As a DevOps engineer, you should be familiar with a wide range of Docker commands to manage containers, images, volumes, and networks efficiently. Here are some of the most important Docker commands you should know:

1. Basic Docker Commands

These are essential commands for basic Docker operations like working with containers and images.

- **`docker --version`**
  Displays the installed Docker version.

  ```
  docker --version
  ```

- **`docker info`**
  Displays system-wide information about Docker, including the number of containers, images, and other statistics.

  ```
  docker info
  ```

- **`docker run`**
  Runs a container from a Docker image.

  ```
  docker run [options] <image_name>
  ```

  Common options:

  - `-d`: Run container in detached mode (background).
  - `-p`: Map ports (e.g., `-p 8080:80`).
  - `--name`: Assign a custom name to the container.

- **`docker ps`**
  Lists all running containers.

  ```
  docker ps
  ```

  To list all containers (including stopped ones):

  ```
  docker ps -a
  ```

- **`docker exec`**
  Execute a command inside a running container.

  ```
  docker exec -it <container_id> <command>
  ```

  Example: Access the shell of a running container.

  ```
  docker exec -it <container_id> /bin/bash
  ```

- **`docker stop`**
  Stops a running container.

  ```
  docker stop <container_id>
  ```

- **docker start**
  Starts a stopped container.

  ```
  docker start <container_id>
  ```

- **docker restart**
  Restarts a running container.

  ```
  docker restart <container_id>
  ```

- **docker rm**
  Removes a stopped container.

  ```
  docker rm <container_id>
  ```

- **docker logs**
  Fetches logs from a running or stopped container.

  ```
  docker logs <container_id>
  ```

2. Working with Docker Images

Images are the blueprints for containers. You can build, list, and remove images using these commands.

- **docker pull**
  Pulls an image from a Docker registry (like Docker Hub).

  ```
  docker pull <image_name>:<tag>
  ```

- **docker build**
  Builds an image from a Dockerfile.

  ```
  docker build -t <image_name>:<tag> <path_to_dockerfile_directory>
  ```

- **docker images**
  Lists all Docker images on your local machine.

  ```
  docker images
  ```

- **docker rmi**
  Removes one or more Docker images.

  ```
  docker rmi <image_id_or_name>
  ```

- **docker tag**
  Tags an image with a new name.

  ```
  docker tag <image_id> <new_image_name>:<tag>
  ```

- **docker push**
  Pushes an image to a Docker registry.

  ```
  docker push <image_name>:<tag>
  ```

3. Managing Docker Volumes

Volumes are used to persist data generated by and used by Docker containers.

- **`docker volume create`**
  Creates a Docker volume.

  ```
  docker volume create <volume_name>
  ```

- **`docker volume ls`**
  Lists all Docker volumes.

  ```
  docker volume ls
  ```

- **`docker volume inspect`**
  Displays detailed information about a volume.

  ```
  docker volume inspect <volume_name>
  ```

- **`docker volume rm`**
  Removes a Docker volume.

  ```
  docker volume rm <volume_name>
  ```

4. Managing Docker Networks

Docker networks allow containers to communicate with each other.

- **`docker network create`**
  Creates a new Docker network.

  ```
  docker network create <network_name>
  ```

- **`docker network ls`**
  Lists all Docker networks.

  ```
  docker network ls
  ```

- **`docker network inspect`**
  Displays detailed information about a network.

  ```
  docker network inspect <network_name>
  ```

- **`docker network rm`**
  Removes a Docker network.

  ```
  docker network rm <network_name>
  ```

5. Docker Compose Commands

Docker Compose is a tool for defining and running multi-container Docker applications.

- **`docker-compose up`**
  Starts services defined in a `docker-compose.yml` file.

```
docker-compose up
```

To run in detached mode:

```
docker-compose up -d
```

- **`docker-compose down`**
  Stops and removes containers, networks, and volumes defined in a `docker-compose.yml` file.

  ```
  docker-compose down
  ```

- **`docker-compose ps`**
  Lists the status of containers defined in a Compose file.

  ```
  docker-compose ps
  ```

- **`docker-compose logs`**
  Fetches logs for the services defined in a Compose file.

  ```
  docker-compose logs
  ```

- **`docker-compose build`**
  Builds images for services defined in the `docker-compose.yml` file.

  ```
  docker-compose build
  ```

6. Docker Container Management and Inspection

You'll often need to inspect or manage containers while troubleshooting or configuring your applications.

- **`docker inspect`**
  Provides detailed information about a container or image.

  ```
  docker inspect <container_id_or_image_name>
  ```

- **`docker top`**
  Displays running processes inside a container.

  ```
  docker top <container_id>
  ```

- **`docker stats`**
  Displays real-time resource usage statistics (CPU, memory, etc.) for running containers.

  ```
  docker stats
  ```

- **`docker diff`**
  Shows changes made to the container's filesystem.

  ```
  docker diff <container_id>
  ```

7. Docker System Commands

These commands help you manage the overall Docker environment.

- **`docker system prune`**
  Removes unused data (dangling images, stopped containers, unused volumes, etc.).

  ```
  docker system prune
  ```

  To prune everything (including volumes):

  ```
  docker system prune -a --volumes
  ```

- **`docker login`**
  Logs into a Docker registry (e.g., Docker Hub).

  ```
  docker login
  ```

- **`docker logout`**
  Logs out from a Docker registry.

  ```
  docker logout
  ```

8. Docker Build and Debugging Commands

For debugging and optimizing Docker images and containers.

- **`docker build --no-cache`**
  Builds a Docker image without using any cache from previous builds.

  ```
  docker build --no-cache -t <image_name> .
  ```

- **`docker build -f`**
  Specifies a different Dockerfile to build the image from.

  ```
  docker build -f <Dockerfile_path> -t <image_name> .
  ```

- **`docker logs -f`**
  Follows the logs of a running container in real-time.

  ```
  docker logs -f <container_id>
  ```

These are just the fundamental Docker commands that every DevOps engineer should be comfortable with. As you advance, you might also need to dive deeper into Dockerfile optimization, container security, and managing container orchestration with tools like Kubernetes. But knowing these basic commands will form the foundation of your Docker expertise! Let me know if you need help with anything else.