Comparing Next-Generation Container Image Building Tools

Akihiro Suda (@_AkihiroSuda_ )
NTT Software Innovation Center
Who am I

• Software Engineer at NTT
• GitHub: @AkihiroSuda
• Twitter: @_AkihiroSuda_

• Docker  Moby core maintainer
  • In April 2017, Docker [ as a project ] transited into Moby
  • Now Docker [ as a product ] has been developed as one of downstreams of Moby

RHEL  :  Fedora
Who am I

• BuildKit initial maintainer
  • Next-generation `docker build`

• containerd maintainer
  • Industry-standard container runtime
  • Can be used as a Docker-replacement for Kubernetes

• Docker Tokyo Community Leader (meetup organizer)
  • https://dockerjp.connpass.com/
Agenda

• Problems of `docker build`

• New image builder tools

  - BuildKit
  - img
  - Buildah
  - umoci&orca
  - kaniko
  - Bazel
  - Source-to-Image
  - Metaparticle

• Comparison & Evaluation

• CBI: "Container Builder Interface"
Introduction to Dockerfile

• Shell-script-like language for building Docker container images

• Each of the lines is cached as a Copy-on-Write filesystem layer, e.g. overlayfs

FROM golang:1.10
COPY . /go/src/github.com/foo/bar
RUN go build -o /bar github.com/foo/bar

mount -t overlay \  
-o lowerdir=0,upperdir=1 ..

mount -t overlay \  
-o lowerdir=1,upperdir=2 ..
Introduction to Dockerfile

- Supports transferring files between stages, starting with Docker 17.05
  - Effectively reduces the size of the final image

FROM golang:1.10 AS foobar
COPY . /go/src/github.com/foo/bar
RUN go build -o /bar github.com/foo/bar

FROM alpine:3.7
COPY --from=foobar /bar /

copy "bar" to the final stage
Introduction to `docker build`

• Docker-integrated tool for building images using Dockerfile
  • Requires Docker daemon to be running

• Similar to `docker run`, but some features are intentionally removed for security reason
  • No volumes (`docker run -v`, `docker run --mount`)
  • No privileged mode (`docker run --privileged`)
Problem: inefficient caching

• Modifying a single line always invalidates the caches of the subsequent lines
  • N-th line is assumed to always depend on the (N-1)-th line

FROM debian
EXPOSE 80
RUN apt update && apt install -y HEAVY-PACKAGES

• A user needs to arrange the instructions carefully for efficient caching
Problem: no concurrency

• A multi-stage Dockerfile has DAG structure

FROM golang AS stage0
...
RUN go build -o /foo ...

FROM clang AS stage1
...
RUN clang -o /bar ...

FROM debian AS stage2
COPY --from=stage0 /foo /usr/local/bin/foo
COPY --from=stage1 /bar /usr/local/bin/bar

Directed Acyclic Graph has concurrency
Problem: no concurrency

• A multi-stage Dockerfile has DAG structure

FROM golang AS stage0
...
RUN go build -o /foo ...

FROM clang AS stage1
...
RUN clang -o /bar ...

FROM debian AS stage2
COPY --from=stage0 /foo /usr/local/bin/foo
COPY --from=stage1 /bar /usr/local/bin/bar

Actual `docker build` implementation (Sequential)
Problem: inaccessible to private assets

• No safe way to access private assets (e.g. Git repos, S3) from build containers

• Copying credentials using `COPY` can leak the credential accidentally
  • Needs to be carefully used with either multi-stage or `--squash`

```
FROM ...
COPY id_rsa ~/.ssh
RUN git clone ssh://...
RUN rm -f ~/.ssh/id_rsa
```

The key still remains in the layer!

• Env vars are vulnerable to accidents as well
Other problems

• Cannot be executed without root privileges
  • Important for building images on Kubernetes

• Cannot preserve compiler caches due to lack of volumes

• Unreproducible builds
  • Non-deterministic command executions
  • Left-pad issue

• Dockerfile can be too complex and hard to maintain
Solutions

BuildKit  img  Buildah  umoci & orca

kaniko  Bazel  Source-to-Image  Metaparticle

• And more!
  • FTL, Smith, Ansible Container...

