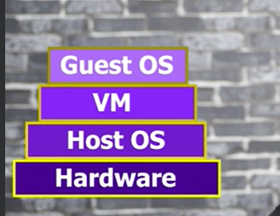
Docker important points:

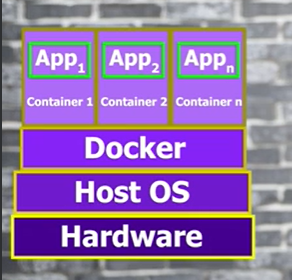
Virtualization :



VM – Virtual Machine

VM we can implement using Hypervisor.

Container:



Docker is a tool to achieve Containerization.

Docker components:

1.Docker Engine – creating and managing containers.

2.Containers.

3.Image – light weight file of your container.

4.Docker file – for creating images

5.Docker hub – library type where useful images are available

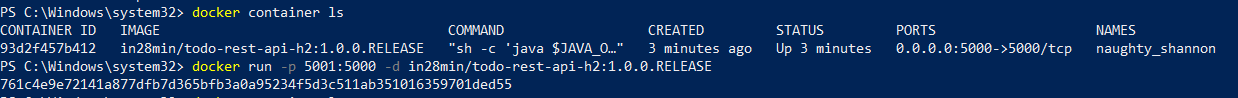
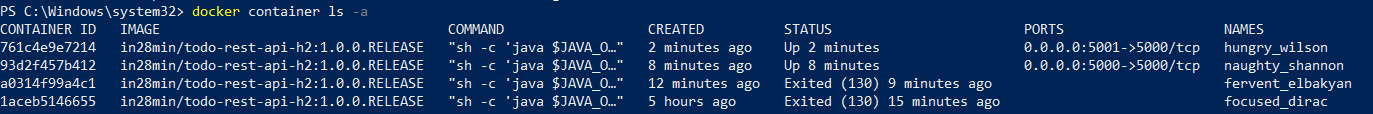
6.Storage and volumes

7.Docker compose – for application requiring **multiple containers.**

**May 20, 2024**

* Docker is a container tool.
* **Hypervisor** : software that creates and run virtual machines or manages virtual machines.
* Run docker with admin access first and then run power shell will admin access.
* How to run a docker container?

**Docker container run in28min/todo-rest-api-h2:1.0.0.RELEASE  
 (or)**  
**docker run in28min/todo-rest-api-h2:1.0.0.RELEASE**

* What is container? **Running version of image is called – container**  
  **Image is a static version and container is a dynamic version.**
* How to run docker container in particular port say 5000? **docker run -p 5000:5000 in28min/todo-rest-api-h2:1.0.0.RELEASE**  
  By default if you run any container it is part of bridge network in docker.NO body will be able to access it.  
  -p 5000:5000 => -p {HostPort}:{ContainerPort}  
  where p is **publish**  
  **Container Port :** The port number within the container that's listening for connections.  
  **Host Port** : The port number on your host machine where you want to receive traffic.  
  AS of now, in my understanding the host port and container port should be same. And those will be same as that of application port we mentioned in the project.
* If we click ctrl + c then application will be terminated. But I don’t want it so I can do this  
  **docker run -p 5000:5000 -d in28min/todo-rest-api-h2:1.0.0.RELEASE**  
  where d = **detach**  
  you will get container Id  
  if you want to see logs then give **docker logs c9b5(part of id – first 3 to 4 characters to uniquely identify the container )**  
  keep following the new logs -> **docker logs -f id**where f is **follow**  
  **ctrl + c** to clear tail logs  
  **clear** -> to clear the shell  
  **docker container ls** -> to display the containers running  
    
