Up to now we know how to create and run the Docker Images & containers, now we are going little in-depth to deal different kinds of data which resides on Images, Containers and Volumes using arguments & Environment variables.

There are different kinds of data that we need to deal and we understand what of kind of data that we are dealing with it.

Up to now we had only deal with Application (Code + Environment), it was add to the image while building the docker image and it was read only. Application (Code + Environment) store on Read-only, hence stored in Images.

Another type of data that we need deal is “Temporary App Data”, it won’t deal with the Application code + environment but it will deal with the data that was generated by application at run time temporarily. We might store this type of data in a variables or in a database and can be lost if the container shutdown. This data can be stored on top of the extra layer ie., running container not on Images.

Last type of data that we need deal is “Permanent App data” ex: User accounts. We can store this data in files or in a database. We should not loose data once the container shuts down, so the data should be read-write on containers permanently on volumes.

So In feature we need to deal with the docker volumes a lot.

Permanent App Data (e.g. user accounts)

Temporary App Data (e.g. entered user input)

Application (Code + Environment)

Read + Write, permanent, stored with Containers & Volumes

Must not be lost if container stops / restarts

Read + Write, temporary, hence stored in containers

Read-only, hence stored in Images

Added to image and container in build phase

Written & provided by you (= the developer)

Dynamic and changing, but cleared regularly

“Fixed”: Can’t be changes once image is built

Stored in files or a database

Stored in memory or temporary files

Fetched / Produced in running container

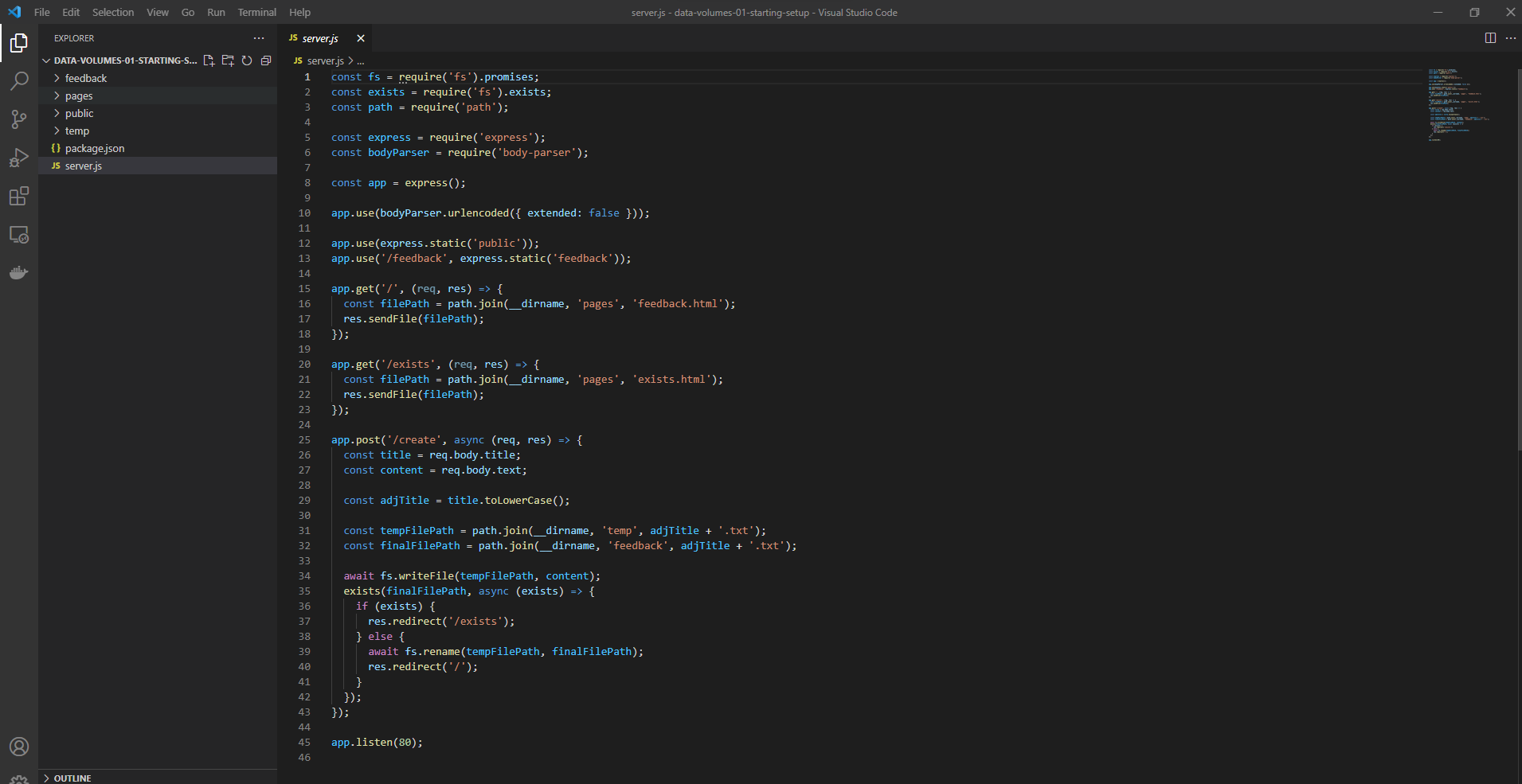
Fetched / Produced in running container

## Dockerize with real time NodeJS Application

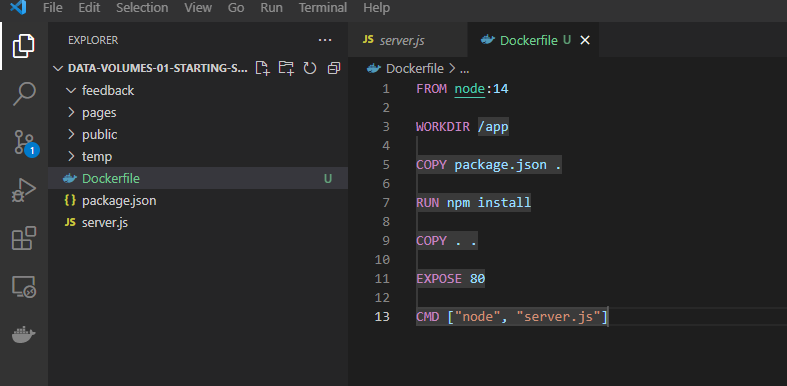
Now we are going a build a NodeJS Application, which will accept a feedback and store in a temporary files initially, once user submits the feedback it will store the files permanently on feedback folder.

