Container build at scale using BuildKit and BuildX

About Me

Lead AI Ops SRE @ Cisco

17 years in the IT (Ex-SUSE/Rancher/Nutanix/Oracle/HPE/IBM)

Blog: https://ansilh.com

Agenda

- Introduction to Docker build
- BuildKit vs. Legacy Docker Build
- BuildKit LLB
- BuildKit Drivers (Docker, Kubernetes, Remote)
- BuildKit Security
- Multi-Platform Builds
- Bake HCL Workflows
- Cache
- Demos

Docker build - Legacy

- Sequential execution
- Each instruction generate a cached layer
- Caching relies on heuristics (e.g., timestamps, Dockerfile instruction text, or file metadata)
- Secrets passed via ARG have security risks
- Multi-platform builds needs workarounds

Docker build - BuildKit

- Graph based execution using Low Level Build (LLB)
- Executes independent steps in parallel
- Supports remote caching and inline caching
- Secrets can be securely mounted using `--mount=type=secret`
- Rootless mode allows builds without root privilege
- Native support for different architectural builds via QEMU or remote builders

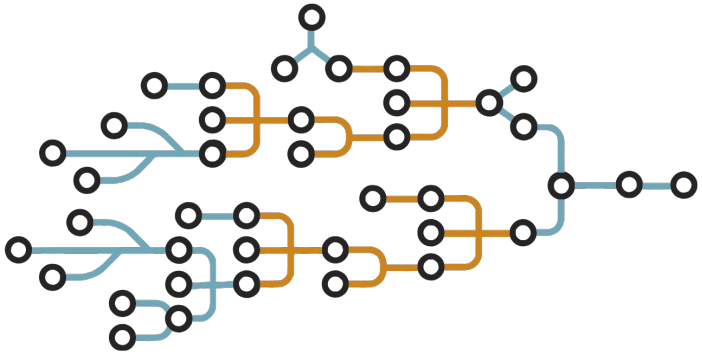
Proposal: https://gist.github.com/tonistiigi/059fc72c4630f066d94dafb5e0e70dc6

BuildKit vs. Legacy Docker Build

Legacy Build	**BuildKit**
Sequential stages	Parallel stage execution
Basic local cache	Multi-layer caching
Limited platform support	Native multi-platform
No built-in secrets	`--mount=type=secret`
No frontend customization	Extensible LLB (Low-Level Build)

BuildKit LLB

- Low Level Build format is an intermediate binary format
- LLB defines a content-addressable dependency graph



Frontend

The “shabang” for build definition

https://docs.docker.com/build/buildkit/frontend/

Flexibility to get features and bug fixes without docker engine updates

```
# syntax=[remote image reference]

# syntax=docker/dockerfile:1
# syntax=docker.io/docker/dockerfile:1
# syntax=example.com/user/repo:tag@sha256:abcdef...
```

BuildX

- A CLI that adds additional features to docker build
- High-level build constructs like “bake”
- Installation - refer https://github.com/docker/buildx

BuildKit Drivers

1. Docker Driver (default, uses Docker's daemon).
2. Docker Container Driver
3. Kubernetes Driver (scale builds across pods).
    - Use case: CI/CD in k8s clusters.
4. Remote Driver (connect to external BuildKitd).
    - Use case: Dedicated BuildKit servers.