  **docker images** -> displays the images that are local to us.  
    
  **docker container ls -a** -> shows the running containers and stopped containers  
  
* How to stop running docker container ?  
  **docker container stop id**  
  

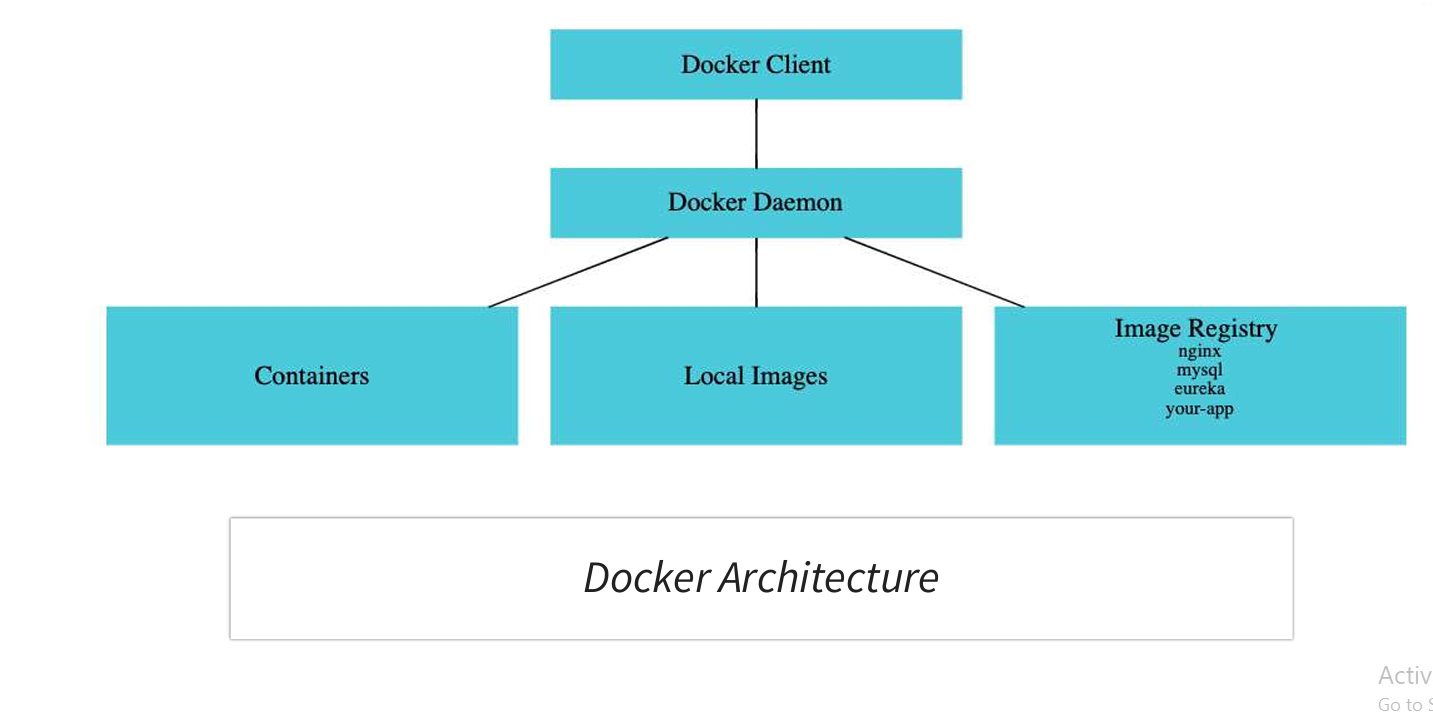
**May 21,2024 (commands on images)**

Docker is a open platform for developing, shipping and running applications.

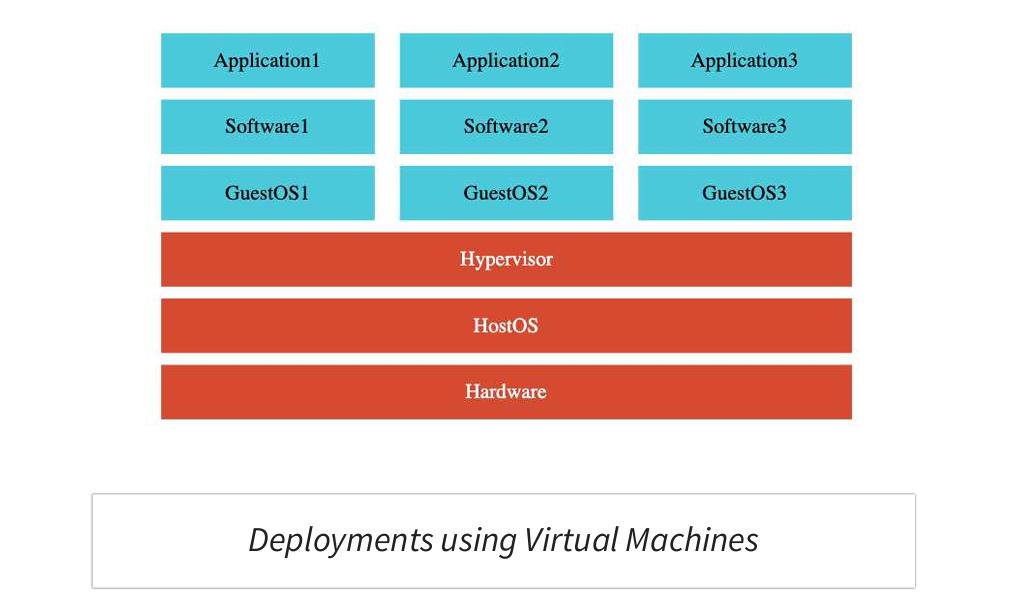
Place we are running commands is called **docker client.**

After starting the docker only you can execute the commands in the power shell.