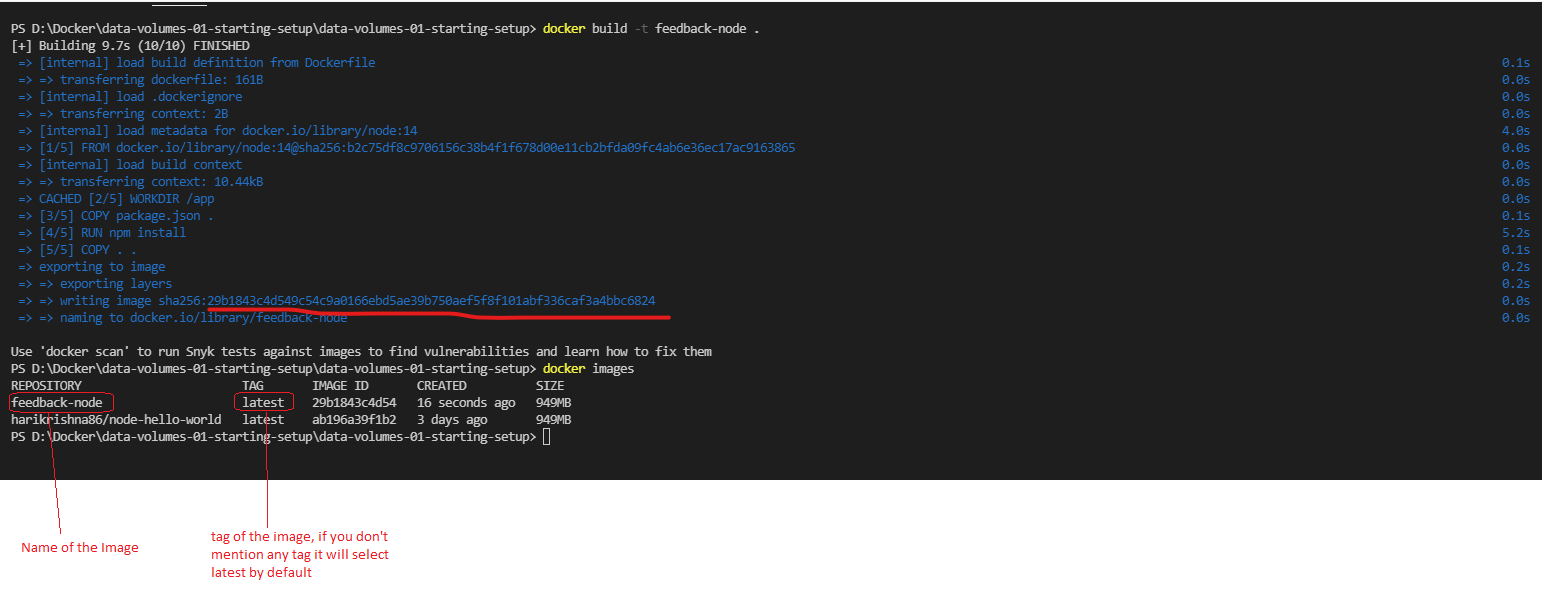
Download the code from the following location.

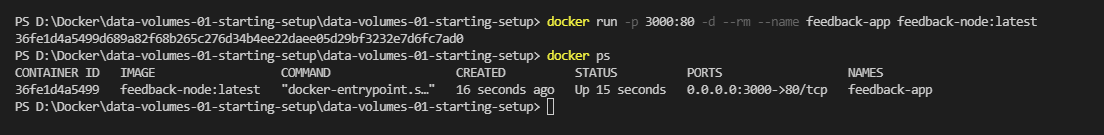
<https://github.com/harikrishna83/Docker-firstdemo-handson/tree/master/data-volumes-01-starting-setup>

