What is a container?**

How do you ensure the version of Python (or Node or the database) your app needs isn't affected by what's already on your machine? How do you manage potential conflicts?Containers are isolated processes for each of your app's components. Each component - the frontend React app, the Python API engine, and the database - runs in its own isolated environment, completely isolated from everything else on your machine.

Seeing a [container](https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-a-container/) is an isolated process, where does it get its files and configuration? How do you share those environments?

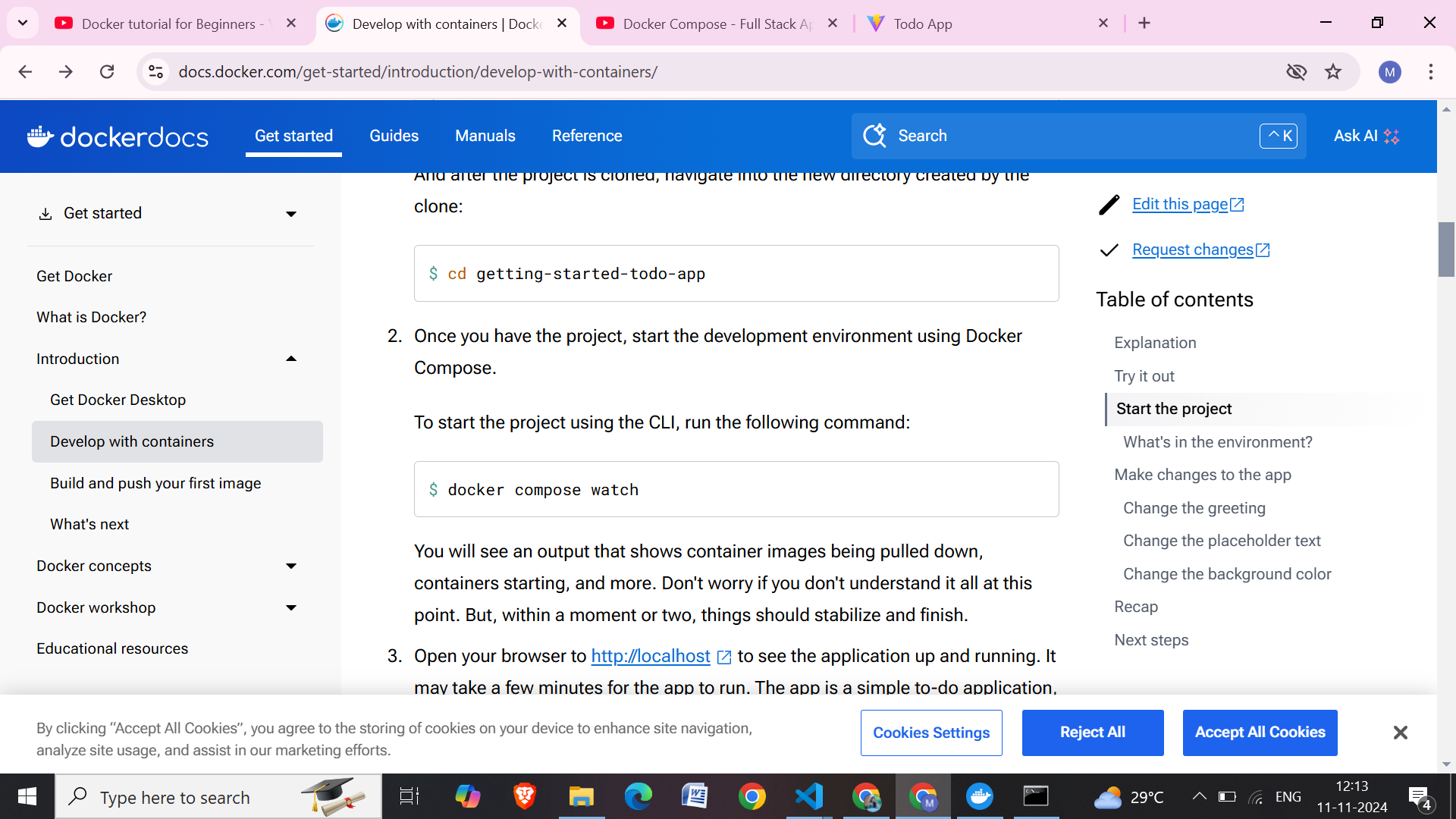
**What is a images? 🡪** We require image to run container. Images are immutable. A container image is a standardized package that includes all of the files, binaries, libraries, and configurations to run a container.

Container images are composed of layers. Each layer represents a set of file system changes that add, remove, or modify files.



Without **registry** there is still a way to create containers but there is no way to share images with the team members. Registry can contain many repositories and repository can contain many images. A **registry**is a centralized location that stores and manages container images, whereas a**repository** is a collection of related container images within a registry. Think of it as a folder where you organize your images based on projects. Each repository contains one or more container images.

With **Docker Compose**, you can define all of your containers and their configurations in a single YAML file. If you include this file in your code repository, anyone that clones your repository can get up and running with a single command.



**Commands**

* **-it 🡪 interactive,  
  -p 🡪 port number,   
  -e 🡪 environment variables**
* **docker run –it ubuntu** //I want to run ubuntu image in my machine

1. Ckecks if ubuntu image is already downloaded,
2. If yes, it will use the image to create a new container
3. If not, it will be download its image first

* **docker container ls** //lists all the running containers
* **docker exec –it *CONTAINER\_NAME* bash**// docker execute interactive   
  -it 🡪 This will connect the terminal with the container(doesn’t gets disconnected)
* **docker images**
* **docker run –it –p 1025:1025** *IMAGE\_NAME* //expose containers port 1025 : to my machine’s port 1025 (-p *MYMACHINE\_PORT : CONTAINER\_PORT*)
* **docker run –it –p 1025:1025** **-e** *KEY=VALUE* **–e** *KEY=VALUE –*network=*NETWORK\_NAME IMAGE\_NAME*
* **docker build –t** *IMAGE\_NAME DOCKERFILE\_PATH  
  EX-* **docker build –t** *youtube . // tagname to convert your application to an image in my local machine*
* **docker run –it** *youtube*
* **docker compose up //***pull all the configurations in docker-compose.yml file and make it up and running*
* **docker compose -f infra.yml up –d //***if we have different file name other than docker-compose.yml(here infra.yml)*
* **docker build -t manideepa18/trial . //***this is for pushing the image to the hub. 1. Create a repository, 2. Build an image with the same name 3.*
* **docker network create mern //***to create a network so that application within the same network could easily communicate*
* **VOLUMES 🡺** we do this to make sure that our container’s particular folder will have access to our specified host machine’s folder. So if a container even gets deleted we can again create a container and mount it to our host machine’s folder and nothing will be lost.

**Ex- docker run –it –v** *MY\_MACHINE\_FOLDER:CONTAINER\_FOLDER IMAGE\_NAME*

**How To create docker File**

* Create an OS // FROM ubuntu
* Install whatever you want to install there //RUN ap install nodejs
* Copy files // COPY src dest(same names preferred)
* RUN npm install
* Set entrypoint, i.e., whenever this container gets run the command in it will be run // ENTRYPOINT [“node”, “main.js”]

**If I want to run multiple containers at a time(use compose)**

* **Make docker-compose.yml file**
* Version : ‘3.8’ //docker compose version
* Services:

Postgres:

Image: postgres //this will be pulled from hub.docker.com

Ports:

- ‘8000:8000’

Environment:

- name=value

**Redis:**

Image: redis

Ports:

- ‘6379:6379’