**Docker Architecture:**

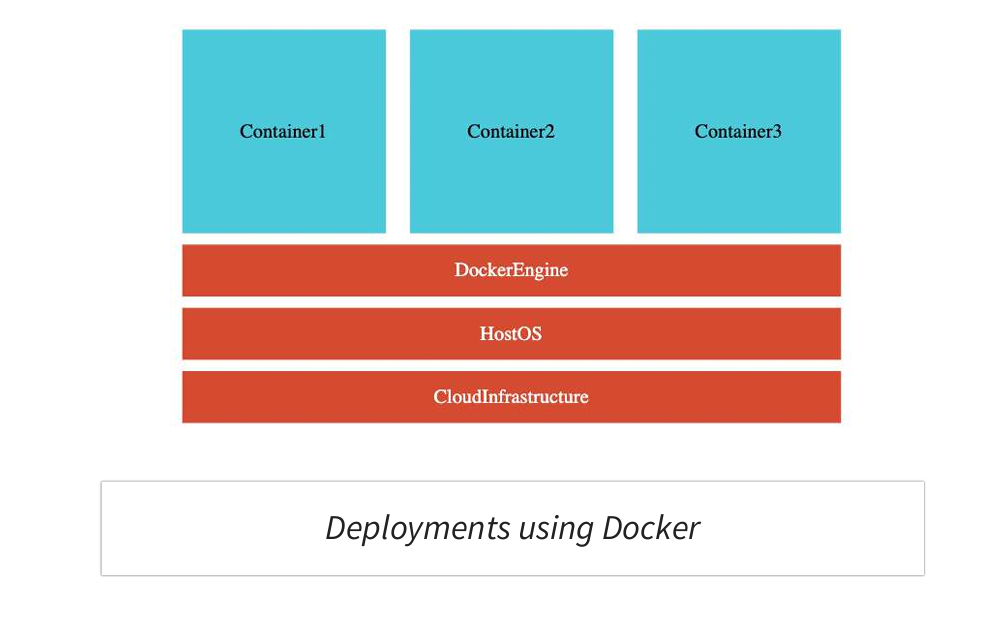


* **Docker demon** is responsible for maintaining the containers, local images and Image registry. You can use Image Registry your push your local images to docker.
* Deployments before docker

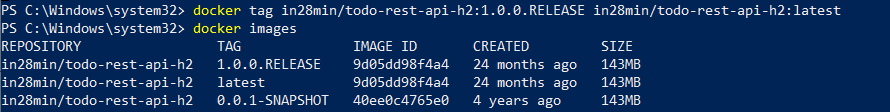
  
The **disadvantage** of this is we need to maintain the guest OS and host OS which makes it **heavy weight**.

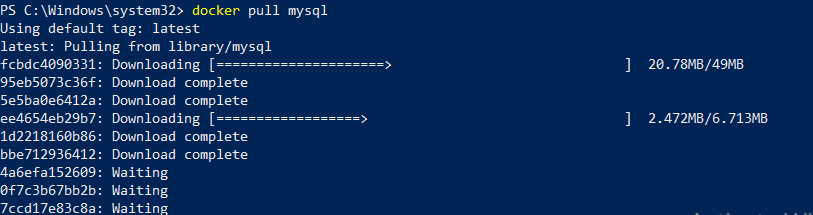
Container is **light weight.**

Existing image if you try to run, then it won’t download again. If the image is new only it will get downloaded into the local registry.



To create one more image version in local with different tag and same name use:

**docker tag in28min/todo-rest-api-h2:1.0.0.RELEASE in28min/todo-rest-api-h2:latest**  
here we have created newer images with latest version from 1.0.0.RELEASE.  
  
Image id will be same and tag will be different.  
latest doesn’t always point to latest version.

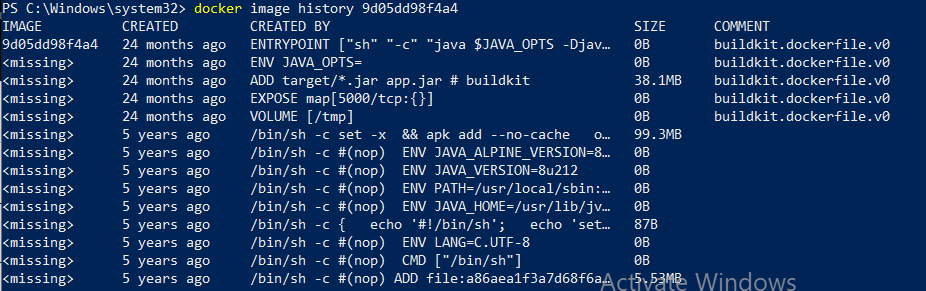


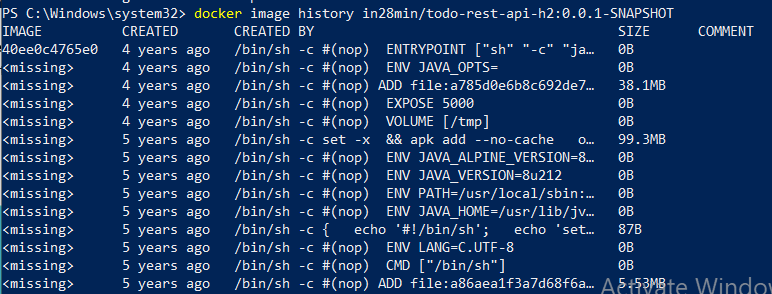
Whenever you try to **pull image it will always pull the latest** unless you mention the version. Latest doesn’t mean it is pulling the latest release version, it can be any version in the past.