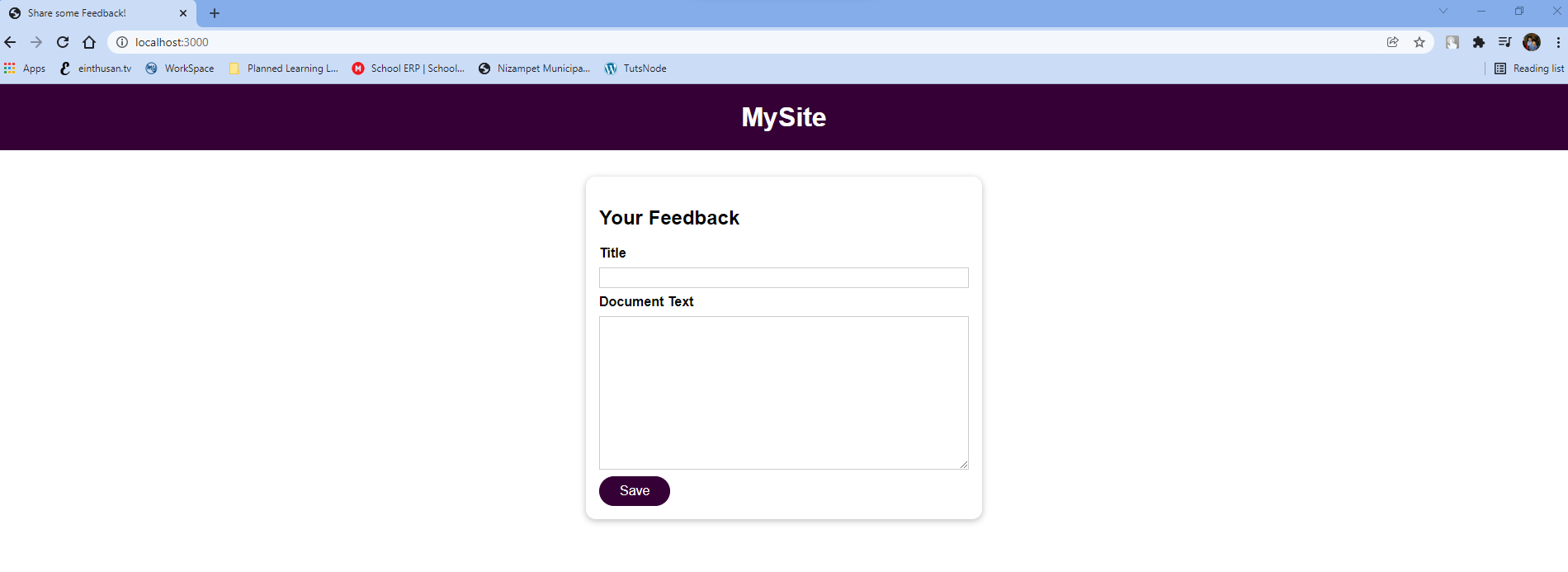


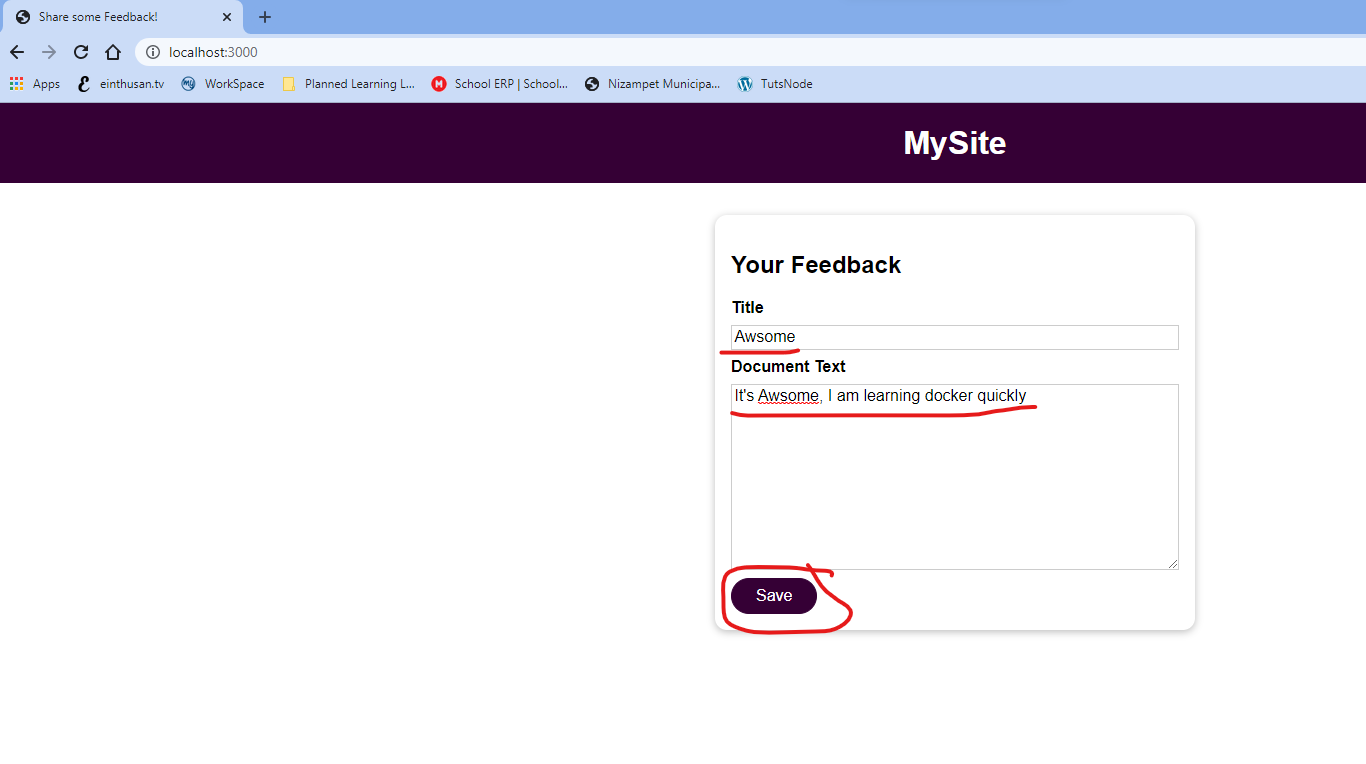
Now let’s get dockerize this app by creating a Dockerfile.









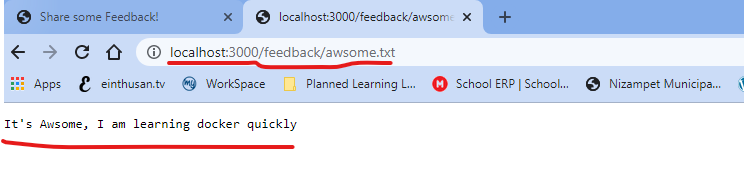


As per the NodeJS application, whatever you enter on title section it will create with that name so make sure you keep the title as simple don’t mention any special characters.

Note: if you enter any capital letters, as per the code it will convert all into small characters and save it.

As I enter awsome on text, if you want to view the feedback we need to browse the below url, there is no other way to browse it.

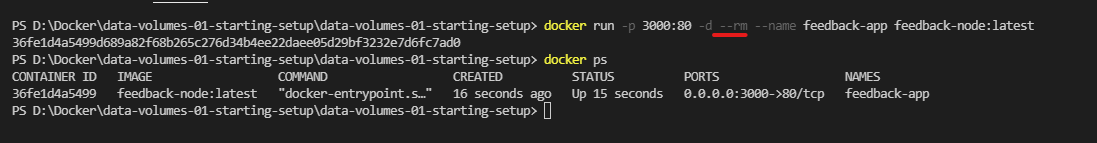
<http://localhost:3000/feedback/awsome.txt>



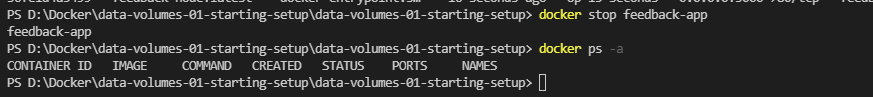
We can’t see the feedback in a files, because once the instruction of “COPY ..” on Dockerfile as there is no connection to your local file system to docker container, and the image is read only, we can’t be able to copy the files locally from your machine to the docker container / images. It only stored in a temporary files and be able to browser [http://localhost/feedback/<filename.txt](http://localhost/feedback/%3cfilename.txt)>

That is why on previously, whenever we change the code we re-builded the image to reflect it. This problem can be addressed in feature by using docker volumes to store the file permanently.

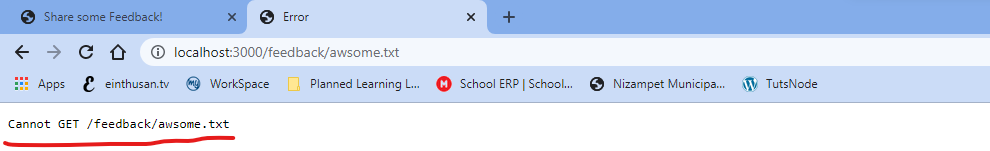
## Understanding the Problem now



As we are running the container by using the option --rm, if we stop the container it will automatically remove the container and you will loose the feedback.

