Docker Basic

- Images, Container and Registry

Basic Commands

- pull
- run
- start
- stop
- exec
- inspect
- logs
- ps
- stats
- images
- detached mode
- environment variables

The concepts of Docker images and containers are fundamental to understanding how Docker works.

Docker Images:
- An image is a **lightweight**, **standalone**, **executable package** that includes everything needed to run a piece of software, including the code, runtime, libraries, environment variables, and configuration files.
- Images are created using a **Dockerfile**, which specifies the instructions needed to build the image.
- Images are **immutable**, meaning they cannot be changed or modified. Instead, when changes are made, a new image is created.
- Docker images are stored in a **Docker registry**, such as Docker Hub, where they can be easily shared and accessed by others.

Docker Containers:
- A container is a **runnable instance** of a Docker image. It encapsulates the application and its dependencies, allowing it to **run in isolation from other processes**.
- Containers are **lightweight**, **portable**, and can be easily deployed across different environments.
- Containers are created from Docker images using the '**docker run**' command.
- Containers can be started, stopped, paused, restarted, and deleted as needed.

Key Differences:
- **Purpose**: Images are used to package and distribute applications, while containers are used to run instances of those applications.
- **State**: Images are immutable and stateless, while containers have a state and can be running, stopped, or paused.
- **Lifecycle**: Images exist independently of containers and can be used to create multiple containers. Containers are ephemeral and can be created, started, stopped, and deleted.
- **Storage**: Images are stored in a Docker registry, while containers run on a Docker host.

In summary, images are the **blueprints** for containers, defining what the container will look like and contain, while containers are the **runtime instances** of those images, executing the application within a controlled environment.

Registry:
A Docker registry is a **repository for Docker images**. It serves as a centralized location where Docker images can be stored, shared, and retrieved. Docker registries play a crucial role in the Docker ecosystem by facilitating collaboration, distribution, and deployment of containerized applications.

Here are some key points about Docker registries:

1. **Public Registries:** Docker Hub is the most popular public Docker registry. It hosts a vast collection of public Docker images that cover a wide range of applications and software stacks.

Anyone can push their Docker images to Docker Hub, making it a convenient platform for sharing and discovering Docker images.

2. **Private Registries:** Organizations often use private Docker registries to store proprietary or sensitive Docker images. Private registries provide more control over access permissions and image distribution. Docker Trusted Registry (DTR) and Amazon Elastic Container Registry (ECR) are examples of private Docker registry solutions.

3. **Image Distribution:** Docker registries allow users to push (upload) Docker images to the registry and pull (download) images from the registry. This enables seamless distribution of Docker images across different environments, such as development, testing, and production.

4. **Authentication and Authorization:** Docker registries support authentication and authorization mechanisms to control access to Docker images. Users may need to authenticate with the registry using credentials (e.g., username and password, API tokens) before they can push or pull images.

5. **Registry Mirroring:** Docker registries support mirroring, which allows organizations to replicate images from one registry to another. This is useful for distributing images across geographically distributed locations or ensuring high availability and redundancy.

6. **Registry Management:** Docker registries provide management interfaces and APIs for administrators to manage images, users, access controls, and other registry configurations. This allows organizations to customize registry settings according to their requirements.

Overall, Docker registries play a vital role in the Docker workflow by serving as the central hub for storing, sharing, and distributing Docker images, whether for public use or within private organizational environments.

Docker basics command with Postgres image:

The **PostgreSQL Docker image** is a popular choice for running PostgreSQL as a containerized application. Here's how you might use some of the Docker commands with the PostgreSQL image:

1. Pull: To download the PostgreSQL Docker image from Docker Hub, you would use:

docker pull postgres

2. Run: To create a container from the PostgreSQL image and start it, you would use:

docker run --name my_postgres_container -e POSTGRES_PASSWORD=test12345 -d postgres

This command creates a new container named '**my_postgres_container'** from the '**postgres'** image, setting the environment variable '**POSTGRES_PASSWORD**' to '**test12345** and running it in detached mode ('**-d**').

3. Start and Stop: To start or stop the PostgreSQL container, you would respectively use:

docker start my_postgres_container

docker stop my_postgres_container

4. Exec: To execute commands inside the running PostgreSQL container, you would use:

docker exec -it my_postgres_container psql -U postgres

docker exec -it my_postgres_container bash

This command starts an interactive '**psql'** session as the '**postgres'** user within the '**my_postgres_container**'.

5. Inspect: To inspect details about the PostgreSQL container, you would use:

docker inspect my_postgres_container

6. Logs: To view the logs of the PostgreSQL container, you would use:

docker logs my_postgres_container

7. PS: To list running containers, including the PostgreSQL container, you would use:

docker ps

8. Stats: To view live resource usage statistics of the PostgreSQL container, you would use:

docker stats my_postgres_container

9. Images: To list all locally stored Docker images, including the PostgreSQL image, you would use:

docker images