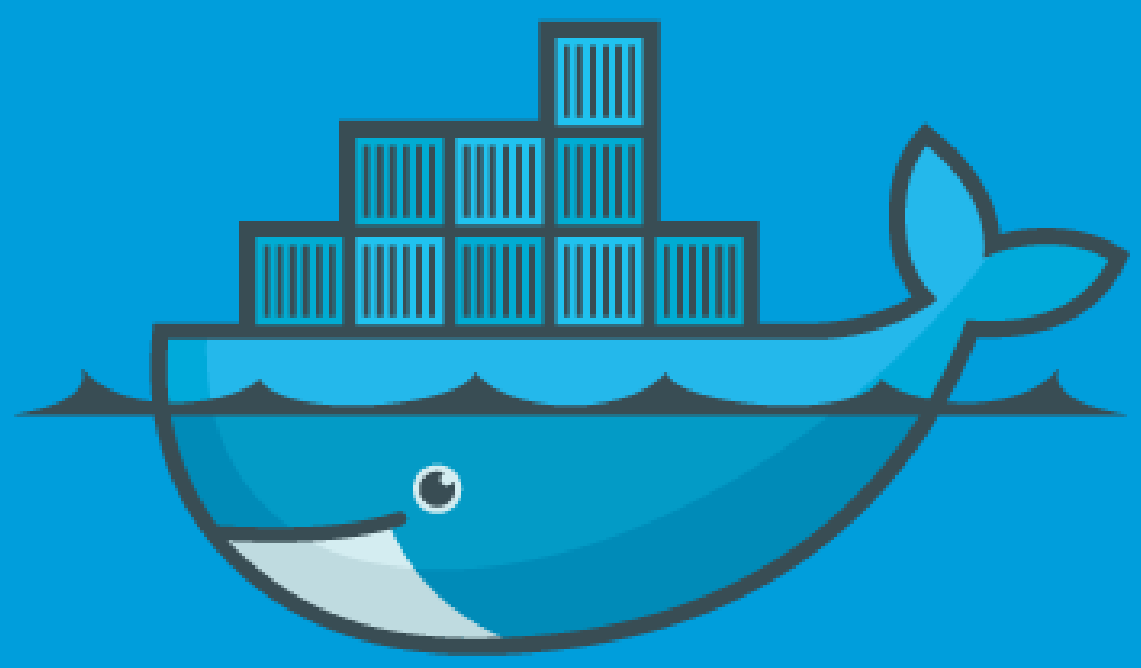
10 Best Practices for Optimizing Docker Images

Learn how to build lightweight, secure, and efficient Docker images.



docker

Read more about it

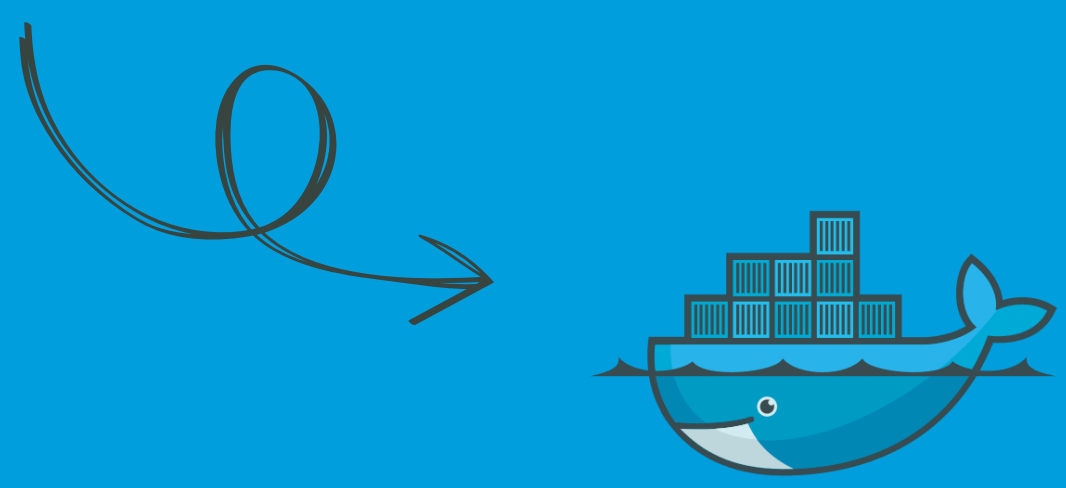
1- Use Official Base Images

Start with official, well-maintained images (e.g., from Docker Hub) to ensure security, stability, and support.