• Some of them still use Dockerfile, others not

• No "silver bullet" solution
BuildKit: next-generation `docker build`

- Uses DAG-style low-level intermediate language called LLB
  - Accurate dependency analysis and cache invalidation
  - Vertices can be executed in parallel

- LLB can be compiled from Dockerfile
  - And also from 3rd party languages

Dockerfile ➔ Compile ➔ LLB DAG

3 vertices can be executed in parallel

- `docker-image://alpine`
- `git://foo/bar`
- `docker-image://gcc`
- `Run("make")`
- `Run("apk add ..")`

https://github.com/moby/buildkit
BuildKit: next-generation `docker build`

• DAG structure of LLB can be described using multi-stage Dockerfile

FROM golang AS stage0
...
RUN go build -o /foo ...

FROM clang AS stage1
...
RUN clang -o /bar ...

FROM debian AS stage2
COPY --from=stage0 /foo /usr/local/bin/foo
COPY --from=stage1 /bar /usr/local/bin/bar
BuildKit: next-generation `docker build`  

Can be also used for building non-container artifacts

Dependency graph of a go binary 😄
awesome buildkit demo highlighting nested invocation at #MobySummit
BuildKit: next-generation `docker build`

- Distributed mode is also on plan (#224, #231)
  - A worker tells the master its loadavg and LLB DAG vertex cache info
  - The master choose the worker for each of the LLB DAG vertices using the info from the workers

"I can reproduce cache for vertex sha256:deadbeef!"
Experimental support for rootless mode

- Runs everything including BuildKit itself as an unprivileged user, using `user_namespaces(7)`
  - Protect the system from potential bugs of BuildKit/containerd/runc.
  - Also useful for HPC users
  - Requires `newuidmap(1)` and `newgidmap(1)` with SUID bit for `apt`

- No patch for runc is needed since June 2018

- Don't confuse this with `dockerd --userns-remap`
  - `dockerd --userns-remap` still requires `dockerd` itself to be executed as the root
BuildKit: next-generation `docker build`

• Rootless BuildKit can be executed inside Docker and Kubernetes
  • But requires `--privileged` for let `RUN` containers mount `*/proc`
    • Will be fixed soon via moby/moby#36644 and kubernetes/kubernetes#64283
  • Still safe because BuildKit works as an unprivileged user

RootlessKit: shim for setting up user NS and mount NS
https://github.com/AkihiroSuda/rootlesskit
BuildKit: next-generation `docker build`

- Plan to support "privileged" build as well
  - likely to use libentitlement (#238)
  - e.g. `buildctl build --entitlements=security.unconfined` for privileged build
  - potential use-cases: GPU, FUSE, ...
BuildKit: next-generation `docker build`

- Supports non-standard Dockerfile "syntax", e.g. `RUN --mount`

  ```
  # syntax = tonistiigi/dockerfile:runmount20180610
  ...
  RUN --mount=target=/root/.cache,type=cache go build
  ```

  Cache mount can be useful for compilers (e.g. Go) and package managers (e.g. apt)

- `RUN --mount` will also support SSH agent socket file and secret files (#262)
BuildKit: next-generation `docker build`

• Benchmark result (from Tonis's slide: [https://t.co/aUKqQCVmXa](https://t.co/aUKqQCVmXa))
BuildKit: next-generation `docker build`

- Will be integrated to Moby & Docker 18.06 ([moby/moby#37151](moby/moby#37151))
  - No change on the `docker build` command line but you need to set `DOCKER_BUILDKIT=1`
  - Will be released by the end of this month

- Also adopted by OpenFaaS Cloud
  - [https://github.com/openfaas/openfaas-cloud](https://github.com/openfaas/openfaas-cloud)
  - "GitOps for your functions with native GitHub integrations"
Developed under Moby's open governance

- But Dockerfile-to-LLB compiler is planned to be moved to Docker, Inc.'s repo (#425)
  - Dockerfile specification is maintained by Docker, Inc.
- LLB allows implementing non-Dockerfile languages
  - Any idea for new language?
img: daemonless BuildKit