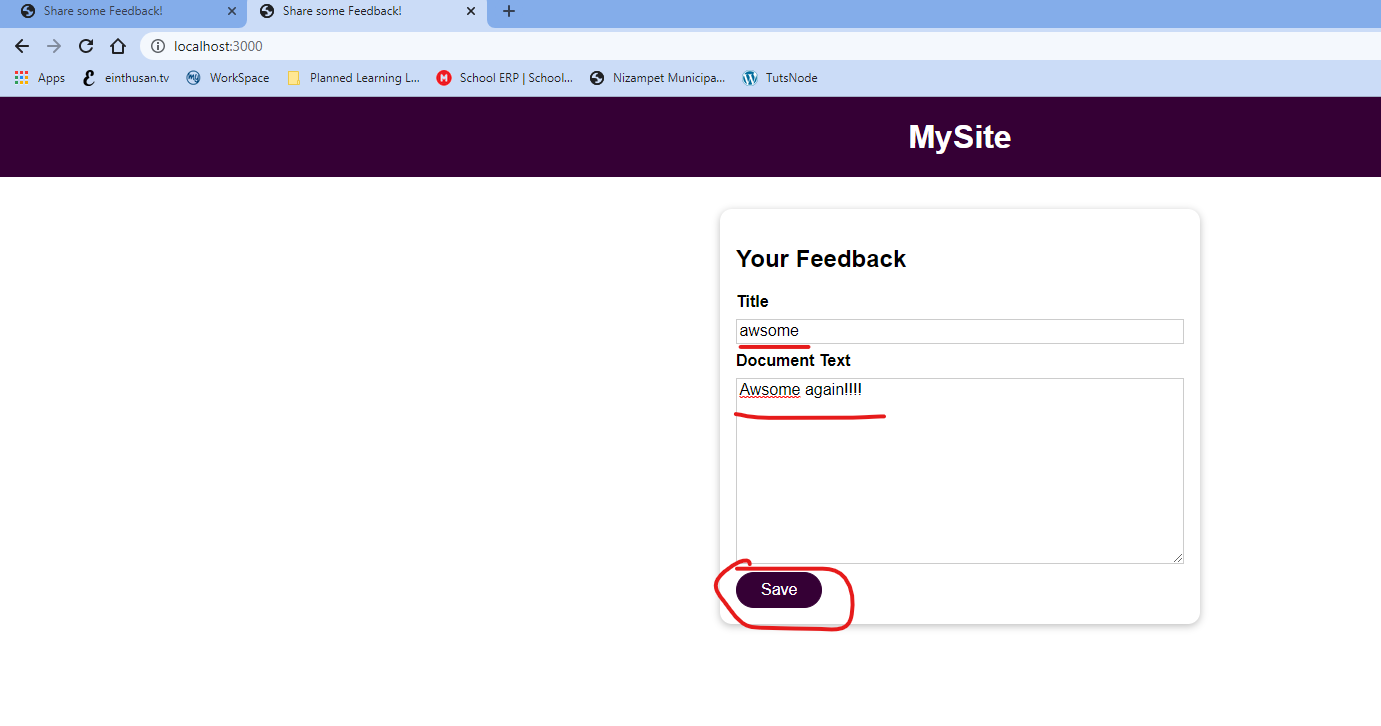


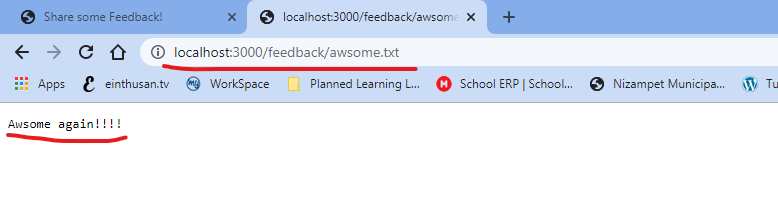




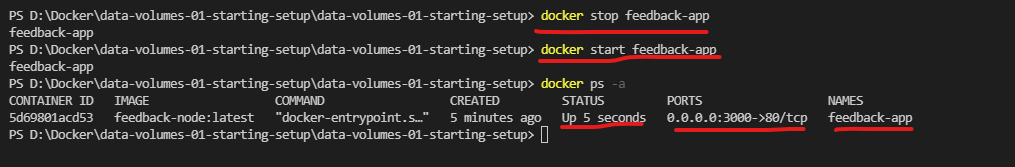
As you had re-created the container, all the feedback got erased now.

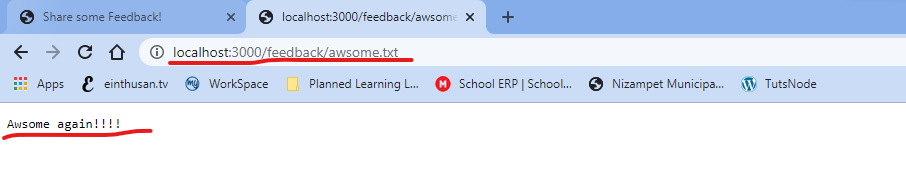
Suppose if you stop the container and start the container, it won’t erase the container filesystem. As container had read-write file system layer the feedback still exists.





Now let’s start and stop the containers and see if the feedback is available or not





As you can see after stop and start the container, the feedback still exists because the container file system layer is read-write and it won’t erase it start and stop the container. We will loose the feedback only if the container is lost or removed as container file system we have only an image and image filesystem is read only and fixed.

Now we understand the problem that we are facing, to deal with this problem docker had inbuilt feature called volumes, we will use this volumes and solve the problem.

Volumes are folder on your host machine, it’s not the folder on containers or images. Which basically means it made available on your local machine and mount that folder to your images / containers.

Volumes are connected inside your local machine to containers, suppose if you create a file on that volume from outside of your container it will reflect that file on the container and viceversa.

Therefore volumes persist even if we rebuild the containers, only thing is you need to mount the volume from your host to the container.

Volumes persist if a container shuts down. If a container (re-)starts and mounts a volume, any data inside of that volume is available in the container.

/app/user-data

Host (Your Computer)

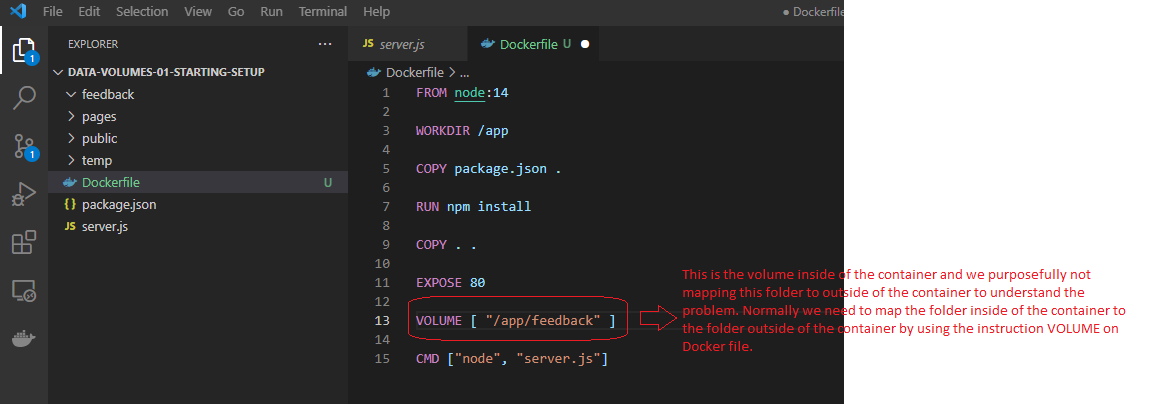
/some-path

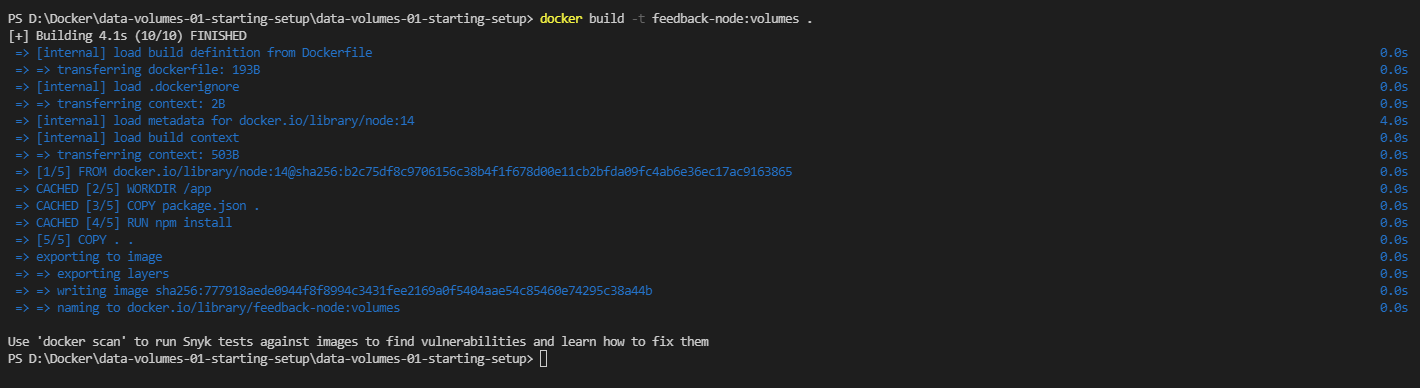
Volumes are folders on your host machine hard drive which are mounted (“made available”, mapped) into containers

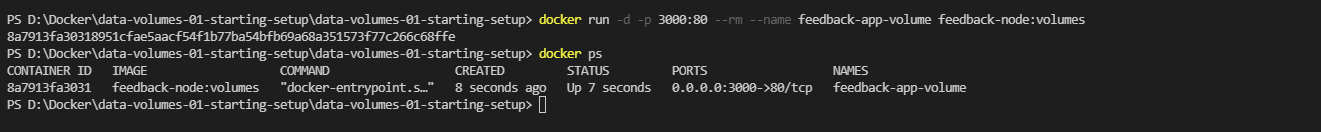
A container can write data into a volume and read data from it.

