Docker Registry, DockerHub, Create a Multi-Stage Build

Docker Registry:

A Docker Registry is a storage and content delivery system for Docker images. It acts as a centralized repository where users can store, manage, and retrieve container images.

Public vs. Private: Public registries (like DockerHub) allow open access, while private registries (e.g., AWS ECR, Harbor) require authentication.

Image Tagging: Images are tagged with a registry address, repository name, and version (e.g., docker.io/library/ubuntu:latest).

Push/Pull: Developers push images to registries and pull them to deploy environments.

DockerHub :

DockerHub is Docker's official cloud-based registry service where we can store and share container images.

DockerHub provides:

- Public repositories (free)
- Private repositories (limited free, paid for more)
- Official images maintained by Docker
- Community images from users worldwide
- Automated builds from GitHub/Bitbucket

Multi-Stage Build :

Multi-stage builds optimize Docker images by splitting the build process into stages, reducing final image size.

Multi-stage builds in Docker allow you to create optimized Docker images by separating the build environment from the runtime environment, resulting in smaller, more secure, and easier-to-maintain images. This approach involves using multiple stages within a single Dockerfile, where each stage represents a separate build environment.

We are creating a simple Multi stage Dockerfile for Node.js Application as follows

```
# Stage 1: Build stage
FROM node:18-alpine AS builder

# Set working directory
WORKDIR /app

# Copy package files
COPY package*.json ./
```

```
# Install all dependencies (including dev dependencies)
RUN npm ci

# Copy source code
COPY . .

# Build the application
RUN npm run build


# Stage 2: Production stage
FROM node:18-alpine

# Install dumb-init for proper signal handling
RUN apk add --no-cache dumb-init

# Create non-root user
RUN addgroup -g 1001 -S nodejs
RUN adduser -S nodejs -u 1001

# Set working directory
WORKDIR /app

# Copy package files
COPY package*.json ./

# Install only production dependencies
RUN npm ci --only=production && npm cache clean --force

# Copy built application from builder stage
COPY --from=builder /app/dist ./dist

# Change ownership to nodejs user
RUN chown -R nodejs:nodejs /app

# Switch to non-root user
USER nodejs

# Expose port
EXPOSE 3000

# Start application with dumb-init
ENTRYPOINT ["dumb-init", "--"]

CMD ["node", "dist/index.js"]
```

Now we’ll create the image and push it to out DockerHub registry

```
# Build the image
docker build -t username/appname:tag .

# Login to DockerHub
docker login

# Push the image
docker push username/appname:tag

# Pull from anywhere
docker pull username/appname:tag
```

In such a way, we have created a Docker Multi-stage image where there are two stages as Build and Production stage.
This allows us to optimize image size and improve efficiency.