• Created by Jessie Frazelle (Microsoft)

• Uses BuildKit as a library but daemonless and has Docker-like CLI
  • Currently no support for running multiple `img` instances with the same cache directory (#92)

• Rootless mode by default

  $ img build -t example.com/foo .
  $ img push example.com/foo
  $ img save example.com/foo | docker load

https://github.com/genuinetools/img
Buildah: Red Hat's daemonless `docker build`

- Created by Red Hat
  - Officially included in RHEL since RHEL 7.5

- Supports Dockerfile, but `buildah run` and `buildah commit` are supported as well
  - as in `docker run` and `docker commit`, without Dockerfile

- Daemonless
  - Can be used as a backend of `podman build`
    - Podman: Red Hat's daemonless and swarmless Docker-like tool

https://github.com/projectatomic/buildah
Buildah: Red Hat's daemonless `docker build`

• Supports secret volume
  • But configuration is globally scoped
    `~/etc/containers/mounts.conf`
    • e.g. `~/usr/share/rhel/secrets:/run/secrets` for allowing all Buildah containers to access RHEL subscriptions
    • Seems to have usability and security concern for other use-cases

• Rootless mode is planned (#386)
Buildah: Red Hat's daemonless `docker build`

- Cache for Dockerfile instructions is not supported but planned (#601)
  
- Parallelization is also planned (#633)
  - And distributed execution as well
Umoci & Orca: the first rootless and daemonless image builder

• Created by Aleksa Sarai (SUSE)

• **Umoci:** Umoci modifies Open Container images
  • Unpacks and repacks OCI Image Spec archives (tar+gz and JSON) into/from OCI Runtime Spec bundles (directories)

• "Pure"-Rootless and daemonless
  • Does not require setting up subuids/subgids (which require SUID binary) for unpacking archives that have multiple UIDs/GIDs
    • Uses `user.rootlesscontainers` xattr instead of `chown(2)`

https://github.com/openSUSE/umoci
https://github.com/cyphar/orca-build
Umoci & Orca: the first rootless and daemonless image builder

• Orca: Umoci-based image builder with support for Dockerfile
  • Can be used with runROOTLESS for images that require multiple UIDs/GIDs (typically Debian/Ubuntu apt)
    • [https://github.com/rootless-containers/runrootless](https://github.com/rootless-containers/runrootless)
    • Emulates several system calls using `ptrace(2)` and `user.rootlesscontainers` xattr values (which are set by Umoci)
      • No SUID binary is required (but slow)

  • Multi-stage Dockerfile and caching are not supported at the moment

  • Planned to be integrated into Umoci
    • [https://twitter.com/lordcyphar/status/987668301890207744](https://twitter.com/lordcyphar/status/987668301890207744)
kaniko: "containerless" rootless builder

- Created by Google

- Kaniko itself needs to be executed in a container, but does not require `--privileged`
  - Execute `RUN` instructions within Kaniko's rootfs and namespaces
    - i.e. `RUN` instructions are executed without creating containers
  - Excludes kaniko itself's binary and configuration files on packing the rootfs archives
  - Seems inappropriate for malicious Dockerfiles due to lack of isolation ( #106)

https://github.com/GoogleCloudPlatform/kaniko
Non-Dockerfile based tools

• Bazel: Google's generic build system
  • Not specific to containers
  • `rules_docker` can build Docker images, but equivalent of `RUN` instruction is intentionally omitted due to poor reproducibility

```java
# https://github.com/bazelbuild/rules_docker#container_image
container_image(
    name = "app",
    base = "@java_base//image",
    files = ["//java/com/example/app:Hello_deploy.jar"],
    cmd = ["Hello_deploy.jar"]
)
```
Non-Dockerfile based tools

• **Source-to-Image: Red Hat OpenShift's build system**
  • Application developers don't need to write any file for building images
  • S2I base images contain scripts for building applications in the language-specific way
    • e.g. `centos/python-35-centos7` for Python 3.5
  • Previous versions depended on Docker, but recent version can produce Dockerfiles that can be built by other tools
Non-Dockerfile based tools