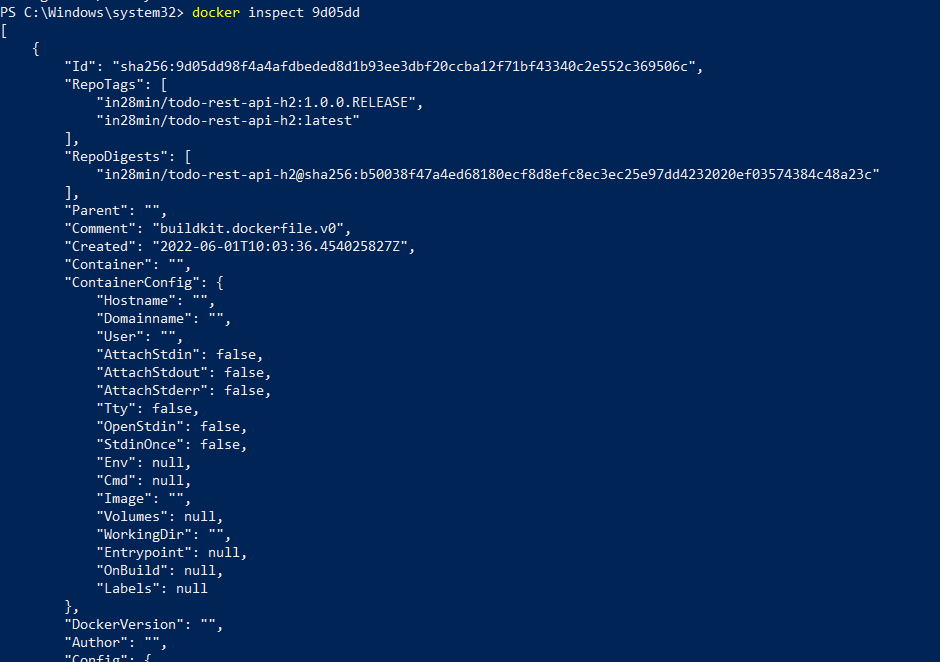
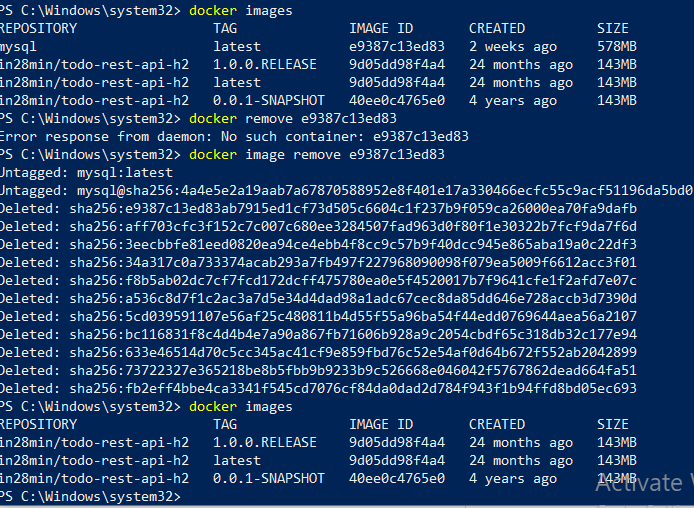
Docker pull will download the image from docker to your local registry, it does not run.

Use **docker run** to run, if it is not present in local then it will try to download it from docker and then run it.  
**mysql is a official image.**

**What is official image ?**

If you want to search any image then u can use docker search. You don’t need to mention entire name instead part of it will be sufficient to that it will pull all the matching names related.  
  
  
**The Docker Official Images are a curated set of Docker repositories hosted on Docker Hub.** These will follow particular standards and docker team will monitor it to make it official.  
**For image history use :**  
for image history you can also use

**docker image history repository:release version**eg : docker image history in28min/todo-rest-api-h2:0.0.1-SNAPSHOT  
docker image history command **will work on local images only.**  


  
 Inspect will works only on local images.  
**docker image remove imageID**  


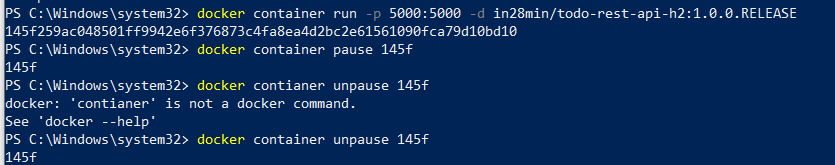
It will delete the image, but it will not delete it from local registry.

