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"]
```

REST/GraphQL APIs

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 ./
COPY schema.graphql* ./
```

- **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 npm run lint || echo "Lint completed"
RUN npm run schema:validate || echo "Schema validation completed"
```

- **Run Tests**

```dockerfile
RUN npm run test:integration
RUN npm run test:unit
```

- **Build Application**

dockerfile

```
RUN npm run build
```

Stage 4: Production Stage

- **Copy Build Artifacts**

dockerfile

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

- **Copy Configuration Files**

dockerfile

```
EXPOSE 4000
CMD ["node", "dist/server.js"]
```

Microservices

Stage 1: Base Stage

- **Build Dependencies Contains**

dockerfile

```
FROM node:18-alpine AS base
WORKDIR /app
RUN apk add --no-cache curl netcat-openbsd
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

dockerfile

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

- **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 npm run lint
RUN npm run proto:validate
```

- **Run Tests**

dockerfile

```
RUN npm run test:unit
RUN npm run test:contract
```

- **Build Application**

dockerfile

```
RUN npm run build
```

Stage 4: Production Stage

- **Install Runtime Dependencies**

dockerfile

```
RUN apk add --no-cache curl netcat-openbsd
```

- **Copy Build Artifacts**

dockerfile

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

- **Copy Configuration Files**

dockerfile

```
COPY healthcheck.sh ./
HEALTHCHECK --interval=30s CMD ./healthcheck.sh
EXPOSE 3000 50051
CMD ["node", "dist/server.js"]
```

CLI Tools (Go)

Stage 1: Base Stage

- **Build Dependencies Contains**

dockerfile

```
FROM golang:1.21-alpine AS base
WORKDIR /app
RUN apk add --no-cache git ca-certificates
```

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 ./...
```

- **Build Application**

dockerfile

```
RUN CGO_ENABLED=0 GOOS=linux go build -ldflags="-w -s" -o cli-tool .
```

- **Verify Build Artifacts**

dockerfile

```
RUN ./cli-tool --version
```

Stage 4: Production Stage

- **Install Runtime Dependencies**

dockerfile

```
FROM alpine:latest AS production
RUN apk --no-cache add ca-certificates
```

- **Copy Build Artifacts**

dockerfile

```
COPY --from=build /app/cli-tool /usr/local/bin/
```

- **Copy Configuration Files**

```dockerfile
ENTRYPOINT ["cli-tool"]
```

CLI Tools (Rust)

Stage 1: Base Stage

- **Build Dependencies Contains**

```dockerfile
FROM rust:1.70-alpine AS base
WORKDIR /app
RUN apk add --no-cache musl-dev
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

```dockerfile
COPY Cargo.toml Cargo.lock ./
```

- **Install Dependencies with Caching**

```dockerfile
RUN cargo fetch
```

Stage 3: Build Stage

- **Copy Source Code**

```dockerfile
COPY src src
```

- **Code Quality Checks**

dockerfile

```
RUN cargo fmt -- --check
RUN cargo clippy -- -D warnings
```

- **Run Tests**

dockerfile

```
RUN cargo test
```

- **Build Application**

dockerfile

```
RUN cargo build --release
```

- **Verify Build Artifacts**

dockerfile

```
RUN ls -la target/release/ && ./target/release/cli-tool --version
```

Stage 4: Production Stage

- **Install Runtime Dependencies**

dockerfile

```
FROM alpine:latest AS production
RUN apk --no-cache add ca-certificates
```

- **Copy Build Artifacts**

dockerfile

```
COPY --from=build /app/target/release/cli-tool /usr/local/bin/
```

- **Copy Configuration Files**

dockerfile

```
ENTRYPOINT ["cli-tool"]
```

CLI Tools (C++)

Stage 1: Base Stage

- **Install System Dependencies**

```dockerfile
RUN apk add --no-cache build-base cmake git
```

- **Build Dependencies Contains**

```dockerfile
FROM alpine:latest AS base
WORKDIR /app
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

```dockerfile
COPY CMakeLists.txt ./
COPY conanfile.txt* ./
```

- **Install Dependencies with Caching**

```dockerfile
RUN cmake -B build -S .
```

Stage 3: Build Stage

- **Copy Source Code**

```dockerfile
COPY src src
COPY include include
```

- **Build Application**

dockerfile

```
RUN cmake --build build --config Release
```

- **Run Tests**

dockerfile

```
RUN ctest --test-dir build
```

- **Verify Build Artifacts**

dockerfile

```
RUN ls -la build/ && ./build/cli-tool --version
```

Stage 4: Production Stage

- **Install Runtime Dependencies**

dockerfile

```
FROM alpine:latest AS production
RUN apk --no-cache add libstdc++
```

