DOCKER

Deep Dive: Essential Commands, Tips & Tricks

Introduction to Docker

Docker is a containerization platform that allows developers to package applications and their dependencies into lightweight, portable containers. It simplifies deployment, scaling, and management of applications across different environments.

Essential Docker Commands

1. Managing Containers

- `docker run <image>` – Run a container from an image.
- `docker ps` – List running containers.
- `docker ps -a` – List all containers (including stopped ones).
- `docker stop <container_id>` – Stop a running container.
- `docker rm <container_id>` – Remove a stopped container.
- `docker restart <container_id>` – Restart a container.
- `docker exec -it <container_id> bash` – Open an interactive shell inside a running container.
- `docker logs <container_id>` – View logs of a container.

2. Managing Images

- `docker images` – List available images.
- `docker pull <image>` – Download an image from Docker Hub.
- `docker build -t <image_name> .` – Build an image from a Dockerfile.
- `docker rmi <image_name>` – Remove an image.

3. Networking in Docker

- `docker network ls` – List Docker networks.
- `docker network create <network_name>` – Create a custom network.
- `docker network inspect <network_name>` – Inspect network details.
- `docker network connect <network> <container>` – Attach a container to a network.

4. Volume Management

- `docker volume ls` – List volumes.
- `docker volume create <volume_name>` – Create a volume.

- `docker volume inspect <volume_name>` – Inspect a volume.
- `docker run -v <volume_name>:/data <image>` – Attach a volume to a container.

5. Docker Compose

- `docker-compose up` – Start up a multi-container application.
- `docker-compose down` – Stop and remove all containers defined in a `docker-compose.yml` file.
- `docker-compose logs` – View logs from services managed by Docker Compose.

Deep Dive: Docker Concepts

1. Dockerfile Best Practices

- Use `FROM` with a minimal base image (e.g., `alpine` or `debian-slim`).
- Minimize layers by combining commands (`RUN apt-get update && apt-get install -y curl`).
- Use `.dockerignore` to exclude unnecessary files from the build context.
- Set an explicit working directory with `WORKDIR`.
- Use `COPY` instead of `ADD` when possible for better transparency.
- Define non-root users for enhanced security.

2. Optimizing Image Size

- Multi-stage builds: Use multiple `FROM` statements to reduce final image size.
- Remove unnecessary dependencies after installation (`RUN apt-get purge -y && rm -rf /var/lib/apt/lists/*`).
- Use smaller base images like `alpine`.

3. Handling Logs & Debugging

- Use `docker logs -f <container_id>` to stream logs.
- Attach a shell to a running container using `docker exec -it <container_id> sh`.
- Check container resource usage: `docker stats <container_id>`.
- Inspect container details: `docker inspect <container_id>`.

4. Securing Docker

- Enable Docker Content Trust (DCT) with `export DOCKER_CONTENT_TRUST=1`.
- Limit container privileges using `--cap-drop`.
- Use read-only filesystems (`--read-only` flag).
- Restrict CPU and memory usage (`--memory` and `--cpu-shares`).

- Regularly update images to patch vulnerabilities.

5. Scaling with Docker

- Use Docker Compose for local multi-container applications.
- Leverage Kubernetes for orchestrating large-scale containerized applications.
- Implement rolling updates using `docker service update` in a Swarm setup.

Pro Tips & Tricks

✅ **Reduce Image Size**: Use `--squash` when building images to merge layers and reduce size.

✅ **Cache Dependencies**: Install dependencies before copying app files to leverage Docker's build cache.

✅ **Auto-remove Containers**: Use `docker run --rm <image>` to remove containers automatically after execution.

✅ **Persist Data Efficiently**: Use named volumes instead of bind mounts for better data persistence.

✅ **Use Health Checks**: Add `HEALTHCHECK` to Dockerfiles to ensure application readiness.

✅ **Avoid Running as Root**: Always create a non-root user inside the container (`USER appuser`).

✅ **Speed Up Builds**: Use `docker build --no-cache` only when necessary to avoid rebuilding unchanged layers.

✅ **Security Audits**: Regularly scan images with `docker scan <image>` to identify vulnerabilities.

Mastering these Docker commands and best practices will help you build efficient, secure, and scalable containerized applications. 🚀