**May 22,2024 (commands on containers)**

For running a container we used shortcut  
**docker run -p 5000:5000 in28min/todo-rest-api-h2:1.0.0.RELEASE**but the actual command for running the container is **docker container run -p 5000:5000 in28min/todo-rest-api-h2:1.0.0.RELEASE**

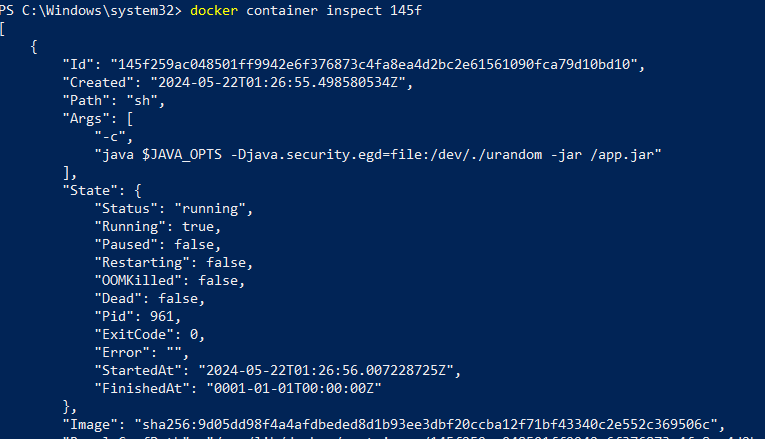
**How can you pause and unpause the container ?**

**docker container pause contianerId**pause is similar to stopping **docker container unpause containerId**Unpause is similar to resume.

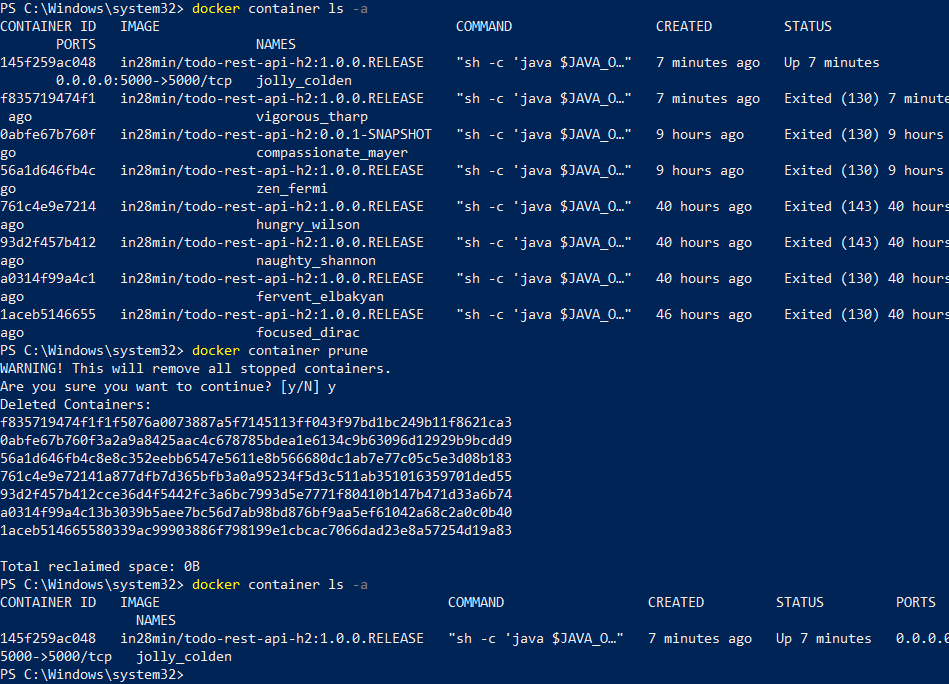


In pause mode you won’t get any response, until you unpause it.

**For inspecting a container you can use :**

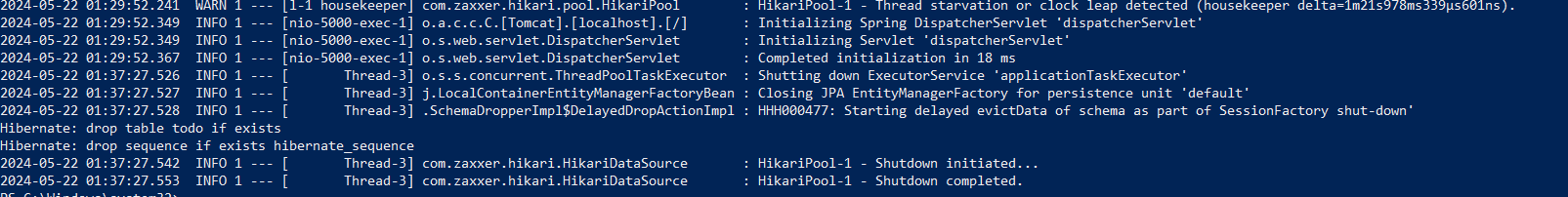


If you want to remove the images history that are stopped then you can use  
 **docker container prune**



Stop => SIGTERM => graceful shutdown.