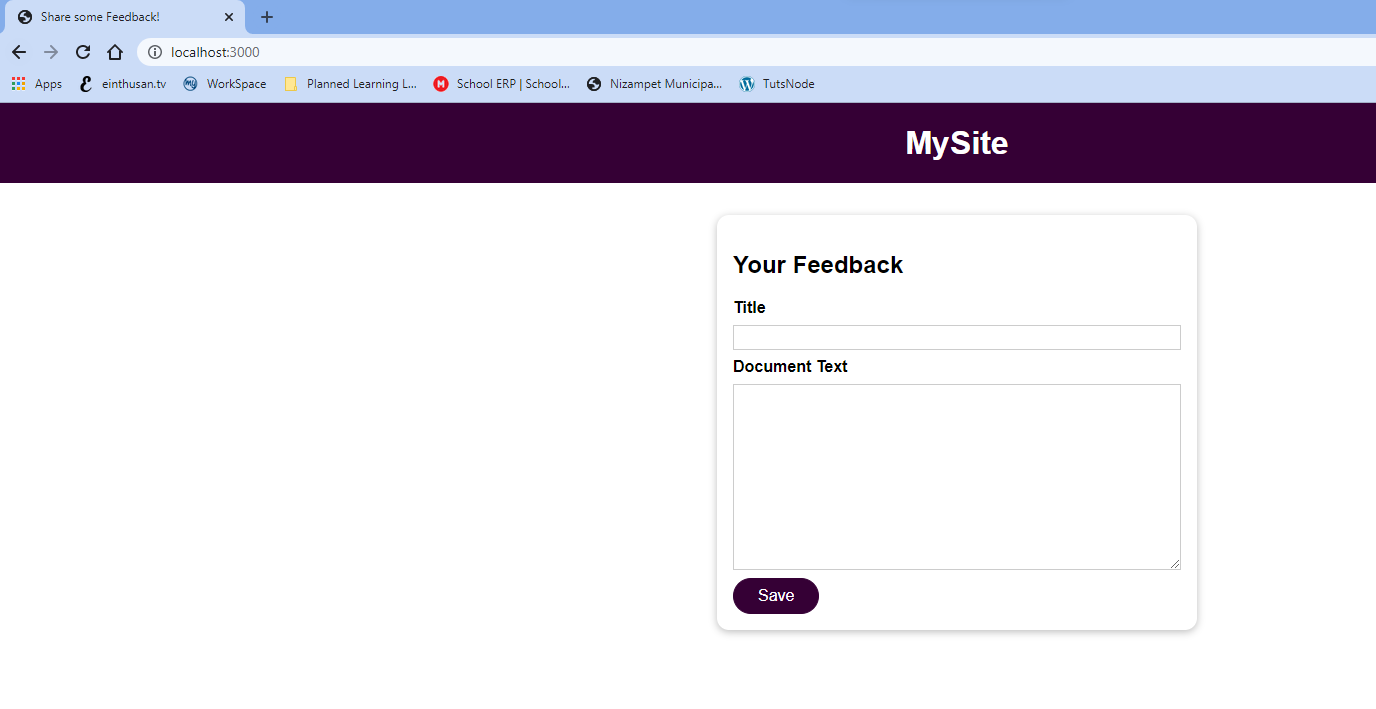
Now let’s try by adding volume to our container by modifying the Dockerfile.

As per the NodeJS Code, I am saving all the feedback files to a folder called feedback and temp files to another directory temp but these temp need not to be saved.

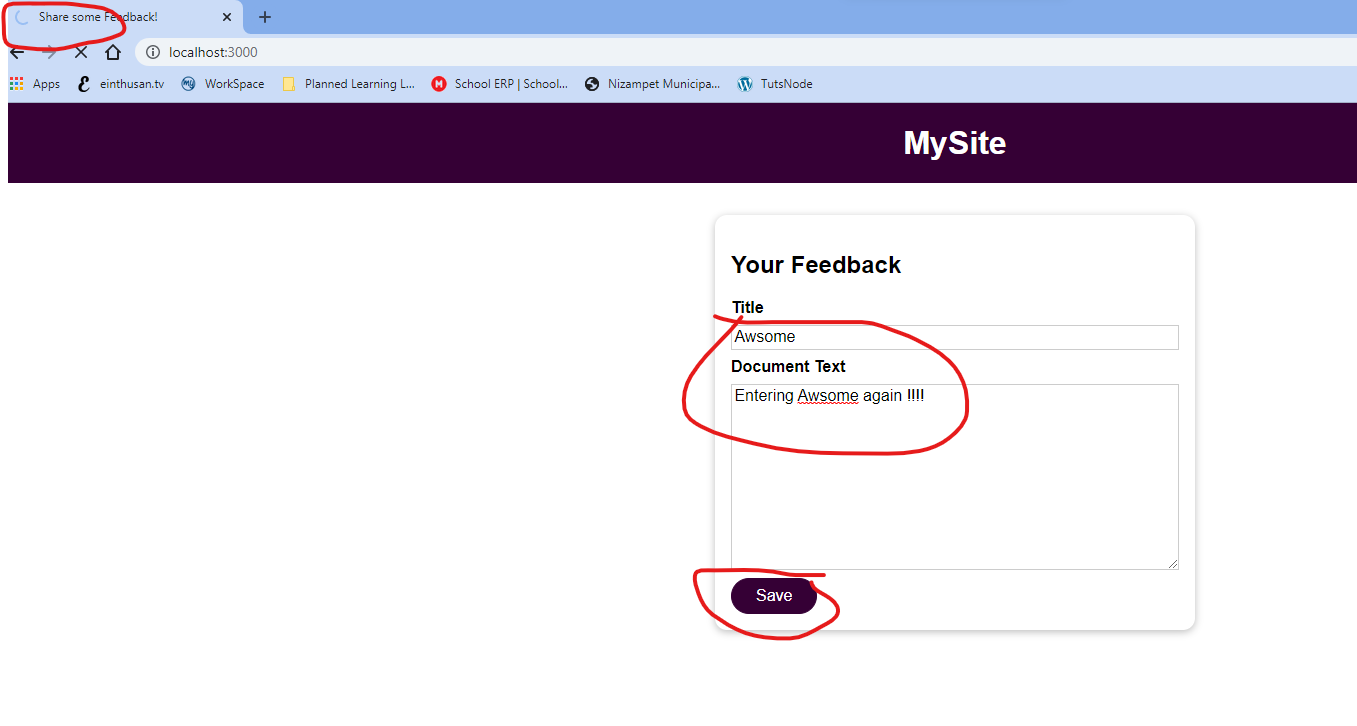
As our whole application resides under /app, I am choosing the volume inside my container as “/app/feedback”, and this folder should be mapped to outside of the container to some path inorder to store the data permanently.



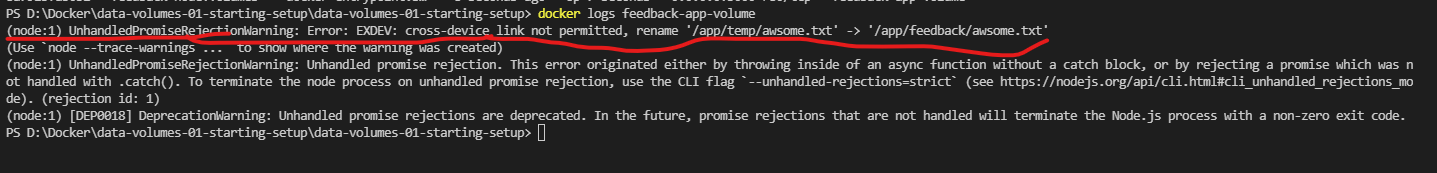




Now let’s try to save the feed and see if it will save or not, as we added the volume on the container.



