Build images with BuildKit

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

For more information on build options, see the reference guide on the command line build options.

Requirements

- System requirements are docker-ce x86_64, ppc64le, s390x, aarch64, armhf; or docker-ee x86_64 only
- Network connection required for downloading images of custom frontends

Limitations

- Only supported for building Linux containers
- BuildKit mode is compatible with UCP 3.2 or newer

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 .
```

To enable docker BuildKit by default, set daemon configuration in `/etc/docker/daemon.json` feature to true and restart the daemon:

```
{ "features": { "buildkit": true } }
```

New Docker Build command line build output

New docker build BuildKit TTY output (default):

```
$ docker build .

[+] Building 70.9s (34/59)
 => [runc 1/4] COPY hack/dockerfile/install/install.sh ./install.sh       14.0s
 => [frozen-images 3/4] RUN /download-frozen-image-v2.sh /build  buildpa  24.9s
 => [containerd 4/5] RUN PREFIX=/build/ ./install.sh containerd           37.1s
 => [tini 2/5] COPY hack/dockerfile/install/install.sh ./install.sh        4.9s
 => [vndr 2/4] COPY hack/dockerfile/install/vndr.installer ./              1.6s
 => [dockercli 2/4] COPY hack/dockerfile/install/dockercli.installer ./    5.9s
 => [proxy 2/4] COPY hack/dockerfile/install/proxy.installer ./           15.7s
 => [tomlv 2/4] COPY hack/dockerfile/install/tomlv.installer ./           12.4s
 => [gometalinter 2/4] COPY hack/dockerfile/install/gometalinter.install  25.5s
 => [vndr 3/4] RUN PREFIX=/build/ ./install.sh vndr                       33.2s
 => [tini 3/5] COPY hack/dockerfile/install/tini.installer ./              6.1s
 => [dockercli 3/4] RUN PREFIX=/build/ ./install.sh dockercli             18.0s
 => [runc 2/4] COPY hack/dockerfile/install/runc.installer ./              2.4s
 => [tini 4/5] RUN PREFIX=/build/ ./install.sh tini                       11.6s
 => [runc 3/4] RUN PREFIX=/build/ ./install.sh runc                       23.4s
 => [tomlv 3/4] RUN PREFIX=/build/ ./install.sh tomlv                      9.7s
 => [proxy 3/4] RUN PREFIX=/build/ ./install.sh proxy                     14.6s
 => [dev 2/23] RUN useradd --create-home --gid docker unprivilegeduser     5.1s
 => [gometalinter 3/4] RUN PREFIX=/build/ ./install.sh gometalinter        9.4s
 => [dev 3/23] RUN ln -sfv /go/src/github.com/docker/docker/.bashrc ~/.ba  4.3s
 => [dev 4/23] RUN echo source /usr/share/bash-completion/bash_completion  2.5s
 => [dev 5/23] RUN ln -s /usr/local/completion/bash/docker /etc/bash_comp  2.1s
```

New docker build BuildKit plain output:

```
$ docker build --progress=plain .

#1 [internal] load .dockerignore
#1       digest:
sha256:d0b5f1b2d994bfdacee98198b07119b61cf2442e548a41cf4cd6d0471a627414
#1         name: "[internal] load .dockerignore"
#1      started: 2018-08-31 19:07:09.246319297 +0000 UTC
#1    completed: 2018-08-31 19:07:09.246386115 +0000 UTC
#1     duration: 66.818µs
#1      started: 2018-08-31 19:07:09.246547272 +0000 UTC
#1    completed: 2018-08-31 19:07:09.260979324 +0000 UTC
#1     duration: 14.432052ms
#1 transferring context: 142B done


#2 [internal] load Dockerfile
#2       digest:
sha256:2f10ef7338b6eebaf1b072752d0d936c3d38c4383476a3985824ff70398569fa
#2         name: "[internal] load Dockerfile"
#2      started: 2018-08-31 19:07:09.246331352 +0000 UTC
#2    completed: 2018-08-31 19:07:09.246386021 +0000 UTC
#2     duration: 54.669µs
#2      started: 2018-08-31 19:07:09.246720773 +0000 UTC
```

```
#2    completed: 2018-08-31 19:07:09.270231987 +0000 UTC
#2     duration: 23.511214ms
#2 transferring dockerfile: 9.26kB done
```

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:1.0-experimental
```

New 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.

