More on Docker

Docker is a containerisation platform that allows you to package and distribute applications and their dependencies consistently and efficiently. It was developed by Docker, Inc. (now part of Mirantis), and it has become a popular technology in software development and deployment.

Here are the key components and concepts of Docker in detail:

- **Container**: A container is a lightweight, standalone, and executable package that includes everything needed to run a piece of software, including the code, runtime, system tools, libraries, and settings. Containers are isolated from one another and from the host system, which makes them portable and consistent across different environments.

- **Docker Engine**: This is the core component of Docker. It's a client-server application with the following parts:

- **Docker Daemon**: The background service manages Docker containers on a host system.

- **Docker CLI**: The command-line interface used to interact with the Docker Daemon.

- **Docker Image**: A Docker image is a read-only template for creating containers. It contains the application code, libraries, and other dependencies needed to run the application. Images are typically built from a set of instructions called a Dockerfile.

- **Docker Container**: A running instance of a Docker image. Containers are isolated and share the host system's kernel but have their own file system, processes, and network stack.

- **Dockerfile**: A text file that contains a set of instructions for building a Docker image. It specifies a base image and a series of commands to install software, copy files, set environment variables, and configure the container.

- **Registry**: Docker images are typically stored in registries, which are repositories for sharing and distributing Docker images. Docker Hub is the default public registry, but you can also set up private registries for your organisation.

- **Docker Compose**: A tool for defining and running multi-container Docker applications. It allows you to define a multi-container application using a YAML file and start them all with a single command. This is particularly useful for complex applications with multiple services.

- **Volumes**: Docker provides a way to persist data using volumes. Volumes are directories on the host system or in another container, which can be mounted into a container, allowing data to be shared or stored outside the container's filesystem.

- **Networking**: Docker provides various networking options to connect containers and the external world. You can create custom networks and assign containers to them, allowing them to communicate securely.

- **Security**: Docker uses several security features like namespaces and control groups to isolate containers from each other and the host system. However, it's essential to properly configure and manage security aspects like user privileges, network settings, and container isolation to ensure a secure environment.

- **Versioning**: Docker images can be versioned to track changes and updates over time. This helps in maintaining consistency and reproducibility in your application deployments.

Docker simplifies the process of packaging, distributing, and running applications, making it easier to develop and deploy software in a consistent and reproducible manner. It has found wide adoption in DevOps, continuous integration, and continuous delivery pipelines, as well as in cloud and container orchestration platforms.

Docker-compose Configuration

Docker Compose allows you to define and run multi-container applications using a simple YAML file. You can specify the services, networks, volumes, and other configurations for your application within this file. Here's a step-by-step guide on how to write a Docker Compose configuration file with a sample example.

Let's Create a Spring Boot application that uses a MySQL database with Docker, which involves several steps. Here's a simplified guide to set up such an application:

Docker-compose file:

```
version: '3'
services:
  docker-mysql:
    image: mysql:5.7
    container_name: docker-mysql
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: mydb
    ports:
      - "3306:3306"
```

We can use the MySQL database container defined in the Docker Compose configuration as a database for a Spring Boot application. Here's how we can integrate your Spring Boot application with the MySQL container:

1. **Spring Boot Configuration**: In the Spring Boot application, we must configure the data source to connect to the MySQL database. We can do this in your application.properties or application.yml file.

```
spring:
  datasource:
    url: jdbc:mysql://docker-mysql:3306/mydb
    username: root
    password: root
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
```

2. **Start Docker Containers:** Run the following command in your project directory to start the Docker containers defined in your docker-compose.yml file:

```
docker-compose up
```

3. **Run Your Spring Boot Application**: Run your Spring Boot application. It will connect to the MySQL container running in Docker.

4. **Access Your Application**: You can access your Spring Boot application via the defined endpoints and interact with the MySQL database.

This setup allows you to run a Spring Boot application that uses a MySQL database within Docker containers. Ensure you have Docker installed and running on your system for this to work. You can expand upon this setup by adding more components and configurations.