2- Choose Minimal Base Images

Use smaller base images like *alpine* or *slim* to reduce image size and the attack surface, leading to faster builds and improved security.

150 MB



30 MB

3- Reduce the Number of Layers

Combine related commands in a single *RUN* statement to minimize layers and create a more efficient image.

Dockerfile

```
RUN apt-get update && \
    apt-get install -y python3 python3-pip && \
    apt-get clean
```

4- Leverage Multi-Stage Builds

Use multi-stage builds to optimize image size by copying only essential files to the final production image.

Dockerfile

```
FROM golang:1.17 AS builder
# Build in stage 1
COPY . .
RUN go build -o app

FROM alpine
# Copy only the executable
COPY --from=builder /app /usr/local/bin/app
CMD ["app"]
```

5- Use .dockerignore File

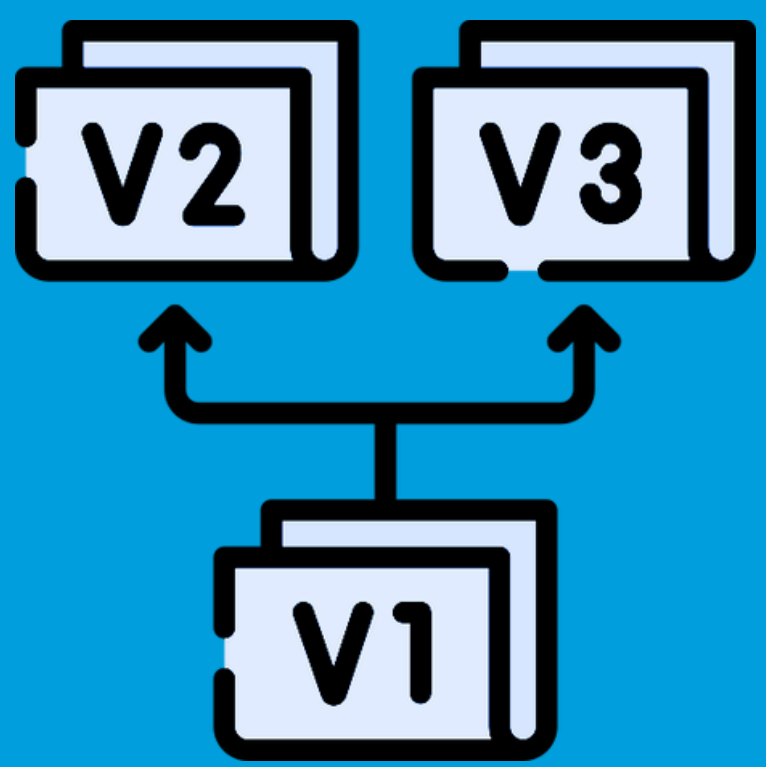
Exclude unnecessary files (e.g., *.git*, *node_modules*, *temporary files*) to reduce the image size and improve build times.

.dockerignore

```
.git
node_modules
temp/
Dockerfile
```

6- Set Explicit Image Tags

Avoid using *latest* tags. Always specify an image version (e.g., *node:14.17-alpine*) to ensure consistency and avoid breaking changes.



7- Minimize the Number of Packages

Install only the essential packages needed for your application to reduce image size and potential vulnerabilities.

Dockerfile

```
RUN apt-get install -y curl
```

8- Use Non-Root User

Run your container as a non-root user for security reasons, preventing potential privilege escalations.

Dockerfile

```
RUN useradd -ms /bin/bash appuser
USER appuser
```

9- Keep Image Layers Clean

Remove build-time dependencies, cache, and temporary files after installation to keep your image clean and efficient.

Dockerfile

```
RUN apt-get install -y build-essential && \
rm -rf /var/lib/apt/lists/*
```

10- Optimize Caching with Layer Order

Organize your Dockerfile so that layers that change less frequently (e.g., *installing dependencies*) are placed earlier to maximize caching and speed up rebuilds.

Dockerfile

```
# Install dependencies early for caching
COPY requirements.txt .
RUN pip install -r requirements.txt

# Add application code later
COPY . .
```

Recap: Key Docker Image Optimization Tips

Dockerfile

1. **Use Official Base Images.**
2. **Choose Minimal Base Images.**
3. **Reduce the number of layers.**
4. **Leverage multi-stage builds.**
5. **Use .dockerignore to exclude unnecessary files.**
6. **Set explicit image tags.**
7. **Minimize the Number of Packages.**
8. **Use a non-root user.**
9. **Keep Image Layers Clean.**
10. **Optimize Caching with Layer Order.**