Docker driver

The default driver

Built directly into the docker engine

Images built will be loaded directly to the image store

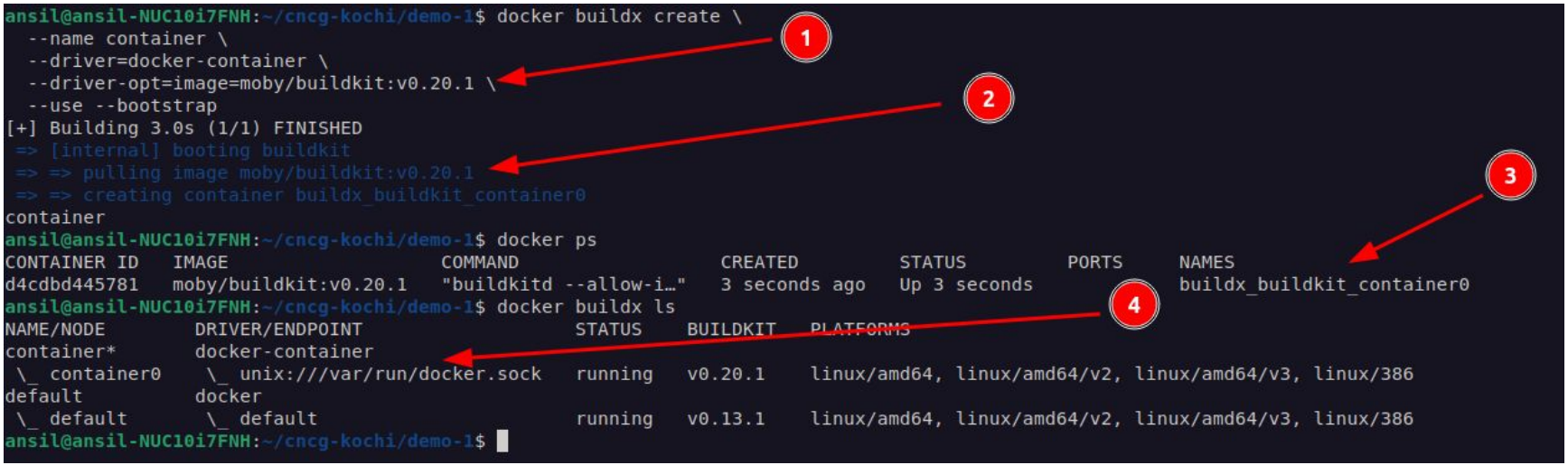
```
ansil@ansil-NUC10i7FNH:~/cncg-kochi/demo-1$ docker buildx ls
NAME/NODE        DRIVER/ENDPOINT                  STATUS    BUILDKIT   PLATFORMS
container*       docker-container
 \_ container0    \_ unix:///var/run/docker.sock   running   v0.20.1    linux/amd64, linux/amd64/v2, linux/amd64/v3, linux/386
default          docker
 \_ default       \_ default                       running   v0.13.1    linux/amd64, linux/amd64/v2, linux/amd64/v3, linux/386
ansil@ansil-NUC10i7FNH:~/cncg-kochi/demo-1$
```

Docker Container Driver

- Specify custom BuildKit versions to use.
- Build multi-arch images
- Advanced options for cache import and export

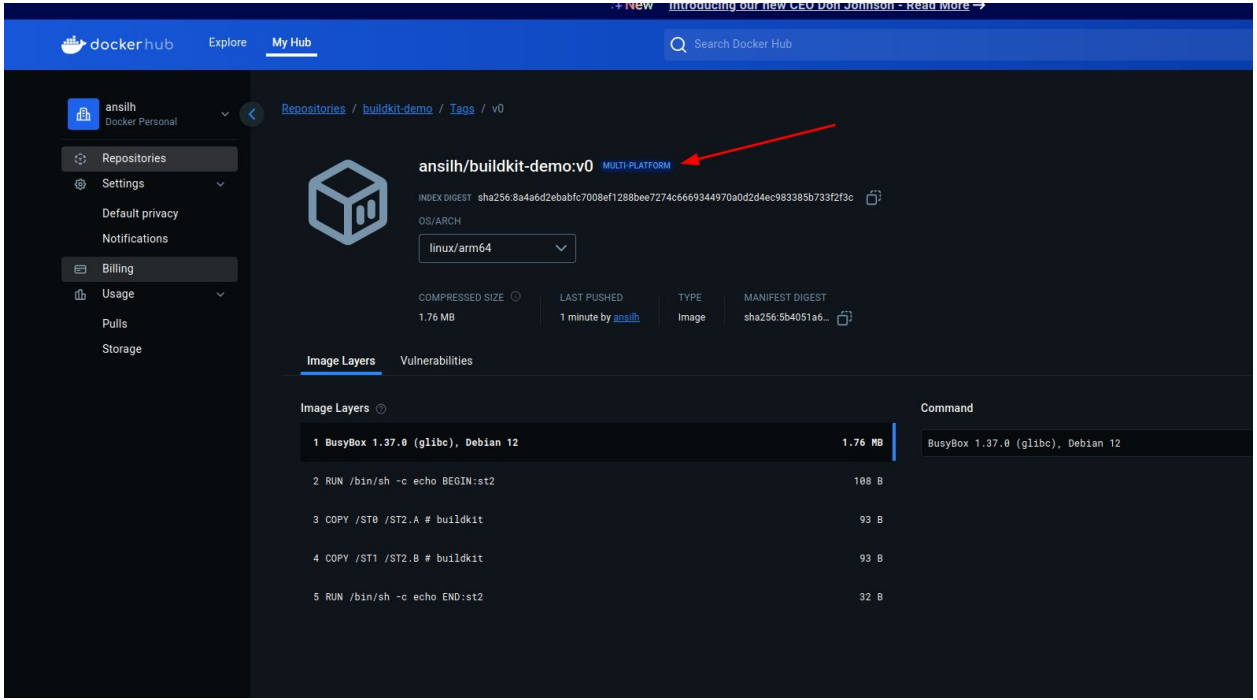
```
docker buildx create \
  --name container \
  --driver=docker-container \
  --driver-opt=image=moby/buildkit:v0.20.1 \
  --use --bootstrap
```

