**🚀 Docker Learning Roadmap**

**🚦 Stage 1: Beginner – Docker Basics**

**📘 Concepts to Learn:**

1. **Containerization vs. Virtualization**
   * Benefits of containers over VMs
   * How Docker fits in the ecosystem
2. **Installation**
   * Docker Engine (Linux)
   * Docker Desktop (Windows/macOS)
3. **Docker Images & Containers**
   * docker run, docker ps, docker stop, docker rm
   * docker exec, docker logs, docker inspect
   * Detached (-d) vs Interactive (-it) mode
4. **Dockerfile & Image Creation**
   * Basic Dockerfile instructions: FROM, COPY, RUN, CMD
   * Build and tag images: docker build, docker tag
5. **Volumes (Data Persistence)**
   * Bind mounts vs Named volumes
   * Commands: docker volume create, -v, --mount
6. **Docker Networking**
   * Bridge, host, none
   * Commands: docker network create, --link, --network
   * Publishing and exposing ports (-p vs -P)
7. **Environment Variables**
   * Using ENV, ARG in Dockerfile
   * .env files with Docker Compose
   * Basics of secret handling
8. **.dockerignore File**
   * Optimize builds by ignoring unnecessary files
9. **Docker Compose (Multi-Container Setup)**
   * docker-compose.yml structure
   * Services, volumes, networks, depends\_on
10. **Logging and Monitoring**

* docker logs, log rotation basics
* Simple performance metrics via docker stats

1. **Docker Registry**

* Push/pull from Docker Hub or private registry

**⚙️ Stage 2: Intermediate – Practical Usage & Optimization**

**🧩 Concepts to Learn:**

1. **Container Lifecycle & Health Checks**
   * HEALTHCHECK in Dockerfile
   * Restart policies: --restart=always, unless-stopped, on-failure
2. **Multi-Stage Builds**
   * Separate build and runtime environments
   * Reduce image sizes
3. **Image Optimization**
   * Use Alpine or minimal base images
   * Minimize layers, combine commands
   * Clean up temp files (rm -rf /var/cache/...)
4. **Security Basics**
   * Use non-root users
   * Limit capabilities, use seccomp/apparmor (intro)
   * Avoid storing secrets in images
5. **Debugging Containers**
   * Use docker exec, docker logs, docker inspect, docker stats
   * Understand container exit codes
6. **Custom Networks & Communication**
   * Creating isolated bridge networks
   * Container DNS resolution via service names
7. **Testing Docker Images**
   * Use hadolint to lint Dockerfiles
   * Smoke testing locally before pushing
   * Validate config with docker-compose config

**🧠 Stage 3: Expert – Production Readiness & Advanced Features**

**🛡 Production-Ready Practices:**

1. **Security Best Practices**
   * **Use docker scan, trivy, or grype for image scanning**
   * **Don't run as root**
   * **Avoid exposing unnecessary ports**
   * **Restrict container capabilities**
2. **Advanced Docker Compose**
   * **Profiles, conditionals**
   * **Compose override files**
   * **External volume/network usage**
3. **Advanced Networking**
   * **Macvlan networks (if needed)**
   * **Overlay networks in Docker Swarm (intro)**
4. **Performance & Monitoring**
   * **Use metrics tools: cAdvisor, Prometheus (optional)**
   * **Use log drivers (syslog, json-file, etc.)**
5. **Image Lifecycle and Caching**
   * **Cache busting techniques**
   * **Layer reusability and ordering in Dockerfile**
6. **CI/CD Integration**
   * **Build and push images in pipelines (GitHub Actions, GitLab CI, Azure DevOps)**
   * **Use labels for image versioning and traceability**
7. **Testing Strategies**
   * **Unit test in containers**
   * **Integration testing with Docker Compose**
   * **Contract testing between services**
8. **Container Patterns**
   * **Init containers, sidecars (useful for K8s later)**
   * **One-process-per-container principle**