Docker Concepts and Commands Documentation

Table of Contents

1. Introduction to Docker
2. Key Concepts
3. Docker Architecture
4. Docker Commands
5. Dockerfile
6. Docker Compose
7. Best Practices

Introduction to Docker

Docker is an open-source platform that automates the deployment, scaling, and management of applications using containerization. It allows you to package an application with all of its dependencies into a standardized unit for software development and deployment.

Key Concepts

1. **Container**: A lightweight, standalone, executable package that includes everything needed to run a piece of software, including the code, runtime, system tools, libraries, and settings.
2. **Image**: A read-only template used to create containers. Images are created with the `docker build` command and can be stored in a Docker registry like Docker Hub.
3. **Dockerfile**: A text file that contains instructions for building a Docker image.
4. **Docker Hub**: A cloud-based registry service for storing and sharing Docker images.
5. **Volume**: A mechanism for persisting data generated by and used by Docker containers.
6. **Network**: A communication system that allows containers to communicate with each other and with the outside world.

Docker Architecture

Docker uses a client-server architecture:

1. **Docker Client**: The primary way users interact with Docker through the command line.
2. **Docker Host**: The machine running the Docker daemon.
3. **Docker Daemon**: The background service running on the host that manages building, running, and distributing Docker containers.
4. **Docker Registry**: Stores Docker images. Docker Hub is a public registry that anyone can use.

Docker Commands

Here are some essential Docker commands:

Image Management

- `docker pull <image>`: Download an image from a registry
- `docker push <image>`: Upload an image to a registry
- `docker build -t <name:tag> .`: Build an image from a Dockerfile
- `docker images`: List all local images
- `docker rmi <image>`: Remove an image

Container Management

- `docker run <image>`: Create and start a new container
- `docker ps`: List running containers
- `docker ps -a`: List all containers (including stopped)
- `docker stop <container>`: Stop a running container
- `docker start <container>`: Start a stopped container
- `docker restart <container>`: Restart a container
- `docker rm <container>`: Remove a container
- `docker exec -it <container> <command>`: Run a command in a running container

System & Info

- `docker info`: Display system-wide information
- `docker version`: Show the Docker version information
- `docker logs <container>`: Fetch the logs of a container

Network Management

- `docker network create <network>`: Create a network
- `docker network ls`: List networks
- `docker network rm <network>`: Remove a network

Volume Management

- `docker volume create <volume>`: Create a volume
- `docker volume ls`: List volumes
- `docker volume rm <volume>`: Remove a volume

Dockerfile

A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. Here's a basic structure:

```
FROM <base_image>
WORKDIR /app
COPY . .
RUN <command>
EXPOSE <port>
CMD ["executable","param1","param2"]
```

Common instructions:

- `FROM`: Sets the base image
- `WORKDIR`: Sets the working directory
- `COPY` or `ADD`: Copies files from host to the container
- `RUN`: Executes commands in a new layer
- `EXPOSE`: Informs Docker that the container listens on specified network ports
- `CMD`: Provides defaults for an executing container

Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. It uses YAML files to configure application services.

Basic structure of a `docker-compose.yml` file:

```
services:
  web:
    build: .
    ports:
      - "8080:8080"
  redis:
    image: "redis:alpine"
```

Common commands:

- `docker-compose up`: Create and start containers
- `docker-compose down`: Stop and remove containers, networks, images, and volumes
- `docker-compose ps`: List containers
- `docker-compose logs`: View output from containers

Best Practices

1. Use official images as base images
2. Minimize the number of layers in your Dockerfile
3. Don't install unnecessary packages
4. Use multi-stage builds for smaller final images
5. Use `.dockerignore` file
6. Make use of Docker volumes for persistent data
7. Use environment variables for configuration
8. Run containers with least privileges
9. Regularly update your Docker images
10. Use Docker Compose for multi-container applications

Remember, Docker is a powerful tool that can significantly simplify your development and deployment processes when used correctly.