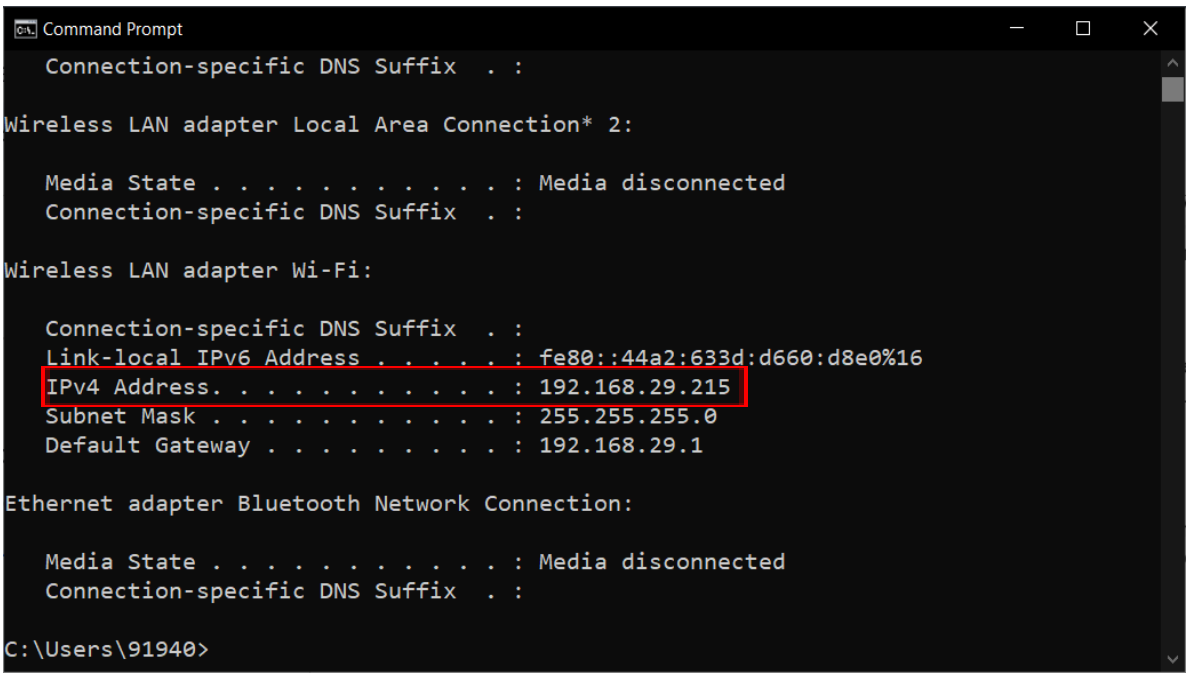
This approach is beneficial for development and testing, as it provides an isolated, reproducible environment with both the Spring Boot application and the MySQL database running in containers. For production, you would typically use more advanced configurations with considerations for data persistence, security, and scaling.

Find your IP address using the command prompt (CMD)

You can quickly get your local IP address on your computer with the Windows Command Prompt tool. It'll show you both your IPv4 address and your IPv6 address. Here's how to find your IP address using CMD:

1. Open the Start menu and type cmd to open the Command Prompt.

2. Type "***ipconfig***" into the Command Prompt and press Enter. The tool will return a set of data that includes your IP address.



How to load a docker file in the local system

To load a Docker image from a file (commonly a tarball file with the .tar extension) into your local Docker system, you can use the Docker load command. Here are the steps to do this:

1. **Transfer the Docker Image File**: If you have the Docker image file on your local system, skip this step. If the image file is on another system, you must transfer it to your local machine. You can use tools like **scp**, **rsync**, or a simple file transfer through a shared network or cloud storage. You can also use the docker desktop application and search and load images from there.

2. **Copy the Docker Image File to a Local Directory**: Copy the Docker image file to a directory on your local machine. Make sure you remember the path to the file.

3. **Load the Docker Image**: Open your terminal or command prompt and navigate to the directory where the Docker image file is located.

4. **Run the "*docker load*" Command**: Use the docker load command to load the Docker image. Replace [*path-to-image-file*] with the actual path to your image file:

```
docker load -i [path-to-image-file]
```

For example, if your image file is named my_image.tar and is located in the current directory, you can use:

```
docker load -i my_image.tar
```

This command will read the image file, unpack it, and load it into your local Docker system.