It will not save the code, it will keep on try for some time and crash the node. In order to see what went wrong let’s issue the command “docker logs feedback-app-volume”

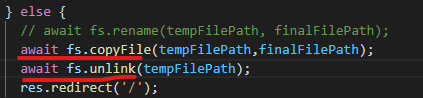


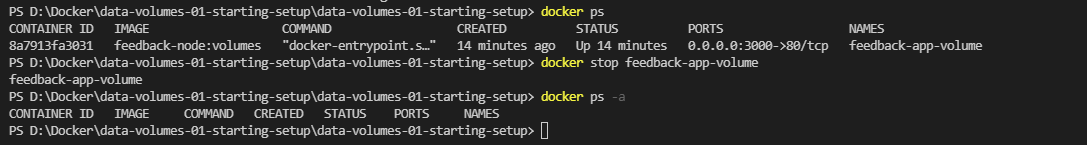
The error says cross-device link is not permitted, which means when it tries to copy the feedback from temp folder to /app/feedback folder this app crashed.

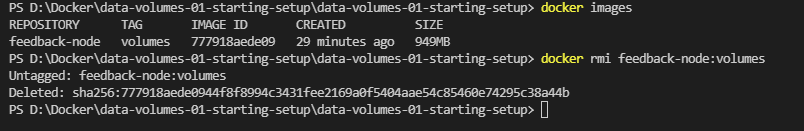
Here the problem is on the server.js rename method, this rename method doesn’t allows to copy the file from inside of the container to the outside path of the container (Volume).

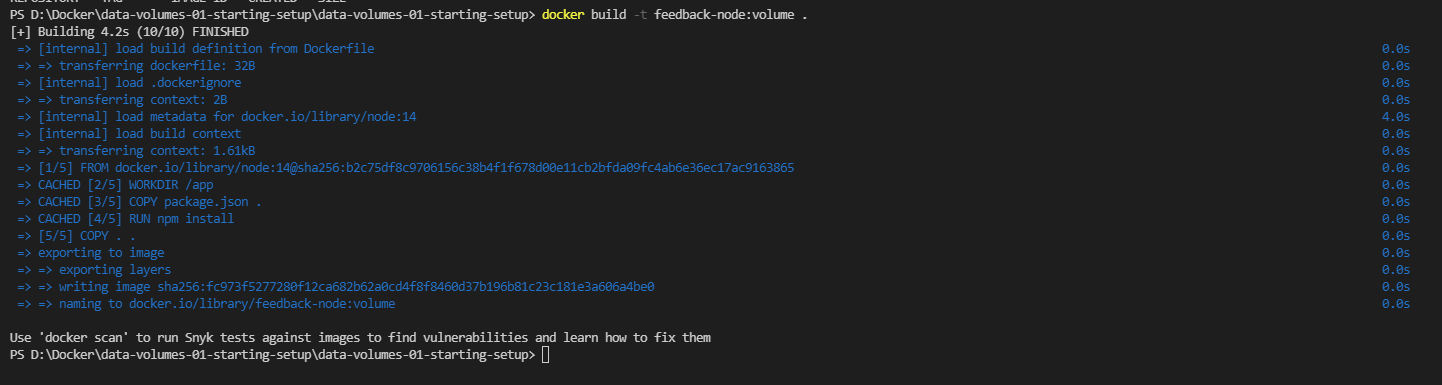


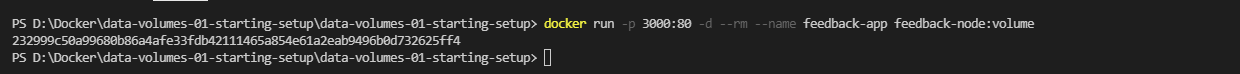
In order to remedy this we need to replace the rename method with copyFile method.

