# An exercise in Discovery, Building Docker Images, using Makefiles & Docker Compose. – Part 1

## Docker images, Multi stage builds, Makefiles Projects with sub directories and their own Makefiles.

**Overview**

This all started with another [blog](https://medium.com/@georgelza/an-exercise-in-discovery-streaming-data-in-the-analytical-world-part-1-e7c17d61b9d2) I was writing, where in I was exploring Kafka, kSqlDB, Apache Flink, Apache Hive, Apache Iceberg & Apache Paimon and various other supporting bits and bobs.

During that “rabbit hole” I ended needing to build “some…” docker images, leading to allot of learning along the way and redoing things over and over to make it easier, faster, simpler.

Figured others might be interested, find value out of it.

I’m by no means an expert, this is purely what I learned along the way to make it easier for myself.

We will cover a basic docker image build (Ubuntu 20.04), installing some basic OS packages, installing application server (based OpenJDK 11) and Apache Hadoop DFS cluster, adding required configuration files (directly into the container during build or mounting them at run time) & dealing with environment variables.

During this phase, I will show 2 Dockerfile’s. One where I stage some files locally first and a 2nd where I pull the source files directly from the internet during the build.

I will additionally discuss the importance of the build order in your Dockerfile.

As a separate mini project, I will also show a docker image build using a multistage image build to reduce the image size and attack vectors.

As a sidetrack we will also have a look at using Makefiles, they are a great way to “package all your commands together.

To demonstrate all the above processes, we will build an Apache Hadoop DFS cluster, based on Ubuntu 20.04 / OpenJDK 11 / Apache Hadoop 3.3.8, comprised out of:

* namenode
* nodemanager
* resourcemanager
* datanodes &
* historyserver

As it stands this will be a X-part posting, but it’s by no means complete.

NOTE: I work on an Apple MacBook based on their ARM64 aka AARC64 architecture. Where needed I will point out which lines can be changed to make everything Intel/AMD64 compatible.

Good luck, as always, this is all fraught with rabbit holes, so many and you can disappear so easily… But it’s all fun and you will always discover something new or validate a previous learned skill.

See [Building Docker Images](ttps://github.com/georgelza/dockerimagebuilding.git) for all the code.

**About Me**

I’m a techie, a technologist, always curious, love data, have for as long as I can remember always worked with data in one form or the other, Database admin, Database product lead, data platforms architect, infrastructure architect hosting databases, backing it up, optimizing performance, accessing it. Data data data… it makes the world go round.

In recent years, pivoted into a more generic Technology Architect role, capable of full stack architecture.

[George Leonard](https://www.linkedin.com/in/george-leonard-945b502/)

[georgelza@gmail.com](mailto:georgelza@gmail.com)

------------------------------------------------------------------------------------------------------------------

# An exercise in Discovery, Building Docker Images, using Makefiles & Docker Compose. – Part 2

## Docker images, Multi stage builds, Makefiles Projects with sub directories and their own Makefiles.

Ok we will start with our base OS image. See the build-ubuntu-os-20.04 sub directory.

NOTE: this is one of our ARM64 specific images, to revert to standard AMD64 (Intel), simply remove the arm64v8/bit from the FROM clause.

FROM arm64v8/ubuntu:20.04

# original https://github.com/YanYunNN/hadoop-cluster-docker-m1/blob/main/Dockerfile

WORKDIR /root

RUN echo "--> Install OS dependencies openssh-server & misc tools" && \

build\_deps="openssh-server wget neovim curl unzip net-tools" && \

apt-get update && \

apt-get install -y $build\_deps

RUN echo "--> Purge apt artifacts" && \

apt-get purge -y --auto-remove $build\_deps && \

apt-get clean && \

rm -rf /var/lib/apt/lists/\*

We start our build by specifying our source image, in this case arm64v8/ubuntu:20.04

Next we define a WORKDIR /root this accomplishes 2 outcomes; it creates the directory if it does not exist, and it changes into the directory.

Following this we execute a docker primitive/command called RUN. I start the command with echo “some text” to output a description of what’s being done, followed by “&& \” which creates a line continue onto the next line.

RUN echo "--> Install OS dependencies openssh-server & misc tools" && \

build\_deps="openssh-server wget neovim curl unzip net-tools" && \

apt-get update && \

apt-get install -y $build\_deps

The second line we define a variable “build\_deps” with a list of packages assigned to it that we want to install using the apt-get install command:

build\_deps=”openssh-server wget neovim curl unzip net-tools”

The value of using a variable here is seen in the next RUN command where we clean up after the install and now instruct apt-get to clean up, using: apt-get purge – auto-remove &build\_deps variable/list define previously.

RUN echo "--> Purge apt artifacts" && \

apt-get purge -y --auto-remove $build\_deps && \

apt-get clean && \

rm -rf /var/lib/apt/lists/\*

NOTE: Lesson learned, the package vim during installation request’s physical location which can’t be by passed, neovim on the other hand not.

The build is executed by calling make build in the same directory, this in return issue:

sudo docker build -t ubuntu20.04:$(VERSION) .

To see the output image execute docker images

Ok, that’s all for Part 2, In the next part we will move onto installing Open JDK 11 application server onto our base image build.

**My Repo’s**

All the code used during this article will be available on the below GIT repo.

[Building Docker Images](ttps://github.com/georgelza/dockerimagebuilding.git)

In the root of the repo is a README.md file with some additional notes, this file is used in combination with the blog and might include some additional notes.

See the [Makefile](https://opensource.com/article/18/8/what-how-makefile" \t "_blank) in the same directory, where we can start by first pulling the source OS images, after which the various images can be build. Remember to go change the base OS architecture if you not on a MBP and look at the JAR files pulled, some of them are arm64/aarch64 specific, I will try and make notes in the various Dockerfile’s where needed.

Below is a diagram depicting the ancestry of the various images used inside the project.

And that’s it for now… Thank you for sticking with me through this exploration. All it did was create a small little to do list that is growing as I am typing this, of things/subjects I’d like to explore more and blog . Till next time.



**About Me**

I’m a techie, a technologist, always curious, love data, have for as long as I can remember always worked with data in one form or the other, Database admin, Database product lead, data platforms architect, infrastructure architect hosting databases, backing it up, optimizing performance, accessing it. Data data data… it makes the world go round.

In recent years, pivoted into a more generic Technology Architect role, capable of full stack architecture.

[George Leonard](https://www.linkedin.com/in/george-leonard-945b502/)

[georgelza@gmail.com](mailto:georgelza@gmail.com)

Some more References:

[Apache Flink](https://flink.apache.org/) originally by [Ververica](https://docs.ververica.com/)

* [Get Started - Installation](https://docs.ververica.com/vvp/getting-started/installation?_gl=1*a1ub31*_gcl_au*MjI3NTE0OTU0LjE3MjA2OTY4NDY).