5. **Verify the Loaded Image**: To verify that the image was loaded successfully, you can list all Docker images on your system using the docker images command:

```
docker images
```

You should see your loaded image in the list.

That's it! The Docker image has been loaded into your local Docker environment. You can now use it to run containers on your local machine.

Please note that the docker load command is used for loading saved Docker images, not for loading container instances or running containers. If you have a Docker Compose file or other configurations, you must use docker-compose up or similar commands to start containers based on the loaded image.

How to delete an Image in the docker

To delete a Docker image, you can use the docker ***rmi*** (remove image) command. Here's how to do it:

```
docker rmi [OPTIONS] IMAGE [IMAGE...]
```

Usage:

1. **List Docker Images**: First, list the Docker images on your system to identify the image you want to delete. You can use the docker images command to record all available images. Note the *REPOSITORY* and *TAG* of the image you want to delete.

```
docker images
```

2. **Delete the Image**: Once you've identified the image you want to remove, use the docker rmi command to delete it. Replace *[IMAGE]* with the name and optional tag of the image you want to remove. For **example**, to delete an image named my_image:

```
docker rmi my_image
```

If the image has a specific tag, you can include it:

```
docker rmi my_image:tag
```

You can also delete multiple images in a single command by providing their names or tags separated by spaces:

```
docker rmi my_image1 image2 image3
```

3. **Force Image Deletion (if necessary):** If the image is in use by running containers, Docker will not allow you to delete it by default. In such cases, you can use the -f or --force option to force the deletion:

```
docker rmi -f my_image
```

Be cautious when using the -f option, as it will forcefully remove the image along with any containers that are currently using it.

4. **Verify Image Deletion:** After successfully running the docker rmi command, the specified image will be deleted from your local Docker environment. You can use docker images again to verify that the image is no longer listed.

```
docker images
```

That's it! You've successfully deleted a Docker image. Please be careful when using the -f option, as it can lead to the loss of associated containers and data.

Common Errors While Using Docker

Common errors and issues when using Docker can occur due to a variety of reasons. Here are some of the most common errors and their potential solutions:

1. **Image Pull Failures**:
    a. **Error**: "docker: image operating system 'Linux' cannot be used on this platform."
    b. **Solution**: This error typically occurs when running a Linux-based Docker image on a Windows or macOS host without enabling Docker Desktop's Linux containers. You need to switch to Linux containers in Docker Desktop.

2. **Container Networking Issues**:
    a. **Error**: Containers can't communicate with each other, or network ports are inaccessible.
    b. **Solution**: Ensure that containers are connected to the same Docker network. Check for port conflicts and make sure containers are using the correct ports.

3. **Containers Not Starting**:
    a. **Error**: Containers fail to start, or they exit immediately.
    b. **Solution**: Review the container logs using docker logs <container_id> for error messages. Make sure the application inside the container is working correctly.

4. **Volume Mounting Problems**:
    a. **Error**: Data is not persisting when using volumes.
    b. **Solution**: Double-check the volume paths file permissions and ensure the mounted directories exist. Also, consider SELinux or AppArmor if using a Linux host.

5. **Permission Denied**:
    a. **Error**: "Permission denied" when accessing files within a container.
    b. **Solution**: Ensure that file permissions in your host and container match. Use the -u flag to set the user when running containers or consider using a volume with proper permissions.

6. **Out of Memory or Resources:**
    a. **Error**: Containers failing with "out of memory" or resource limitations.
    b. **Solution**: Adjust resource limits for containers using the --memory and --cpus options when running them. Also, ensure there are enough resources on the host.

7. **Docker Daemon Unresponsive**:
    a. **Error**: Docker commands don't respond, or the Docker daemon is unresponsive.
    b. **Solution**: Restart Docker or your system. Check the system's resource usage and logs for any issues.

8. **Image or Container Already Exists**:
    a. **Error**: "Conflict. The container name is already in use."
    b. **Solution**: Choose a unique container name, or use docker rm <container_id> to remove the existing container with the same name.

Refer to these resources for more information

1. Docker official documentation
2. Spring boot docker
3. Container vs Image
4. Doker Desktop for Windows