**What is graceful shutdown ?**

Correctly shutting down the application instead of stopping it then and there. **docker container stop imageId**

**Killing the container   
  
docker container kill imageId**

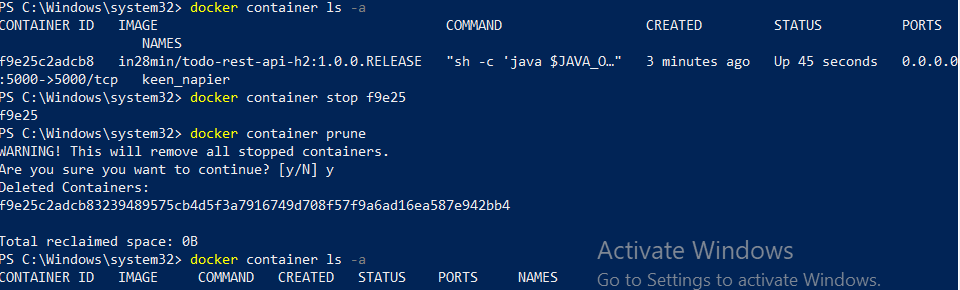
The application will be stopped then and there without giving any time for shutdown the resources.  
kill => SIFKILL => immediately terminates the process.

The docker kill command is used to stop one or more running containers immediately.

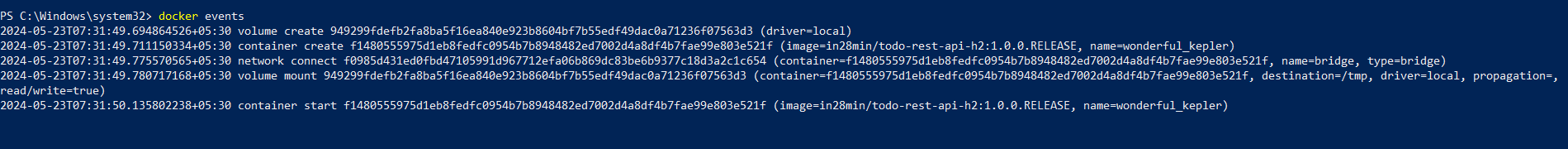
**How to run a docker container even after restart of docker ?**

Use : **docker container run -p 5000:5000 -d --restart=always in28min/todo-rest-api-h2:1.0.0.RELEASE**

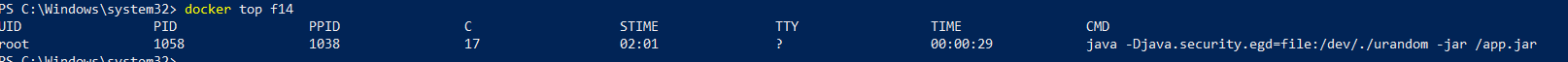
Popular values for restart is --restart=always and --restart=no, by default no will be present.  
Even if you stop the container, when you restart your docker automatically this container will be running.  
restart policy is useful when you want to use a container always running like database.  
  
**How to prevent it from happening restart ?**  
by stopping the running container and giving docker container prune.



**May 23, 2024**

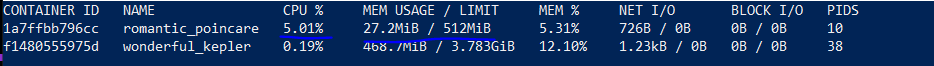
**docker events**   
helps us to see what all events are going to happen with the image.  
To check this you need to have two tabs , one containing docker events and other to run the containers.

**docker top continerID**  
helps us to see what is the top process running in a specific container.



**docker stats**

show the information about running containers.  
  
  
**docker container run -p 5001:5000 -m 512m --cpu-quota 5000 -d in28min/todo-rest-api-h2:1.0.0.RELEASE**

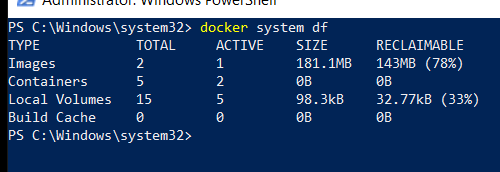
-m 512m = allocate the memory of 512 mega bytes, you can also use 1GB and 0.5GB as well.  
  
--cpu quota = allocating the cpu usage for container  
  
100000 = 100% then 50000 = 50% then 5% = 5000  


Since the above one is using 5% CPU it is very slow in starting up.

