Multi-Framework Dockerfile Reference Points

Vue.js Applications

Stage 1: Base Stage

- **Install System Dependencies**

```dockerfile
RUN apk add --no-cache git curl bash ca-certificates && rm -rf /var/cache/apk/*
```

- **Build Dependencies Contains**

```dockerfile
FROM node:18-alpine AS base
WORKDIR /app
RUN addgroup -g 1001 -S nodejs && adduser -S vueuser -u 1001
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

```dockerfile
COPY package.json package-lock.json* yarn.lock* pnpm-lock.yaml* ./
```

- **Install Dependencies with Caching**

```dockerfile
RUN npm ci --only=production=false --no-audit --prefer-offline
```

- **Install Development Dependencies**

```dockerfile
RUN npm ci --only=production --no-audit --prefer-offline
```

Stage 3: Build Stage

- **Copy Source Code**

dockerfile

```
COPY . .
ENV NODE_ENV=production
```

- **Code Quality Checks**

dockerfile

```
RUN npm run lint || echo "Lint completed"
RUN npm run type-check || echo "Type check completed"
```

- **Run Tests**

dockerfile

```
RUN npm run test:unit -- --run --coverage
```

- **Build Application**

dockerfile

```
RUN npm run build
```

- **Verify Build Artifacts**

dockerfile

```
RUN ls -la dist/ && test -f dist/index.html
```

Stage 4: Production Stage

- **Copy Build Artifacts**

dockerfile

```
COPY --from=build /app/dist /usr/share/nginx/html
```

Angular Applications

Stage 1: Base Stage

- **Install System Dependencies**

dockerfile

```
RUN apk add --no-cache git curl bash ca-certificates && rm -rf /var/cache/apk/*
```

- **Build Dependencies Contains**

dockerfile

```
FROM node:18-alpine AS base
WORKDIR /app
RUN npm install -g @angular/cli
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

dockerfile

```
COPY package.json package-lock.json ./
```

- **Install Dependencies with Caching**

dockerfile

```
RUN npm ci --no-audit --prefer-offline
```

Stage 3: Build Stage

- **Copy Source Code**

dockerfile

```
COPY . .
```

- **Code Quality Checks**

dockerfile

```
RUN ng lint || echo "Lint completed"
RUN npm run test -- --watch=false --browsers=ChromeHeadless --code-coverage
```

- **Build Application**

dockerfile

```
RUN ng build --configuration=production
```

- **Verify Build Artifacts**

dockerfile

```
RUN ls -la dist/ && test -d dist/
```

Stage 4: Production Stage

- **Copy Build Artifacts**

dockerfile

```
COPY --from=build /app/dist /usr/share/nginx/html
```

Static Site Generators (Gatsby)

Stage 1: Base Stage

- **Build Dependencies Contains**

dockerfile

```
FROM node:18-alpine AS base
WORKDIR /app
RUN npm install -g gatsby-cli
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

dockerfile

```
COPY package.json package-lock.json gatsby-config.js ./
```

- **Install Dependencies with Caching**

dockerfile

```
RUN npm ci --no-audit --prefer-offline
```

Stage 3: Build Stage

- **Copy Source Code**

```dockerfile
COPY . .
```

- **Build Application**

```dockerfile
RUN gatsby build
```

- **Verify Build Artifacts**

```dockerfile
RUN ls -la public/ && test -f public/index.html
```

Stage 4: Production Stage

- **Copy Build Artifacts**

```dockerfile
COPY --from=build /app/public /usr/share/nginx/html
```

Next.js Applications

Stage 1: Base Stage

- **Build Dependencies Contains**

```dockerfile
FROM node:18-alpine AS base
WORKDIR /app
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

dockerfile

```
COPY package.json package-lock.json next.config.js* ./
```

- **Install Dependencies with Caching**

dockerfile

```
RUN npm ci --no-audit --prefer-offline
```

Stage 3: Build Stage

- **Copy Source Code**

dockerfile

```
COPY . .
ENV NEXT_TELEMETRY_DISABLED=1
```

- **Build Application**

dockerfile

```
RUN npm run build
```

Stage 4: Production Stage

- **Install Runtime Dependencies**

dockerfile

```
FROM node:18-alpine AS production
WORKDIR /app
```

- **Copy Build Artifacts**

dockerfile

```
COPY --from=build /app/.next/standalone ./
COPY --from=build /app/.next/static ./.next/static
COPY --from=build /app/public ./public
```

- **Copy Configuration Files**

dockerfile

```
EXPOSE 3000
CMD ["node", "server.js"]
```

Node.js with TypeScript

Stage 1: Base Stage

- **Build Dependencies Contains**

dockerfile

```
FROM node:18-alpine AS base
WORKDIR /app
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

dockerfile

```
COPY package.json package-lock.json tsconfig.json ./
```

- **Install Dependencies with Caching**

dockerfile

```
RUN npm ci --no-audit --prefer-offline
```

Stage 3: Build Stage

- **Copy Source Code**

dockerfile