• Metaparticle: library for cloud-native apps on Kubernetes
  • Supports .NET, Go, Java, JS, Python, Ruby, Rust
  
  ```python
  from metaparticle import Containerize
  @Containerize(package={'repo': 'foo/bar', ...})
  def main():
      ...
  ```

• Hard to change the target repository without editing source codes
  • Or implementing a new library on top of Metaparticle
• Also provides service-related features
  • e.g. sharding HTTP requests based on URL
Non-Dockerfile based tools

• FTL
  • Similar to S2I but only for Node.js, Python, and PHP

• Smith
  • Supports Oracle's "Microcontainer Manifest"

• Ansible Container
  • Supports Ansible Playbook
  • README says "no longer under active development"
### Comparison across Dockerfile-based tools

<table>
<thead>
<tr>
<th></th>
<th>Docker</th>
<th>BuildKit</th>
<th>img</th>
<th>Buildah</th>
<th>Orca</th>
<th>kaniko</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction cache</td>
<td>Limited</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallelization</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>Planned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributed execution</td>
<td>Planned</td>
<td></td>
<td></td>
<td>Planned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Daemonless</td>
<td>As a library</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Rootless</td>
<td>✔ (1)</td>
<td>✔ (1)</td>
<td></td>
<td>Planned</td>
<td>✔ (2)</td>
<td>✔ (3)</td>
</tr>
</tbody>
</table>

1. Requires SUID binary for apt
## Comparison across Dockerfile-based tools

<table>
<thead>
<tr>
<th></th>
<th>Docker</th>
<th>BuildKit</th>
<th>img</th>
<th>Buildah</th>
<th>Orca</th>
<th>kaniko</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction cache</td>
<td>Limited</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallelization</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributed execution</td>
<td>Planned</td>
<td></td>
<td></td>
<td>Planned</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Daemonless</td>
<td>As a library</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Rootless</td>
<td>✔️ 1</td>
<td>✔️ 1</td>
<td>Planned</td>
<td>✔️ 2</td>
<td>✔️ 3</td>
<td></td>
</tr>
</tbody>
</table>

2 No SUID required but slow
## Comparison across Dockerfile-based tools

<table>
<thead>
<tr>
<th>Feature</th>
<th>Docker</th>
<th>BuildKit</th>
<th>img</th>
<th>Buildah</th>
<th>Orca</th>
<th>kaniko</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction cache</td>
<td>Limited</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallelization</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributed execution</td>
<td>Planned</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Daemonless</td>
<td>As a library</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Rootless</td>
<td>✔️</td>
<td>✔️ ✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*3 Executable in containers without `--privileged` but still has security concern*
Benchmark

Always without cache

Some builders use cache

Build #1
Put a dummy file
Build #2
Prune the state
Build #1
Put a dummy file
Build #2
Prune the state
...

Average time of Build #1 (5 times)

Simulates trivial code change

Average time of Build #2 (5 times)
Benchmark

• Benchmark script is available
  • [https://github.com/AkihiroSuda/buildbench](https://github.com/AkihiroSuda/buildbench)
  • Supported tools: Docker, Buildkit, img, Buildah, Kaniko
  • Everything is containerized
  • Builders (except Kaniko) are configured to use overlayfs

• Tested on Travis CI (June 19, 2018)
  • Logs (contains version info and raw data): [https://travis-ci.org/AkihiroSuda/buildbench/builds/393967682](https://travis-ci.org/AkihiroSuda/buildbench/builds/393967682)
    • See also [https://github.com/AkihiroSuda/buildbench/issues/5](https://github.com/AkihiroSuda/buildbench/issues/5)
  • 2 bursted vCPUs, 7.5GB RAM
Benchmark: examples/ex01

FROM alpine AS buildc
RUN apk add --no-cache build-base
RUN echo ... > hello.c
COPY . /foo
RUN gcc -o /a.out /hello.c

FROM alpine AS buildgo
RUN apk add --no-cache build-base
RUN apk add --no-cache go
RUN echo ... > hello.go
RUN go build -o /a.out /hello.go

FROM alpine
COPY --from=buildc /a.out /hello1
COPY --from=buildgo /a.out /hello2

Only the cache for the next line SHOULD be invalidated on modification of the build ctx