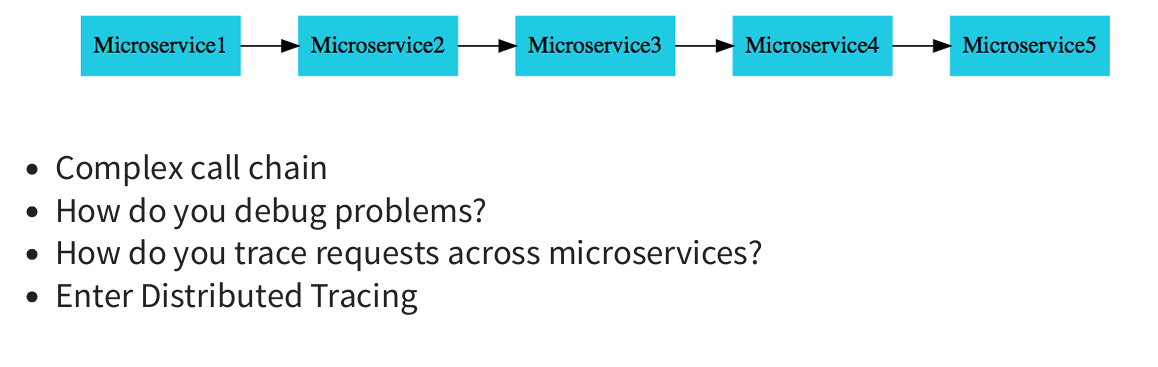
If I increase to 50000 then its working like a pro.

**docker container run -p 5001:5000 -m 512m --cpu-quota 5000 -d in28min/todo-rest-api-h2:1.0.0.RELEASE**

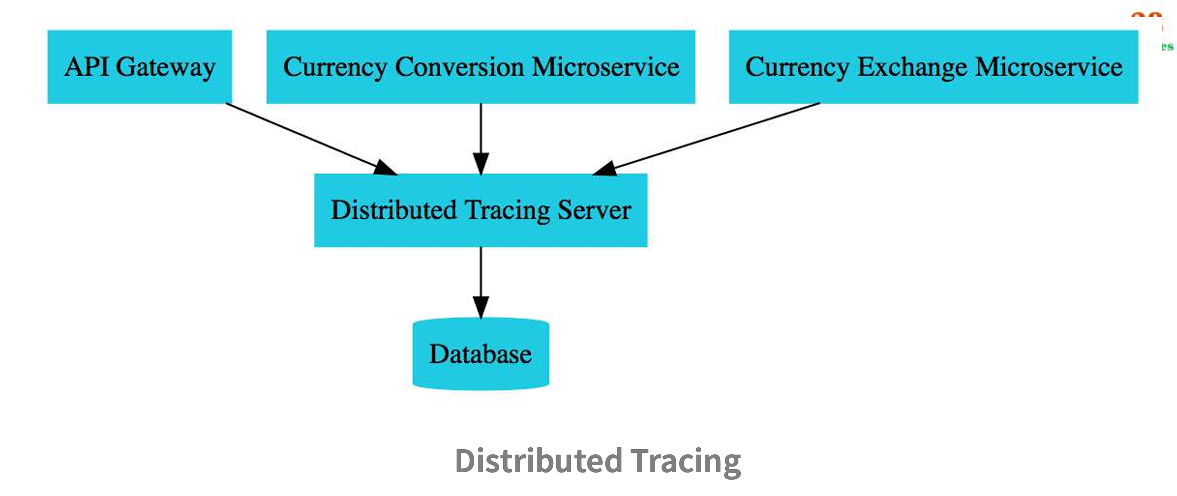
**docker system df**command displays information regarding the amount of disk space used by the Docker daemon.  
its helps us to look us different things that docker daemon manages.



24th May ,2024



To trace a request across multiple microservices we use **distributed tracing**

  
  
Here the distributed tracing server which we are using is **Zipkin**.  
To pull zipkin from docker used   
**docker run -p 9411:9411 openzipkin/zipkin:2.23**  
TO see it in local used : localhost:9411/zipkin/  
  
**What is observability and open Telemetry ?**

Before understanding observability you need to understand monitoring :

**Monitoring**: Gather the data : metrics , logs and traces

Monitoring is reactive. Observability is proactive.

Monitoring is a subset of observability.  
  
**Observability**: How well do we understand what’s happening in a system ?

Step 1 : gather data : logs, metrics or traces

Step 2: Get intelligence : AI/Ops and anomaly detection

**Earlier we used to have different standards for each of logs, metrics and traces.**

**But now we are having a single standard of all the three, i.e open Telemetry.**  
  
**Open Telemetry** : Collection of tools, APIs, and SDKs to instrument, generate, collect & export telemetry data(metrics, logs & traces)  
>All applications will have metrics, logs and traces  
 Why do we need to have a separate standard for each one of these ?  
>Open Telemetry : How about one standard for metrics, logs and traces ?  
>Almost all cloud platform provides support for open telemetry today.

**Integration with microservices** :

Dependency’s you need to add for registering microservices with zipkin.

