Some Docker best practices that will improve your app image quality and size

1. Use Official Base Images: When building Docker images, it's recommended to use official base images from trusted sources like Docker Hub. Official images are typically well-maintained and regularly updated. For example, using the official Nginx base image:

FROM nginx:latest

2. Minimize Image Layers: Aim to minimize the number of layers in your Docker image. Each layer adds overhead and increases the image size. Use multi-stage builds to reduce the final image size. Here's an example:

FROM node:14 AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
FROM nginx:latest
COPY --from=build /app/dist /usr/share/nginx/html

3. Use .dockerignore File: Create a **.dockerignore** file to exclude unnecessary files and directories from being copied into the image. This helps reduce the build context and improves build performance.

4. Specify Versioned Tags: When pulling base images or other dependencies, specify versioned tags instead of using the **latest** tag. This ensures consistency and avoids unexpected changes. For example:

FROM node:14

5. Use COPY Instead of ADD: Prefer using the **COPY** instruction over **ADD** unless you specifically require the additional functionality provided by **ADD**. **COPY** is simpler and has fewer side effects. For example:

COPY src /app/src

6. Remove Unnecessary Packages and Files: Clean up unnecessary packages and files in the same layer to reduce the image size. Use the **RUN** instruction to remove packages or files that are no longer needed. For example:

**RUN apt-get purge -y --auto-remove build-essential && **

rm -rf /var/lib/apt/lists/*

7. Use Environment Variables for Configuration: Use environment variables to pass configuration to your application. This makes it easier to configure the application in different environments. Here's an example:

ENV PORT=8080 ENV
DATABASE_URL=mysql://user:password@hostname/database

8. Use Container Orchestration: Consider using container orchestration tools like Kubernetes or Docker Swarm for managing and scaling your containers in production. These tools provide features for service discovery, load balancing, scaling, and rolling updates.

These are just a few best practices to follow when working with Docker. It's important to continuously update and improve your Dockerfiles and container configurations based on your specific use case and requirements.