```
COPY src/ ./src/
```

- **Code Quality Checks**

dockerfile

```
RUN npm run lint || echo "Lint completed"
RUN npm run type-check
```

- **Run Tests**

dockerfile

```
RUN npm run test -- --coverage
```

- **Build Application**

dockerfile

```
RUN npm run build
```

- **Verify Build Artifacts**

dockerfile

```
RUN ls -la dist/ && test -f dist/index.js
```

Stage 4: Production Stage

- **Install Runtime Dependencies**

dockerfile

```
FROM node:18-alpine AS production
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm ci --only=production --no-audit
```

- **Copy Build Artifacts**

dockerfile

```
COPY --from=build /app/dist ./dist
```

- **Copy Configuration Files**

dockerfile

```
EXPOSE 3000
CMD ["node", "dist/index.js"]
```

Go Applications

Stage 1: Base Stage

- **Install System Dependencies**

```dockerfile
RUN apk add --no-cache git ca-certificates tzdata
```

- **Build Dependencies Contains**

```dockerfile
FROM golang:1.21-alpine AS base
WORKDIR /app
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

```dockerfile
COPY go.mod go.sum ./
```

- **Install Dependencies with Caching**

```dockerfile
RUN go mod download && go mod verify
```

Stage 3: Build Stage

- **Copy Source Code**

```dockerfile
COPY . .
```

- **Code Quality Checks**

```dockerfile
RUN go vet ./...
RUN go fmt ./...
```

- **Run Tests**

```dockerfile
RUN go test -v -race -coverprofile=coverage.out ./...
```

- **Build Application**

```dockerfile
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o main .
```

- **Verify Build Artifacts**

```dockerfile
RUN ls -la main && file main
```

Stage 4: Production Stage

- **Install Runtime Dependencies**

```dockerfile
FROM alpine:latest AS production
RUN apk --no-cache add ca-certificates tzdata
WORKDIR /root/
```

- **Copy Build Artifacts**

```dockerfile
COPY --from=build /app/main .
```

- **Copy Configuration Files**

```dockerfile
EXPOSE 8080
CMD ["./main"]
```

Java/Spring Boot

Stage 1: Base Stage

- **Build Dependencies Contains**

```dockerfile
FROM openjdk:17-jdk-alpine AS base
WORKDIR /app
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

```dockerfile
COPY pom.xml ./
COPY mvnw ./
COPY .mvn .mvn
```

- **Install Dependencies with Caching**

```dockerfile
RUN ./mvnw dependency:go-offline -B
```

Stage 3: Build Stage

- **Copy Source Code**

```dockerfile
COPY src src
```

- **Code Quality Checks**

```dockerfile
RUN ./mvnw checkstyle:check || echo "Checkstyle completed"
```

- **Run Tests**

```dockerfile
RUN ./mvnw test
```

- **Build Application**

dockerfile

```
RUN ./mvnw clean package -DskipTests
```

- **Verify Build Artifacts**

dockerfile

```
RUN ls -la target/*.jar
```

Stage 4: Production Stage

- **Install Runtime Dependencies**

dockerfile

```
FROM openjdk:17-jre-alpine AS production
WORKDIR /app
```

- **Copy Build Artifacts**

dockerfile

```
COPY --from=build /app/target/*.jar app.jar
```

- **Copy Configuration Files**

dockerfile

```
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Python with Dependencies

Stage 1: Base Stage

- **Install System Dependencies**

dockerfile

```
RUN apk add --no-cache gcc musl-dev libffi-dev openssl-dev
```

- **Build Dependencies Contains**

dockerfile

```
FROM python:3.11-alpine AS base
WORKDIR /app
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

dockerfile

```
COPY requirements.txt requirements-dev.txt* ./
```

- **Install Dependencies with Caching**

dockerfile

```
RUN pip install --no-cache-dir -r requirements.txt
```

- **Install Development Dependencies**

dockerfile

```
RUN pip install --no-cache-dir -r requirements-dev.txt
```

Stage 3: Build Stage

- **Copy Source Code**

dockerfile

```
COPY . .
```

- **Code Quality Checks**

dockerfile

```
RUN flake8 . || echo "Linting completed"
RUN black --check . || echo "Format check completed"
RUN mypy . || echo "Type check completed"
```

- **Run Tests**

dockerfile

```
RUN pytest --cov=. --cov-report=term-missing
```

- **Security Audit**

dockerfile

```
RUN pip-audit || echo "Security audit completed"
```

- **Build Stage Cleanup**

dockerfile

```
RUN pip uninstall -y -r requirements-dev.txt
```

Stage 4: Production Stage

- **Install Runtime Dependencies**

dockerfile

```
FROM python:3.11-alpine AS production
WORKDIR /app
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
```

- **Copy Build Artifacts**

dockerfile

```
COPY --from=build /app .
```

- **Copy Configuration Files**

dockerfile

```
EXPOSE 8000
CMD ["python", "app.py"]
```