How to create images from Dockerfiles

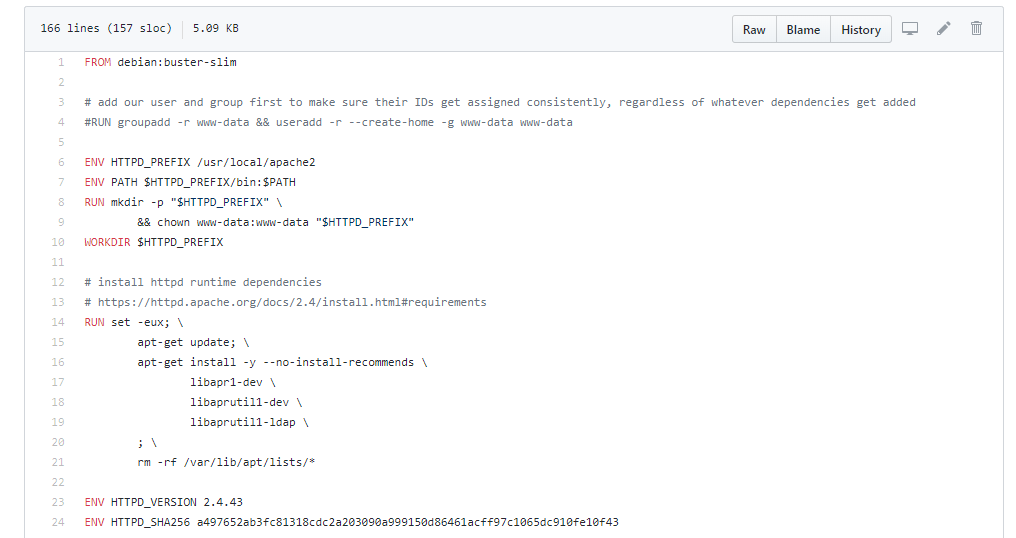
1. **Dockerfiles** are the basic templates of Docker images. They contain data, commands and steps to execute which serves as configuration for the images being built. Dockerfiles might contain the following:
   1. Reference to previously generated images (which serves as baseline for the build), or from **scratch.**
   2. Some (shell) commands, which perform activities such as:
      1. Downloading files for installation, configuration, or as dependencies to be used later on the container instance.
      2. Creation/deletion of files and directories, which serves as optimization for the image’s filesystem. The image’s filesystem only contains files, binaries and directories that are needed by container to perform its assigned function. This makes the image sizes smaller, hence easier to distribute and pull from repositories. Some files can also be transferred from one location to another inside the container instance depending on how they would be used in applications.
      3. Defining and configuration of environment variables needed by the container and the applications installed.
      4. Triggering of commands to start, reset, or stop certain services running within the container instance.
      5. Other housekeeping, cleanup, and optimization tasks.
   3. Configuration data, such as environment variables, directory references, network configuration, or default port assignments.
   4. Default command to execute once the image build has been completed.
   5. Integration information, such as endpoint data (URLs, resource name definitions) to access the container, its data, and its functionality.
   6. Other relevant information to configure the container to communicate with other containers, or to orchestrate actions of multiple containers.

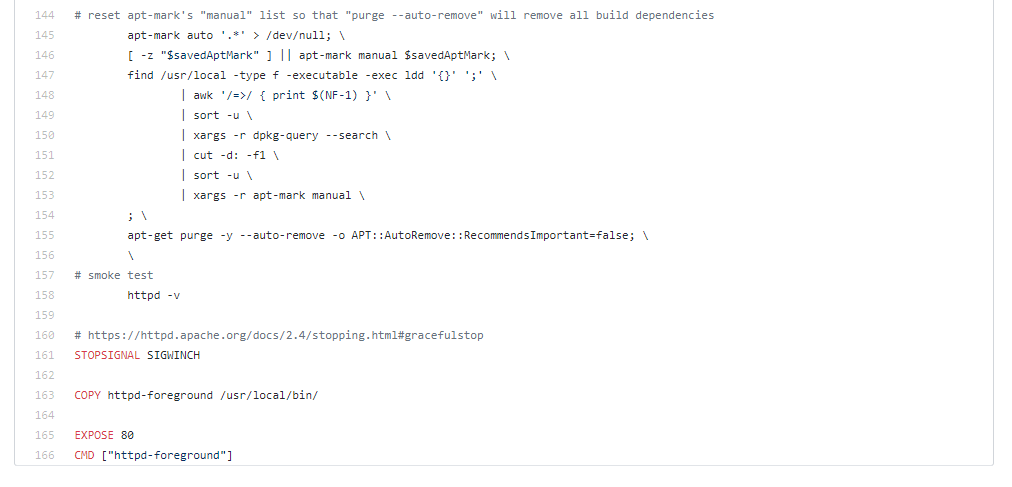
These portions are marked by keywords that indicate the function that they perform. Examples of these keywords are explained below:

1. **ARG, FROM** – Signifies a call to a previously built image, or from **scratch** (blank reference)**.**

When this keyword is run, it will pull a copy of an image based on the parameters included, and it would serve as base model to begin the image build. ARGs provide a way to customize the fetching of images depending on the version needed, specific release tags of image releases, and to add a dynamic configuration feature for the baseline image creation.

1. **ENV** – These are environment variables that are defined to be referenced by the applications and services that would be running on the container instance.
2. **WORKDIR, VOLUME** – Defines directories and volumes to be used and referenced by the applications and services that would run on the container.
3. **RUN** – These are mostly shell scripts intended to be run on specific parts, and sequences, to trigger internal actions within the container and its installed applications and services.
4. **COPY, ADD, LABEL** – Commands to manipulate data, to be used in processing the image build, or as metadata to describe the image build details.
5. **ENDPOINT, EXPOSE** – Keywords that provide integration points to the images and containers.
6. **STOPSIGNAL** – Triggers the system call signal that will be sent to the container to exit.
7. **Anatomy of a Dockerfile**





A sample Dockerfile (Apache HTTP Server - httpd)

1. How to create images from Dockerfiles.
   1. Activities when creating images from Dockerfiles can be grouped as follows:
      1. **Image Build** – Tasks related to the start of the image build, and configuration tasks to be used with the current build.
      2. **Instance Creation** – Tasks which involve creating container instances from the generated image. This may be part of testing the image build, or the actual usage of the container for application deployment.
      3. **Image and Container Management** – This is the part where we manage images and containers. Pulling of reference images, build image completion, container creation, intermediary images creation, image uploads to repositories, and cleanup of unused images and containers.
   2. Fill up the Dockerfile with the actions and data that need to be configured to the build. For example, to create a container to host Java applications, a step to install Java would need to be performed. This might include downloading the package, configuring the container’s package manager with a reference location to download and install the package. This also include the installation execution details, which may include building of source code, and the creation of needed binaries.
   3. Organize and orchestrate the sequence of activities for the build. You can set the early actions to downloading the needed dependencies and other configuration steps, then set the latter part of the build to executing the actions to set up the services and applications needed for the container build.
   4. Once the Dockerfile has been completed, execute the build by running the following command: ***(sudo\* - if Docker was installed with root privileges) docker build -t <image-repository-userid>/<repository-or-image-name>:<revision-number-or-tag >***. The build progress would be displayed after the command has been run. Depending on the configured verbosity level, each layer, or step in the build process will show a message indicating the part of the build being executed, and the progress of the overall build.
   5. Once the build completes, base images, intermediary images, and other reference images would be pulled from their sources and generated on the host machine. These can be managed later on, after the build is successful. Images and containers can be viewed in a list by **docker image / container ls -a**. Active and terminated containers can be viewed by ***docker container ps -a***. When no longer needed, these images can be removed to reclaim disk space. Images can be removed by ***docker image rm <image-repository-userid>/<repository-or-image-name>:<revision-number-or-tag > | <image-id>.*** In a similar manner, container instances can be cleaned up by ***docker container rm <image-repository-userid>/<repository-or-image-name>:<revision-number-or-tag > | <image-id>.*** Orphaned, or unused images and containers can be cleaned by ***docker image / container prune***
   6. Images can be run in many ways. Interactive, detached, headless, or remotely accessible. Use docker run -itd –name <assigned-container-name> <image-name-details>:<revision-tag>
2. sdsadsadsadsadsadasd