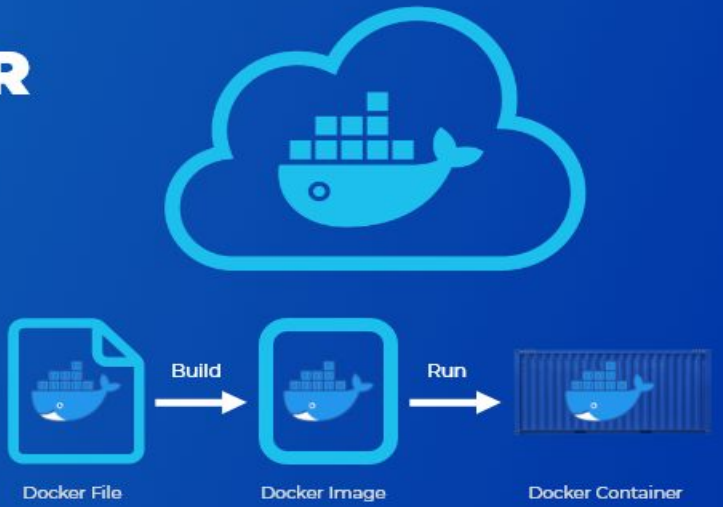
9 Best practices for Docker Image

Fibonalabs

9 BEST PRACTICES FOR DOCKER IMAGE



Docker is an Open-sourced platform which provides a framework for packaging and distributing containerized applications to build and run containers. A Docker build is run on any suitable OS with additional tools and services to facilitate automation. In order to execute a code in the container, we make use of a file called Docker image.

DOCKER IMAGE AND ITS KEY CONCEPTS

Docker image acts as a template for building a Docker container and is the beginning point in Docker usage. A Docker image contains Application code, libraries, configuration files, run times, dependencies, tools and all files which are necessary to run an application.

The key concepts which are needed to understand Docker Image are listed below:

- **Image Layers:** The files that make up a Docker image are called as image layers. The layers are built on top of one another and are dependent on the immediate layer below it. Defining the layer hierarchy is important because when a change is made in a particular layer, Docker rebuilds all the layers that were built from it. So, it is advisable to place changing layers are the topmost positions.
- **Container Layer:** Docker adds a thin writable layer each time when it launches a container. This layer stores all the changes which occurred in the container during runtime.

- **Parent Image:** The first layer of the Docker Image is called the parent layer. It acts as the foundation for the other layers. A wide-range of readymade images is available on the public container registry Docker Hub and on third party services like Google Container Registry. There is also an option to use our own images.
- **Base Image:** Base image is the empty first layer which facilitates building Docker Images from scratch. It is often used by advanced Docker professionals.
- **Docker Manifest:** This is an additional file which includes the description of the image in JSON format. It contains image tags, digital signature and steps to configure the container for different host platforms.

- **Container Repositories:** They are physical locations where the Docker images are stored. The images are grouped based on their relativity and naming.
- **Container Registries:** They are catalogs of the repositories, where we can push and pull container images.

CREATING A DOCKER IMAGE

There are two methods to create a Docker Image. They are Interactive method and Docker File Method.

- In interactive method, the user runs a container from an existing Docker Image. The environment is changed manually before saving the image. The first step of this method is to launch Docker and open a terminal session.

The Docker command for run ‘image_name:tag_name’ is executed and this starts a shell session with the container that was launched with the image.

In the Dockerfile approach, a plaintext Dockerfile is made. It gives specifications for creating an image. Here the required commands are added in the Docker file. After the Dockerfile is started, the .dockerignore file is set up. This file helps in excluding files which are not needed for the final build. Then the Docker build command is used to create the Docker image with a name and appropriate tags are set. The created images are viewed by using the Docker images command.

BEST PRACTICES FOR BUILDING AN IMAGE

The following are a few best practices which can be followed to build an efficient and secure Docker Image:

1.COMPATIBLE AND MINIMAL BASE IMAGES:

Each image in Docker is built on top of an existing base image. It is an added advantage if the base image is lightweight and minimal. Sometimes a minimal image may ignore the compatibility factors such as presence of required libraries and dependencies needed to run the application. In this case, manual installation of the same is required, which may be a time-consuming task. We may install unnecessary packages and expose ports which compromise the security of the whole system.

The language which we work on plays an important role in this case. For example, Elixir is a complied language where the resulting compiled binary properties are self-contained and can run on any Linux application. Here Alpine images can be a perfect fit. In this case the base image is only 5MB. After compilation, images sized 25MB can be produced, which are stable and optimal.

2. REFRAIN FROM USING MULTIPLE LAYERS

Adding layers which are necessary and ordering them correctly results in optimized Docker Image productions. While the older versions required minimization of layers manually, the new versions include the following feature to overcome this issue:

- Only RUN, COPY, ADD instructions create layers. The other instructions create temporary intermediate images. They do not increase the size of the build.

- Multi-stage builds are used when need. Only the artifacts which are needed are copied into the final image. This allows us to include debug tools and information in intermediary build stages without increasing the size of the final image. Multi stage builds help in leveraging build cache, minimizing image layers.

3.LEVERAGING BUILD CACHE

Build cache helps in speeding up the whole image creation process. However when we choose to create the image from scratch, there is a need to invalidate existing cache builds. If we don’t want to utilize cache, we can use the –no-cache=true option on the docker build command.

4.AVOID USING ROOT AS ENTRY POINT

It is always better to avoid using root as the container entry point for running the Docker images. The USER instruction is used to switch between the default effective UID and a non-root user. Sometimes the execution environment blocks containers running as roots by default e.g. Openshift. Also, the container should be run as a non-root user but the UID shouldn’t be made a requirement.

The following instructions are to be followed to run a non-root image in the container:

- Ensure that the user specified in the USER instruction exists inside the container.
- Provide appropriate file system permission in the locations where the processes are read and written.

5.USE DISTROLESS IMAGES

Distroless images contain only the specific application and runtime dependencies. They do not contain package managers, shells or any other programs in a standard Linux distribution. This is a best practice employed by Google. These images can be built by using multi stage builds.

6.AVOID SECRETS IN YOUR IMAGES

The Docker Image should never contain any confidential information or configuration values about environments like production, staging etc. The image should be customized by injecting the values on runtime, especially if it’s confidential. The configuration files must be included with safe or dummy values.

COPY function can be used to carry out this task as it is more predictable and has lesser chances of error. Also, the RUN instruction can be used to download and extract a package and remove the original file. This also helps in layer reduction. These functions are more secure than the ADD function.

7.INCLUDE METADATA LABELS

Including metadata labels during image building helps in image management activities. These can include managing application versions, website links, maintenance contact methods, etc.

8.RESTRICT APP CAPABILITIES

This step is to be followed post image building as a security measure. The application capabilities can be restricted to minimal required set using –cap-drop in Docker or securityContextcapabilitiesdrop in Kubernetes. This ensures that even if the container or image is compromised, the attacker’s range of action is limited.

9.LINTING, SCANNING AND VERIFYING

Using linting tools like Haskell Dockerfile linter can help in detecting bad practices in the Docker files and expose issues inside shell commands executed by RUN instruction.

Image scanning should be done as different stages in the software development cycle and also when the image is pushed to the container. The common misconfigurations are images running as root, usage of ADD instructions, hardcoded secrets or damaged images. This scanning and health check helps in creating persistent images. It also guides us to upgrade the images when needed.

Docker content trust, Docker notary or similar tools can be used to digitally sign images. The signature must be then verified during run time. This ensures that the confidentiality of the image data is preserved.

THANK YOU