Docker Container Driver

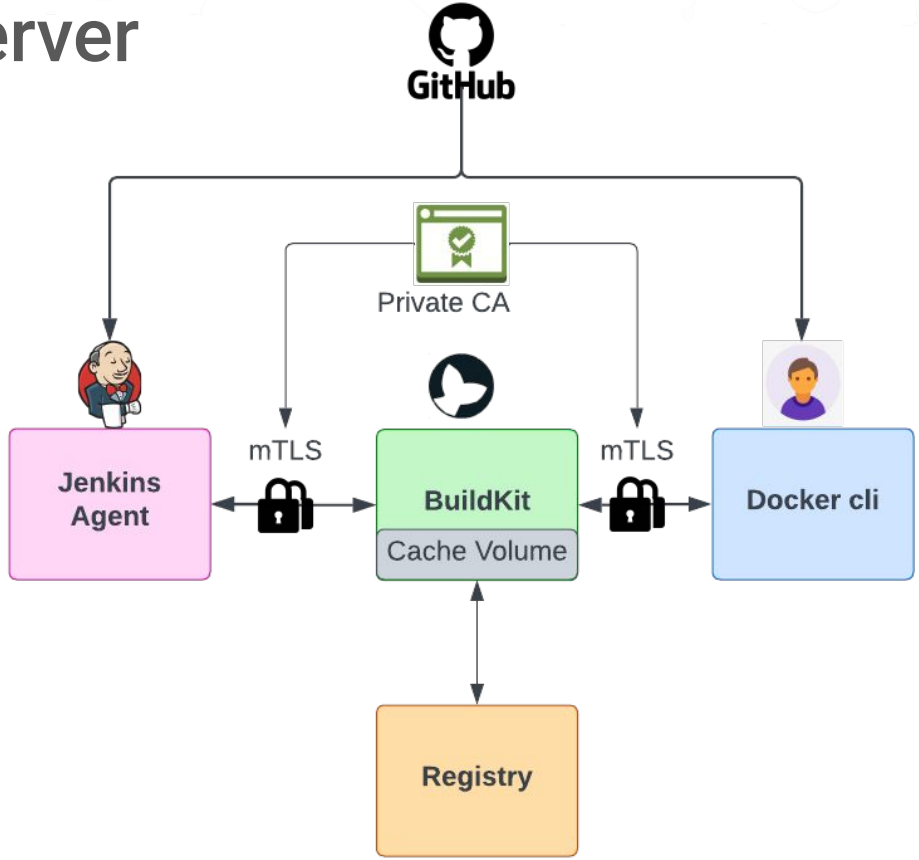


time docker buildx build --platform=linux/amd64,linux/arm64 --builder=container -t ansilh/buildkit-demo:v0 --push --progress=plain .

