Container Image Minimization Guidelines

Motivation: The importance of container image size

Small container images offer the following benefits:

- Reduced build time
- Reduced storage space usage
- Reduced download time
- Faster deployments
- Better security due to smaller surface attack

Image layers

To reduce container image sizes, it's important to understand that container images are composed of multiple layers (think of them as intermediate images) which form the final image.

At build time, each Dockerfile instruction creates an extra layer.

```
FROM ubuntu                          # 4 layers (base image)
MAINTAINER adolfo@orangemonk.net     # 1 layer
RUN mkdir -p /app/bin                # 1 layer
RUN apt-get install -y wget          # 1 layer
```

So, generally speaking, reducing image size is the art of reducing the number of layers.

Guidelines

1.Choose a small base image

Base images are container images that do not have a parent. Container images usually have their own root filesystem with

an operating system installed. Typically, you will create a new image using a base image that contains an operating system. Consider the following when selecting a base image:

- Operating system distribution: Determine if your application requires specific libraries or tools from a specific operating system
- Base image size: Use small base images, such as busybox (2 MB) or alpine (5 MB). In contrast, a standard minimal operating system, such as Fedora or CentOS could be up to 200MB in size.
- Updates: Use only base images from "official repositories", such as Docker Hub.

2. Be mindful of where your Dockerfile is located (i.e. understand the build context)

Irrespective of where the `Dockerfile` is located, all recursive contents of files and directories in the current directory are sent to the Docker daemon as the build context.

Inadvertently including files that are not necessary for building an image results in a larger build context and larger image size.

3. Use multi-stage builds

Use multi-stage builds, to only copy the artifacts you need into the final image.

Tools and debug information can be added to intermediate build stages without increasing the size of the final image.

4.Don't install unnecessary packages

Avoid installing extra or unnecessary packages. This will reduce complexity, dependencies, build times and image sizes.

For example, don't include a text editor in a database image.

5.Minimize the number of layers

`RUN`, `COPY`, `ADD` instructions create layers and increase the size of image.

To reduce the number of layers and the image size, don't use `ADD` to download packages from URLs. Use `curl` or `wget and` delete the files you no longer need after they've been extracted.

Example:

```
RUN mkdir -p /usr/src/ether \
    && curl -SL http://vacuum.com/huge.tar.xz \
    | tar -xJC /usr/src/ether \
    && make -C /usr/src/ether all
```

6.Chain commands together to reduce the number of layers

A new container layer is created for every new instruction in the Dockerfile. Commands that are chained together become part of the same image layer.

To provide a good understanding of the content of each layer, group related operations together so that they become part of a single layer.

Take for example:

RUN dnf install -y --setopt=tsflags=nodocs \

httpd vim && \

systemctl enable httpd && \

dnf clean all

Each command associated with the installation and configuration of httpd is grouped and creates a single layer. Grouping operations this way reduces the number of layers and contributes to the legibility of the instructions.