For example, with a secret piece of information stored in a text file:

```
$ echo 'WARMACHINEROX' > mysecret.txt
```

And with a Dockerfile that specifies use of a BuildKit frontend `docker/dockerfile:1.0-experimental`, the secret can be accessed.

For example:

```
# syntax = docker/dockerfile:1.0-experimental
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
```

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:

```
$ docker build --no-cache --progress=plain --secret id=mysecret,src=mysecret.txt .
...
#8 [2/3] RUN --mount=type=secret,id=mysecret cat /run/secrets/mysecret
#8       digest:
sha256:5d8cbaeb66183993700828632bfbde246cae8feded11aad40e524f54ce7438d6
#8         name: "[2/3] RUN --mount=type=secret,id=mysecret cat
/run/secrets/mysecret"
#8      started: 2018-08-31 21:03:30.703550864 +0000 UTC
#8 1.081 WARMACHINEROX
#8    completed: 2018-08-31 21:03:32.051053831 +0000 UTC
#8     duration: 1.347502967s


#9 [3/3] RUN --mount=type=secret,id=mysecret,dst=/foobar cat /foobar
#9       digest:
sha256:6c7ebda4599ec6acb40358017e51ccb4c5471dc434573b9b7188143757459efa
#9         name: "[3/3] RUN --mount=type=secret,id=mysecret,dst=/foobar cat
/foobar"
#9      started: 2018-08-31 21:03:32.052880985 +0000 UTC
#9 1.216 WARMACHINEROX
#9    completed: 2018-08-31 21:03:33.523282118 +0000 UTC
#9     duration: 1.470401133s
...
```

Using SSH to access private data in builds

> **Acknowledgment**: Please see Build secrets and SSH forwarding in Docker 18.09 for more information and examples.

The `docker build` has a `--ssh` option to allow the Docker Engine to forward SSH agent connections. For more information on SSH agent, see the OpenSSH man page.

Only the commands in the `Dockerfile` that have explicitly requested the SSH access by defining `type=ssh` mount have access to SSH agent connections. The other commands have no knowledge of any SSH agent being available.

To request SSH access for a `RUN` command in the `Dockerfile`, define a mount with type `ssh`. This will set up the `SSH_AUTH_SOCK` environment variable to make programs relying on SSH automatically use that socket.

Here is an example Dockerfile using SSH in the container:

```
# syntax=docker/dockerfile:experimental
FROM alpine

# Install ssh client and git
RUN apk add --no-cache openssh-client git

# Download public key for github.com
RUN mkdir -p -m 0600 ~/.ssh && ssh-keyscan github.com >> ~/.ssh/known_hosts
```

```
# Clone private repository
RUN --mount=type=ssh git clone git@github.com:myorg/myproject.git myproject
```

Once the `Dockerfile` is created, use the `--ssh` option for connectivity with the SSH agent.

```
$ docker build --ssh default .
```

Troubleshooting : issues with private registries

x509: certificate signed by unknown authority

If you are fetching images from insecure registry (with self-signed certificates) and/or using such a registry as a mirror, you are facing a known issue in Docker 18.09 :

```
[+] Building 0.4s (3/3) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 169B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => ERROR resolve image config for docker.io/docker/dockerfile:experimental
------
 > resolve image config for docker.io/docker/dockerfile:experimental:
------
failed to do request: Head
https://repo.mycompany.com/v2/docker/dockerfile/manifests/experimental: x509:
certificate signed by unknown authority
```

Solution : secure your registry properly. You can get SSL certificates from Let's Encrypt for free. See /registry/deploying/

image not found when the private registry is running on Sonatype Nexus version < 3.15

If you are running a private registry using Sonatype Nexus version < 3.15, and receive an error similar to the following :

```
------
 > [internal] load metadata for docker.io/library/maven:3.5.3-alpine:
------
------
 > [1/4] FROM docker.io/library/maven:3.5.3-alpine:
------
rpc error: code = Unknown desc = docker.io/library/maven:3.5.3-alpine not found
```

you may be facing the bug below : NEXUS-12684

Solution is to upgrade your Nexus to version 3.15 or above.