For tracing every request use :

management.tracing.sampling.probability=1.0  
  
 for tracing only 5% then use 0.05 instead of 1.0

with using **resttemplate for calling other microservice will not be able to trace in zipkin**.  
To integrate RestTemplate with micrometer, create RestTemplateBuilder

@Configuration(proxyBeanMethods = false)

class RestTemplateConfiguration {

@Bean

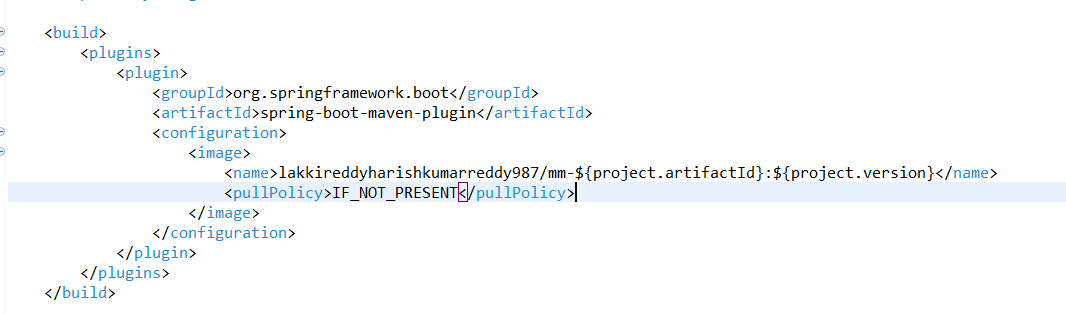
RestTemplate restTemplate(RestTemplateBuilder builder) {

return builder.build();

}

}

May 25,2024  
How to build docker images from existing local code ?

You can use spring-boot-maven-plugin for creating images.

Docker Id is lakkireddy….  
name is mm-artifactId : projectversion  
To build a docker image spring uses a lot of docker plugins.  
To configure it I used the pullPolicy. By default is always. Instead of default I made it in more meaningful way i.e if images are not present locally then go and pull them.  
run as maven build   
give command -> spring-boot:build-image -DskipTests  
do not use -DskipTests for real projects.  
give Run.

It will take some time to download and move the image to container.

Then we can be able to launch and hit the urls.

Launching each and every microservice in the above process is not easy, so to simplify this we use docker compose.

TO start all in single go use YAML file.

Compose simplifies the control of your entire application stack, making it easy to manage services, networks, and volumes in a single, comprehensible YAML configuration file. Then, with a single command, you create and start all the services from your configuration file.

docker -compose –version

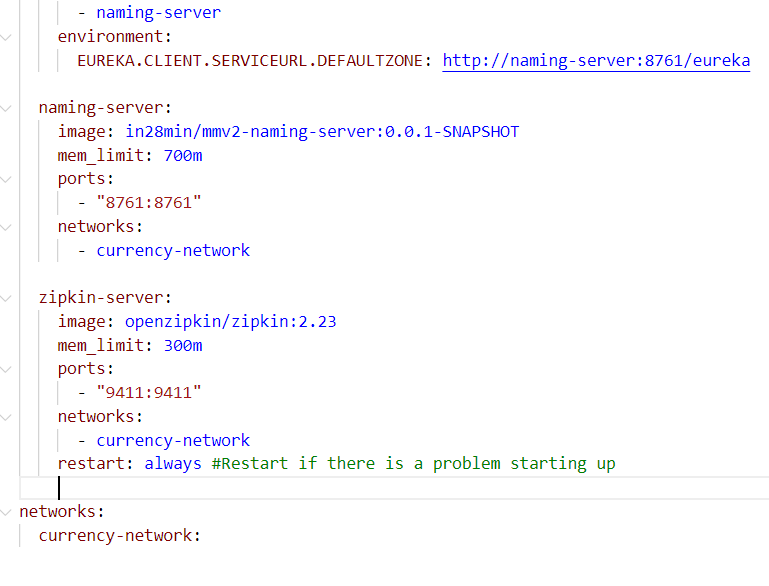
do not use tabs in side yaml file, always use spaces.  
 have the file name as docer-compose and give command after entering into that directory,  
running use command -> docker compose up  
if the file name is different then give   
docker compose -f test.yml up  
  
from docker localhost does not means similar way as you are running from your machine.  
ways to change :

eureka.client.serviceUrl.defualtZone = <http://naming-server:8761/eureka>  
  
naming-server is the name which we have given in yml file.

2nd way is by environment variable in yml file  
    environment:

      EUREKA.CLIENT.SERVICEURL.DEFAULTZONE: http://naming-server:8761/eureka

Example of a docker compose file below :

We can extract code from local image to local file.  
From one repo in docker we can download image and add it to another repo.  
private company’s use to follow private repository’s.  
**4️⃣ Key Differences in a Table**

| **Feature** | **Path Variable (@PathVariable)** | **Query Parameter (@RequestParam)** | **Matrix Variable (@MatrixVariable)** |
| --- | --- | --- | --- |
| **Placement** | In URL path (/users/{id}) | After ? (/users?name=John) | Inside path (/users;name=John;age=25) |
| **Use Case** | Identify a **specific resource** | **Filtering, sorting, pagination** | **Multiple filters inside a path** |
| **Mandatory?** | ✅ Yes | ❌ No (optional) | ❌ No (optional) |
| **Format** | /users/123 | /users/search?name=John | /users/filter;name=John;age=25 |