***Docker-Build Kit***

Docker Build is one of the most used features of the Docker Engine - users ranging from developers, build teams, and release teams all use Docker Build.

Docker Build enhancements for 18.09 release introduces a much-needed overhaul of the build architecture. By integrating BuildKit, users should see an improvement on performance, storage management, feature functionality, and security.

* Docker images created with BuildKit can be pushed to Docker Hub just like Docker images created with legacy build
* the Dockerfile format that works on legacy build will also work with BuildKit builds
* The new --secret command line option allows the user to pass secret information for building new images with a specified Dockerfile

***To enable BuildKit builds***

Easiest way from a fresh install of docker is to set the DOCKER\_BUILDKIT=1 environment variable when invoking the docker build command, such as:

DOCKER\_BUILDKIT=1 docker build .

## *Overriding default frontends*

The new syntax features in Dockerfile are available if you override the default frontend. To override the default frontend, set the first line of the Dockerfile as a comment with a specific frontend image:

# syntax=<frontend image>, e.g. # syntax=docker/dockerfile:1.2

The examples on this page use features that are available in docker/dockerfile version 1.2.0 and up. We recommend using docker/dockerfile:1, which always points to the latest release of the version 1 syntax. BuildKit automatically checks for updates of the syntax before building, making sure you are using the most current version

***Features of Docker BuildKit***

### ***Faster builds using parallelism***

Consider the following multi-stage Dockerfile. By building in multiple stages, it enables both caching for fast rebuilds and smaller images in production.

FROM python:3.8-slim-buster AS build-stage

RUN apt-get update && apt-get install -y --no-install-recommends gcc

RUN python -m venv /venv

ENV PATH=/venv/bin:$PATH

RUN pip install pyrsistent

FROM python:3.8-slim-buster AS runtime-stage

RUN apt-get update && apt-get -y upgrade

COPY --from=build-stage /venv /venv

ENV PATH=/venv/bin:$PATH

ENTRYPOINT ["python", "-c", "import pyrsistent; print(pyrsistent.\_\_file\_\_)"]

On my computer, building this takes about 22 seconds with classic Docker. When I turn on BuildKit, however, it takes only 16 seconds.

**This is because BuildKit can build multiple stages in parallel.** Notice that the second stage image’s apt-get does not depend in any way on the first stage; the dependency happens only once the COPY --from=build-stage happens. BuildKit can figure that out and run the build steps in parallel until that dependency becomes a blocker.

## *Docker Build secret information*

The new --secret flag for docker build allows the user to pass secret information to be used in the Dockerfile for building docker images in a safe way that will not end up stored in the final image.

id is the identifier to pass into the docker build --secret. This identifier is associated with the RUN --mount identifier to use in the Dockerfile. Docker does not use the filename of where the secret is kept outside of the Dockerfile, since this may be sensitive information.

dst renames the secret file to a specific file in the Dockerfile RUN command to use.

Echo “password” > mysecret.txt

# syntax=docker/dockerfile:1.2

FROM alpine

# shows secret from default secret location:

RUN --mount=type=secret,id=mysecret cat /run/secrets/mysecret

# shows secret from custom secret location:

RUN --mount=type=secret,id=mysecret,dst=/foobar cat /foobar

The secret needs to be passed to the build using the --secret flag. This Dockerfile is only to demonstrate that the secret can be accessed. As you can see the secret printed in the build output. The final image built will not have the secret file:

## *Using a Cache Mount*

If you’re sick of re-downloading all external dependencies every time there’s a change to one of them, the [cache mount](https://github.com/moby/buildkit/blob/master/frontend/dockerfile/docs/syntax.md#run---mounttypecache) can help you save time in the future.

Inside of your Dockerfile, add a mount flag, specifying which directories should be cached during the step.

RUN --mount=type=cache,target=/var/cache/apt ...

You can cache multiple directories this way, by adding multiple –mount flags to the same RUN instruction. Those directories will be preserved in between builds, even if the step has to be re-executed and wouldn’t be able to make use of the usual Docker cache. Neat!

Think of this feature like volumes which you can use during your build. If done right, this feature will help to speed up the rebuilds of your images.

## *Passing SSH Credentials*

Accessing private repositories from within a Docker build used to be fiddly. As with other secrets, you didn’t want to leave behind your SSH credentials inside of your image layers.

With the new [SSH mount type](https://github.com/moby/buildkit/blob/master/frontend/dockerfile/docs/syntax.md#run---mounttypessh) you can allow your Docker build to make use of your host’s SSH keys.

RUN --mount=type=ssh ...

No need to handle this kind of secrets. Speaking of secrets…