**Docker Optimization: Distroless Images & Multi-Stage Builds**

**1. Software Delivery Pipeline Overview**

**CI/CD Flow**

1. **Code Development** →
2. **SAST Testing** (Checkmarx/SonarQube) →
3. **Build Artifacts** (Maven/Gradle) →
4. **Dockerize** →
5. **Push to Registry** (ECR/Docker Hub)

**Key Insight:**

"Like cooking biryani, we don't serve raw ingredients (code) directly - we process it through defined steps."

**2. Distroless Images Deep Dive**

**Characteristics**

| **Feature** | **Traditional Image** | **Distroless Image** |
| --- | --- | --- |
| **Size** | 100MB+ | 2-5MB |
| **Shell Access** | Yes (/bin/bash) | No |
| **SSH** | Available | Not Available |
| **Security** | Medium | High (CVE-free) |

**Practical Demo**

dockerfile

Copy

Download

FROM gcr.io/distroless/static-debian11

COPY app /app

ENTRYPOINT ["/app"]

**Gotcha:**

bash

Copy

Download

docker exec -it container sh *# Fails - no shell!*

**When to Use**

* Production deployments needing maximum security
* Serverless functions (AWS Lambda)
* Final stage of multi-stage builds

**3. Multi-Stage Builds Masterclass**

**Standard vs Multi-Stage Comparison**

| **Metric** | **Single-Stage (682MB)** | **Multi-Stage (284MB)** |
| --- | --- | --- |
| **Build Time** | Faster | Slower |
| **Final Size** | Larger | 58% smaller |
| **Security** | All tools included | Only runtime deps |

**Spring Boot Example**

dockerfile

Copy

Download

# Stage 1: Builder

FROM maven:3.8.6-jdk-11 AS builder

WORKDIR /app

COPY . .

RUN mvn package -DskipTests

# Stage 2: Runtime

FROM eclipse-temurin:17-jre-jammy

WORKDIR /app

COPY --from=builder /app/target/\*.jar app.jar

EXPOSE 8080

CMD ["java", "-jar", "app.jar"]

**Key Optimization:**

* Builder stage includes JDK (~400MB)
* Final stage uses JRE-only image (~150MB)

**4. Dockerfile Best Practices**

**Layer Optimization**

**Bad:**

dockerfile

Copy

Download

RUN apt update

RUN apt install -y curl

RUN curl -o file.tar.gz http://example.com

RUN tar -xzf file.tar.gz

**Good:**

dockerfile

Copy

Download

RUN apt update && \

apt install -y curl && \

curl -o file.tar.gz http://example.com && \

tar -xzf file.tar.gz && \

apt remove -y curl && \

rm file.tar.gz

**Nginx Official Example**

dockerfile

Copy

Download

RUN set -x \

&& addgroup --system --gid 101 nginx \

&& adduser --system --disabled-login --ingroup nginx --no-create-home --home /nonexistent --gecos "nginx user" --shell /bin/false --uid 101 nginx \

&& apt-get update \

&& apt-get install --no-install-recommends --no-install-suggests -y gnupg1 ca-certificates \

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

**5. Hands-On Lab Results**

**Image Size Reduction**

1. **Initial Build:** 682MB
2. **Multi-Stage v1:** 469MB (-31%)
3. **Optimized v2:** 284MB (-58% from original)

**Verification Commands**

bash

docker history kiran236/v3 *# View layer sizes*

docker scan kiran236/v3 *# Security audit*

**6. Pro Tips**

1. **Always:**
   * Use .dockerignore to exclude build files
   * Pin base image versions (FROM maven:3.8.6 not FROM maven)
2. **Never:**
   * Store secrets in layers (use secrets management)
   * Use latest tag in production

**7. Homework Assignment**

1. **Optimize This:**

dockerfile

Copy

Download

FROM ubuntu

RUN apt update

RUN apt install -y python3

COPY . .

RUN pip install -r requirements.txt

CMD ["python3", "app.py"]

1. **Target:** Reduce size by 50% using multi-stage builds

**Submission:** Share your Dockerfile on LinkedIn with #DockerOptimization

**8. Key Takeaways**

1. **Distroless = Maximum Security** (but no debugging tools)
2. **Multi-Stage = Smaller Images** (separate build/runtime)
3. **Fewer Layers = Better Performance** (merge RUN commands)

**Next Video:** Docker Compose for Microservices!

This note combines:  
✅ **Technical comparisons** (tables showing before/after metrics)  
✅ **Actionable code snippets** (optimized Dockerfile examples)  
✅ **Real-world benchmarks** (size reduction measurements)  
✅ **Security considerations** (distroless tradeoffs)