Docker container driver - multi-platform build



Buildkit Server



BuildKit Security

Secrets Management:

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

Rootless Mode: Run builds without root privileges.

https://github.com/rootless-containers/rootlesskit/

Bake

kubernetes

Buildx Bake

Bake is a command built into the Buildx CLI,

Bake is an abstraction for the docker build command that lets you more easily manage your build configuration (CLI flags, environment variables, etc.) in a consistent way for everyone on your team.

You can write Bake files in HCL, YAML (Docker Compose files), or JSON. In general, HCL is the most expressive and flexible format

Example

```
# docker build -f Dockerfile -t myapp:latest .
```

docker-bake.hcl

```
target "myapp" {
  context = "."
  dockerfile = "Dockerfile"
  tags = ["myapp:latest"]
}
# docker buildx bake myapp
```

Targets

```
# Custom target
target "webapp" {
  dockerfile = "webapp.Dockerfile"
  tags = ["docker.io/ansilh/webapp:latest"]
  context = "https://github.com/ansilh/webapp"
}
# Default target
target "default" {
  dockerfile = "webapp.Dockerfile"
  tags = ["docker.io/ansilh/webapp:latest"]
  context = "https://github.com/username/webapp"
}

# docker buildx bake --print
```

Inheritance

```
target "_common" {
  args = {
    GO_VERSION = "1.23"
    BUILDKIT_CONTEXT_KEEP_GIT_DIR = 1
  }
}

target "app-dev" {
  inherits = ["_common"]

  args = {
    BUILDKIT_CONTEXT_KEEP_GIT_DIR = 0
  }

  tags = ["docker.io/ansilh/myapp:dev"]
  labels = {
        "org.opencontainers.image.source" = "https://github.com/ansilh/myapp"
        "org.opencontainers.image.author" = "ansilh@example.com"
  }
}

target "app-release" {
  inherits = ["app-dev", "_common"]
  tags = ["docker.io/ansilh/myapp:latest"]
  platforms = ["linux/amd64", "linux/arm64"]
}
```

Variables

```
group "default" {
  targets = [ "webapp" ]
}

variable "TAG" {
  default = "latest"
}

target "webapp" {
  context = "."
  dockerfile = "Dockerfile"
  tags = ["docker.io/ansilh/webapp: ${TAG}"]
}
```

Functions

Bake ships with built-in support for the go-cty

https://github.com/zclconf/go-cty/blob/main/README.md

```
variable "TAG" {
  default = "latest"
}

group "default" {
  targets = ["webapp"]
}

target "webapp" {
  args = {
      buildno = "${add(123, 1)}"
  }
}
```

https://github.com/zclconf/go-cty/blob/main/cty/function/stdlib/number.go#L30

User defined functions

```
function "increment" {
  params = [number]
  result = number + 1
}

group "default" {
  targets = ["webapp"]
}

target "webapp" {
  args = {
    buildno = "${increment(123)}"
  }
}
```

Cache

Cache

Inline

Local

Registry

Amazone S3 (Experimental)

GitHub Action (Experimental)

Azure Blobe Storage (Experimental)

Inline

Cache will be part of the image

Not scalable

Local

Cache inside a directory in your filesystem using OCI format

Good choice for testing

Registry

An extension of inline cache, but the cache will be stored as a separate image

Allows for separating the cache and resulting image artifacts so that you can distribute your final image without the cache inside.

It can efficiently cache multi-stage builds in max mode, instead of only the final stage.

It works with other exporters for more flexibility, instead of only the image exporter.

Exporters

Image and registry exporters

The image exporter outputs the build result into a container image format.

The registry exporter is identical, but it automatically pushes the result by setting push=true.

Local and tar exporters

The local and tar exporters output the root filesystem of the build result into a local directory.

They're useful for producing artifacts that aren't container images.

OCI and Docker exporters

The oci exporter outputs the build result into an OCI image layout tarball.

The docker exporter behaves the same way, except it exports a Docker image layout instead.

DEMO