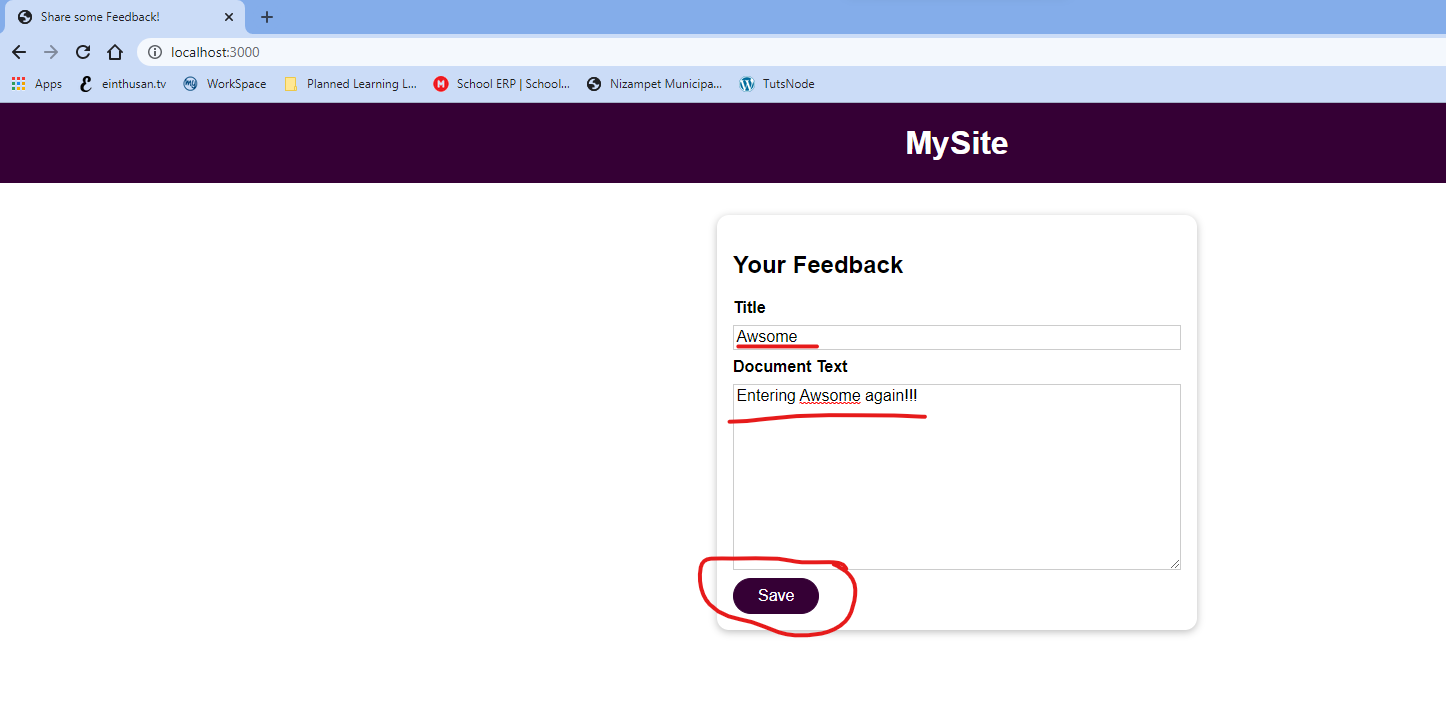




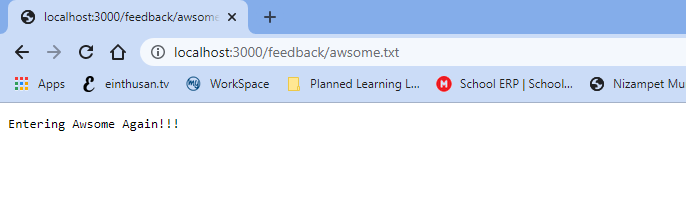




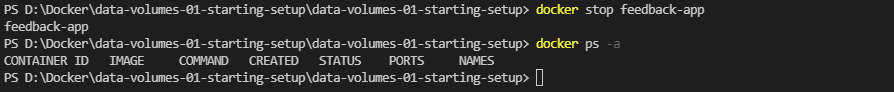




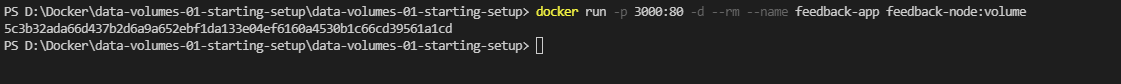
Now this time it will save the code. You can also verify the feedback by browsing the following url.

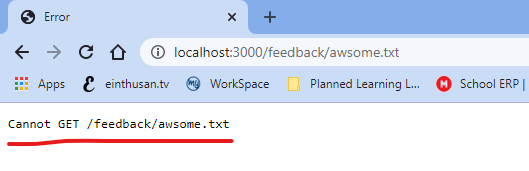


As you had used the option --rm previously it will remove the container and hopefully we will see the feedback still.



Now if I run the brand new container and verify it once we will not be able to find the feedback.





With docker we have multiple external data storage mechanism.

1. Volumes
2. Bind Mounts

Bind Mounts we will deal with later, now Volumes are of two types

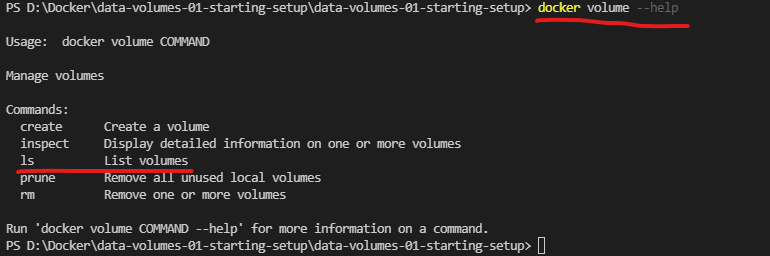
🡪 Volumes

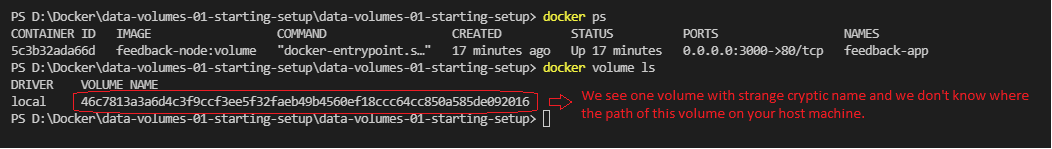
* Anonymous Volumes (Which will created by mentioning instruction on Dockerfile)
* Named Volumes (which will create by dynamically by using docker run command)

Each of them will have their own purpose, up to now we had create Anonymous Volumes by mentioning the instruction “VOLUME” on Dockerfile.

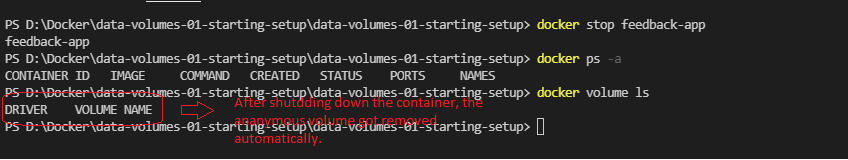
In either case Anonymous or Named volumes, docker will map the folder inside to the container with some path on your host which we don’t know where it is on your host system.

We can only see these volumes by using docker command “docker volume <option>”.



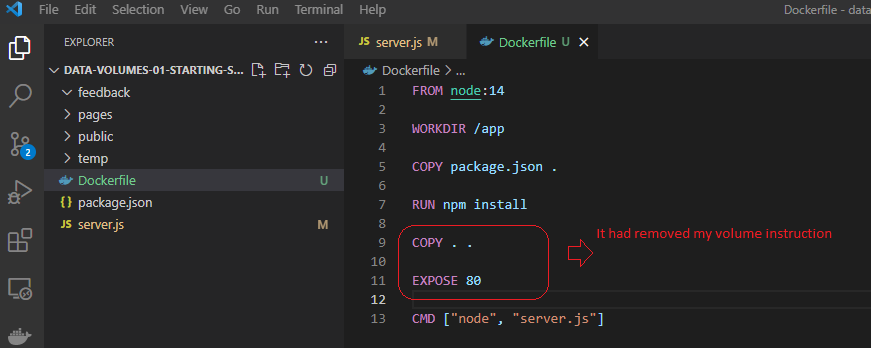


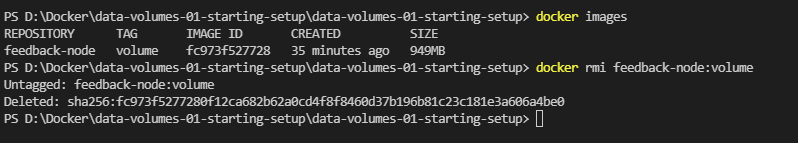
Now if we shutdown the container, as we had used the option --rm it will remove the container and see if the anonymous volume exists or not.

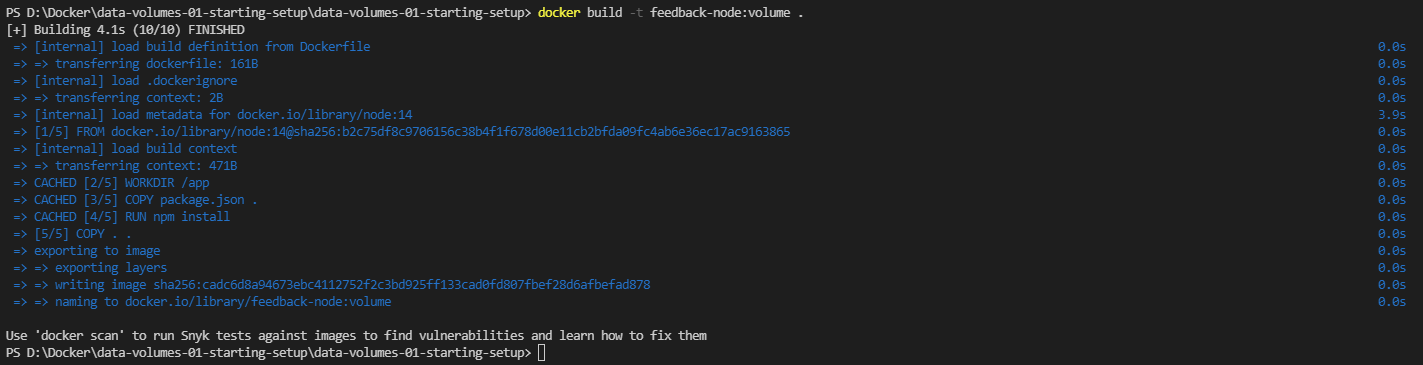


Now let’s see if we create a Named volume and if we shutdown and recreate the container whether the volume will exists?