- **Copy Build Artifacts**

dockerfile

```
COPY --from=build /app/build/cli-tool /usr/local/bin/
```

- **Copy Configuration Files**

dockerfile

```
ENTRYPOINT ["cli-tool"]
```

Static Site Generators (Hugo)

Stage 1: Base Stage

- **Build Dependencies Contains**

dockerfile

```
FROM klakegg/hugo:ext-alpine AS base
WORKDIR /app
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

dockerfile

```
COPY config.yaml package.json* ./
```

- **Install Dependencies with Caching**

dockerfile

```
RUN if [ -f package.json ]; then npm ci; fi
```

Stage 3: Build Stage

- **Copy Source Code**

dockerfile

```
COPY . .
```

- **Build Application**

dockerfile

```
RUN hugo --minify
```

- **Verify Build Artifacts**

dockerfile

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

Stage 4: Production Stage

- **Copy Build Artifacts**

dockerfile

```
FROM nginx:alpine AS production
COPY --from=build /app/public /usr/share/nginx/html
```

Static Site Generators (Jekyll)

Stage 1: Base Stage

- **Build Dependencies Contains**

dockerfile

```
FROM ruby:3.1-alpine AS base
WORKDIR /app
RUN apk add --no-cache build-base
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

dockerfile

```
COPY Gemfile Gemfile.lock ./
```

- **Install Dependencies with Caching**

dockerfile

```
RUN bundle install --frozen
```

Stage 3: Build Stage

- **Copy Source Code**

dockerfile

```
COPY . .
```

- **Build Application**

dockerfile

```
RUN bundle exec jekyll build
```

- **Verify Build Artifacts**

dockerfile

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

Stage 4: Production Stage

- **Copy Build Artifacts**

dockerfile

```
FROM nginx:alpine AS production
COPY --from=build /app/_site /usr/share/nginx/html
```

CMS 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 ./
COPY strapi/ ./strapi/
```

- **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 npm run build
```

- **Verify Build Artifacts**

dockerfile

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

Stage 4: Production Stage

- **Install Runtime Dependencies**

dockerfile

```
FROM node:18-alpine AS production
WORKDIR /app
RUN apk add --no-cache sqlite
```

- **Copy Build Artifacts**

dockerfile

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

- **Copy Configuration Files**

dockerfile

```
COPY database/ ./database/
EXPOSE 1337
CMD ["npm", "start"]
```

Laravel Applications

Stage 1: Base Stage

- **Install System Dependencies**

```dockerfile
RUN apt-get update && apt-get install -y \
    git curl libpng-dev libonig-dev libxml2-dev zip unzip
```

- **Build Dependencies Contains**

```dockerfile
FROM php:8.2-fpm AS base
WORKDIR /app
RUN docker-php-ext-install pdo_mysql mbstring exif pcntl bcmath gd
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

```dockerfile
COPY composer.json composer.lock ./
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
```

- **Install Dependencies with Caching**

```dockerfile
RUN composer install --no-dev --optimize-autoloader --no-interaction
```

Stage 3: Build Stage

- **Copy Source Code**

```dockerfile
COPY . .
```

- **Code Quality Checks**

dockerfile

```
RUN php artisan config:clear
RUN php artisan route:clear
```

- **Run Tests**

dockerfile

```
RUN php artisan test
```

- **Build Application**

dockerfile

```
RUN php artisan config:cache
RUN php artisan route:cache
RUN php artisan view:cache
```

- **Security Audit**

dockerfile

```
RUN composer audit || echo "Security audit completed"
```

Stage 4: Production Stage

- **Install Runtime Dependencies**

dockerfile

```
FROM php:8.2-fpm AS production
WORKDIR /app
RUN docker-php-ext-install pdo_mysql
```

- **Copy Build Artifacts**

dockerfile

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

- **Copy Configuration Files**

dockerfile

```dockerfile
COPY .env.production .env
EXPOSE 9000
CMD ["php-fpm"]
```

Symfony Applications

Stage 1: Base Stage

- **Install System Dependencies**

dockerfile

```dockerfile
RUN apt-get update && apt-get install -y \
    git curl libicu-dev libzip-dev zip unzip
```

- **Build Dependencies Contains**

dockerfile

```dockerfile
FROM php:8.2-fpm AS base
WORKDIR /app
RUN docker-php-ext-install intl zip pdo_mysql opcache
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

dockerfile

```dockerfile
COPY composer.json composer.lock symfony.lock ./
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
```