`apk add build-base` SHOULD NOT be executed twice
Benchmark result: examples/ex01

- **Docker**:
  - #1: 15.6s
  - #2: 1.2s

- **BuildKit**:
  - #1: 7.9s
  - #2: 1.1s

- **img**:
  - #1: 10.0s
  - #2: 1.9s

- **Buildah**:
  - #1: 15.3s
  - #2: 13.5s

- **Kaniko**:
  - #1: 13.2s
  - #2: 13.1s
Another benchmark: moby/moby

- Dockerfile used for the development of Moby
- Good example of complex DAG
  - 13 stages can be executed in parallel at maximum
  - Buildah and Kaniko don't support this DAG at the moment
    - `FROM base` results in attempt to pull `docker.io/library/base`

```
FROM base
```

```golang:1.10.3
```
Benchmark result: moby/moby

- **Docker**
  - #1: 351.8s
  - #2: 6.4s

- **BuildKit**
  - #1: 278.8s
  - #2: 1.9s

- **img**
  - #1: 447.1s
  - #2: 7.6s
So.. which one is the best?

• My recommendation is BuildKit, but it is not the "silver bullet"
  • disclosure: I'm a maintainer of BuildKit

• Other tools are attractive as well
  • Language-specific builders, e.g. S2I
  • SUID-less rootless mode, e.g. Orca and Kaniko
  • Enterprise support, e.g. Buildah

• Can we define the common interface for all of them?
CBI: Container Builder Interface for Kubernetes

- [https://github.com/containerbuilding/cbi](https://github.com/containerbuilding/cbi)
- Defines "BuildJob" as a Kubernetes CRD
- Supports several backends

Note: not an official CNCF/Kubernetes project
CBI: Container Builder Interface for Kubernetes

The CBI controller converts "BuildJob" CRD objects into Kubernetes batch/v1 Job objects.

Most plugins accept Dockerfile, but non-Dockerfile plugins are also supported. e.g. Source-to-Image.

```yaml
apiVersion: cbi.containerbuilding.github.io/v1alpha1
kind: BuildJob
metadata:
  name: ex0
spec:
  registry:
    target: example.com/foo/bar
    push: true
  language:
    dockerfile: {}
  context:
    git:
      url: git://github.com/foo/bar
  pluginSelector: plugin.name=buildkit
```
CBI: Container Builder Interface for Kubernetes

apiVersion: cbi.containerbuilding.github.io/v1alpha1
kind: BuildJob
metadata:
  name: ex0
spec:
  registry:
    target: example.com/foo/bar
    push: true
  language:
    dockerfile: {}
  context:
    git:
      url: git://github.com/foo/bar
  pluginSelector: plugin.name=buildkit

Registry and Git credentials can be provided as Kubernetes secret objects

Also supports ConfigMap, HTTP, S3, SFTP, and even Dropbox.. (using Rclone)
CBI: Container Builder Interface for Kubernetes

• Supported plugins:
  • Docker
  • BuildKit
  • img
  • Buildah
  • kaniko
  • OpenShift Source-to-Image
  • Google Cloud Container Builder
    • Managed service for `docker build`

• New plugin can be also added easily as a Kubernetes service
CBI: Container Builder Interface for Kubernetes

• POC for Skaffold integration is available (GoogleContainerTools/skaffold#596)

apiVersion: skaffold/v1alpha2
kind: Config
build:
  artifacts:
  - imageName: example.com/foo/bar
deploy:
  kubectl:
    manifests:
    - k8s-pod.yaml
profiles:
  - name: cbi
    build:
      cbi: {}

Deploy a Kubernetes pod using the image

By default the local Docker is used, but can be easily switched to CBI (`skaffold dev -p cbi`)
Conclusion

• **My recommendation is BuildKit** (disclosure: I'm a maintainer)
  • Will be integrated to Docker 18.06 experimentally (planned to be released by the end of this month)

• **But other tools are promising as well**

• **Now is the time for standardization**
  • [https://github.com/containerbuilding/cbi](https://github.com/containerbuilding/cbi)