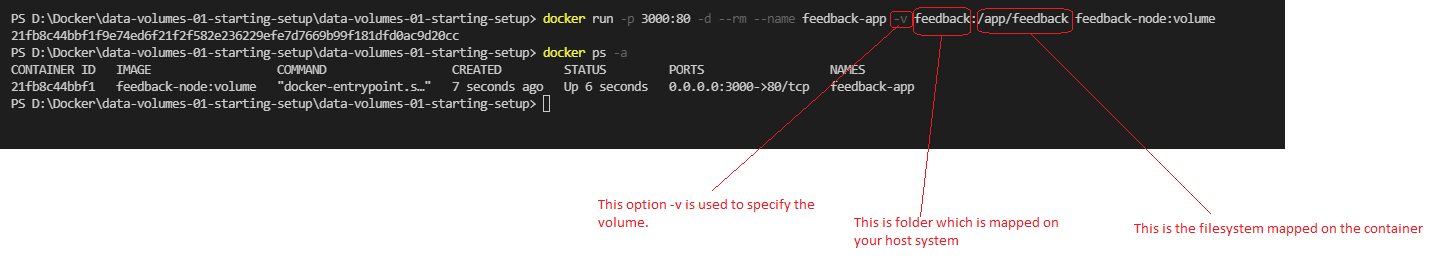
As we know to create a named volume, we need to mention at docker run command by using the option -v

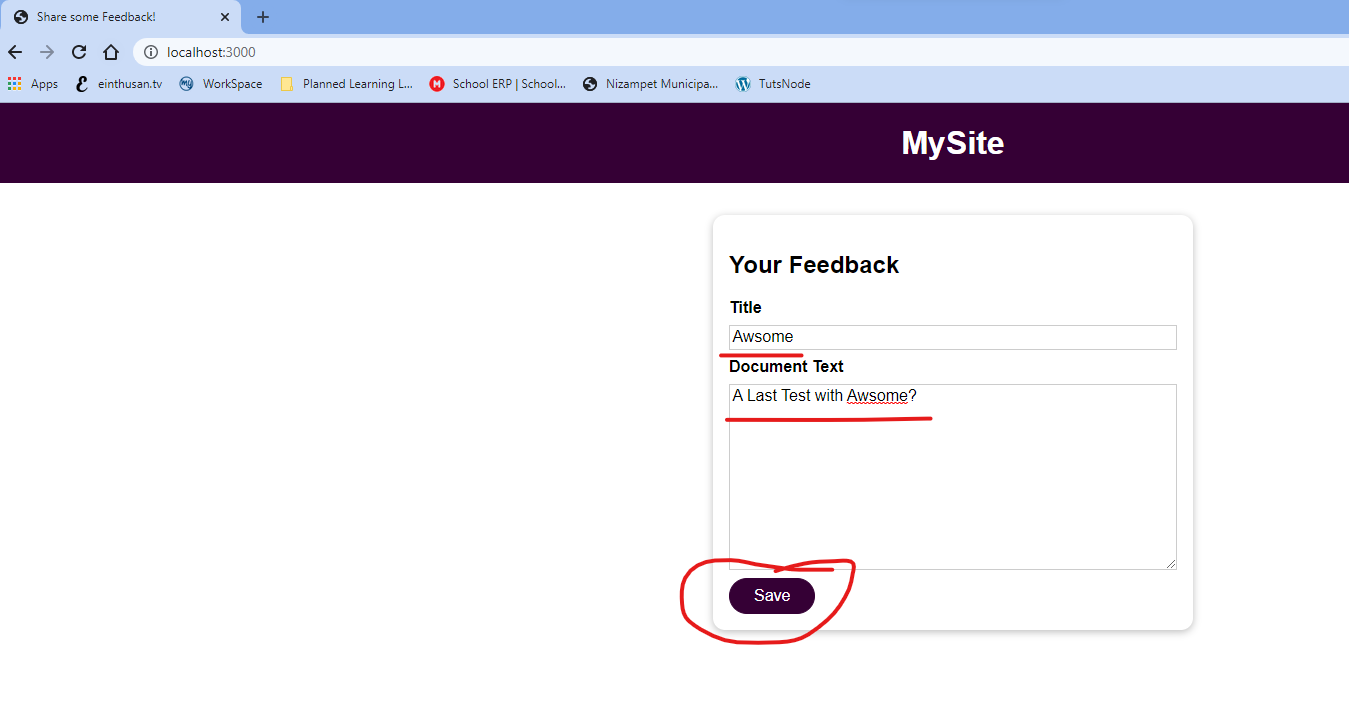


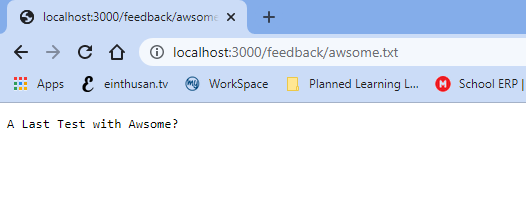


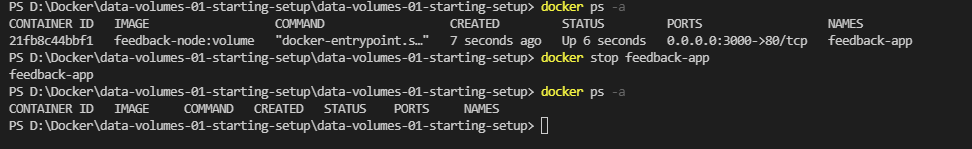


Note: Anonymous volumes are closely attached to one specific container, named volumes are not attached to container.

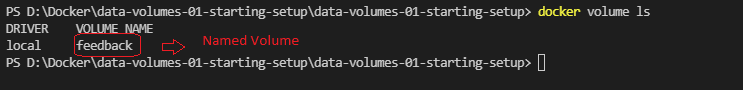






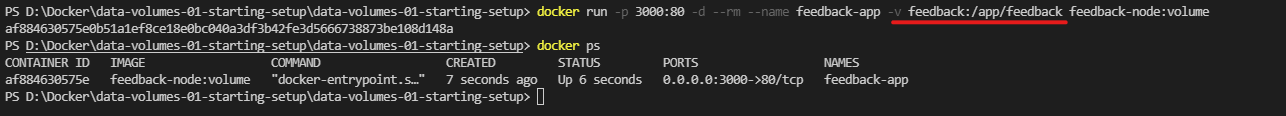


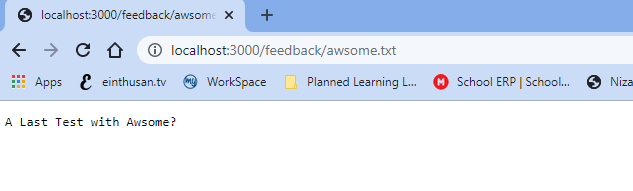
Now check whether the named volume exists or not.



As you can see the volume still exists, we had option --rm previously, it will remove the container.

Now we if start the new container by attaching the same named volume, you will find the feedback, which was saved on that volume.





Finally we managed to persist data with help of named volumes.