- **Install Dependencies with Caching**

dockerfile

```dockerfile
RUN composer install --no-dev --optimize-autoloader --no-interaction
```

Stage 3: Build Stage

- **Copy Source Code**

dockerfile

```
COPY . .
```

- **Code Quality Checks**

dockerfile

```
RUN php bin/console lint:container
RUN php bin/console lint:yaml config/
```

- **Run Tests**

dockerfile

```
RUN php bin/phpunit
```

- **Build Application**

dockerfile

```
RUN php bin/console cache:clear --env=prod
RUN php bin/console assets:install --env=prod
```

Stage 4: Production Stage

- **Copy Build Artifacts**

dockerfile

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

- **Copy Configuration Files**

dockerfile

```
COPY .env.prod .env.local
EXPOSE 9000
CMD ["php-fpm"]
```

WordPress (Modern)

Stage 1: Base Stage

- **Install System Dependencies**

dockerfile

```
RUN apt-get update && apt-get install -y \
    libfreetype6-dev libjpeg62-turbo-dev libpng-dev libzip-dev
```

- **Build Dependencies Contains**

dockerfile

```
FROM php:8.1-fpm AS base
WORKDIR /var/www/html
RUN docker-php-ext-configure gd --with-freetype --with-jpeg
RUN docker-php-ext-install -j$(nproc) gd mysqli zip opcache
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

dockerfile

```
COPY composer.json composer.lock ./
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
```

- **Install Dependencies with Caching**

dockerfile

```
RUN composer install --no-dev --optimize-autoloader
```

Stage 3: Build Stage

- **Copy Source Code**

dockerfile

```
COPY . .
```

- **Build Application**

dockerfile

```
RUN if [ -f package.json ]; then npm ci && npm run build; fi
```

- **Verify Build Artifacts**

dockerfile

```
RUN ls -la wp-content/themes/ && ls -la wp-content/plugins/
```

Stage 4: Production Stage

- **Copy Build Artifacts**

dockerfile

```
COPY --from=build /var/www/html .
```

- **Copy Configuration Files**

dockerfile

```
COPY wp-config-production.php wp-config.php
EXPOSE 9000
CMD ["php-fpm"]
```

CakePHP

Stage 1: Base Stage

- **Install System Dependencies**

dockerfile

```
RUN apt-get update && apt-get install -y \
    libicu-dev libzip-dev zip unzip
```

- **Build Dependencies Contains**

dockerfile

```
FROM php:8.2-fpm AS base
WORKDIR /app
RUN docker-php-ext-install intl zip pdo_mysql opcache
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

dockerfile

```
COPY composer.json composer.lock ./
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
```

- **Install Dependencies with Caching**

dockerfile

```
RUN composer install --no-dev --optimize-autoloader
```

Stage 3: Build Stage

- **Copy Source Code**

dockerfile

```
COPY . .
```

- **Run Tests**

dockerfile

```
RUN ./vendor/bin/phpunit
```

- **Build Application**

dockerfile

```
RUN bin/cake cache clear_all
```

Stage 4: Production Stage

- **Copy Build Artifacts**

dockerfile

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

- **Copy Configuration Files**

dockerfile

```
COPY config/app_production.php config/app_local.php
EXPOSE 9000
CMD ["php-fpm"]
```

PHP-Apache (General)

Stage 1: Base Stage

- **Install System Dependencies**

dockerfile

```
RUN apt-get update && apt-get install -y \
    libpng-dev libjpeg-dev libfreetype6-dev libzip-dev
```

- **Build Dependencies Contains**

dockerfile

```
FROM php:8.2-apache AS base
WORKDIR /var/www/html
RUN docker-php-ext-install gd zip pdo_mysql
RUN a2enmod rewrite
```

Stage 2: Dependencies Stage

- **Copy Dependency Files First**

dockerfile

```
COPY composer.json composer.lock* ./
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
```

- **Install Dependencies with Caching**

dockerfile

```
RUN if [ -f composer.json ]; then composer install --no-dev --optimize-autoloader; fi
```

Stage 3: Build Stage

- **Copy Source Code**

dockerfile

```
COPY . .
```

- **Code Quality Checks**

dockerfile

```
RUN php -l index.php
```

- **Build Application**

dockerfile

```
RUN if [ -f package.json ]; then npm ci && npm run build; fi
```

Stage 4: Production Stage

- **Copy Build Artifacts**

dockerfile

```
COPY --from=build /var/www/html .
```

- **Copy Configuration Files**

dockerfile

```
COPY .htaccess ./
COPY php.ini /usr/local/etc/php/
EXPOSE 80
CMD ["apache2-foreground"]
```