Docker BuildKit TOML Configuration - Word by Word Breakdown

🏗️ Think of BuildKit as a Construction Company

BuildKit is like a **construction company** that builds Docker images. The `buildkitd.toml` file is like the **company's operations manual** - it tells the construction workers (buildkitd daemon) how to operate, where to store materials, how to handle security, and what rules to follow.

📍 File Location Structure

```
/etc/buildkit/buildkitd.toml      ← Rootful mode (like admin office)
~/.config/buildkit/buildkitd.toml ← Rootless mode (like home office)
```

Real-World Analogy: Like having a company handbook either at corporate headquarters (rootful) or at your home office branch (rootless).

🔧 Global Settings Section

Debug and Trace Settings

```toml
debug = true
trace = true
```

ASCII Visualization:

```
Normal Mode:  [BuildKit] → [Build] → [Success]
Debug Mode:   [BuildKit] → [🔍Debug Info] → [Build] → [📋Detailed Log] → [Success]
Trace Mode:   [BuildKit] → [🔍🔍ULTRA Debug] → [Every Step Logged] → [Success]
```

Real-World Analogy:

- **Normal**: Construction supervisor gives basic updates
- **Debug**: Supervisor explains each decision and problem
- **Trace**: Supervisor describes every single nail hammered and screw turned

Root Directory

```toml
root = "/var/lib/buildkit"
```

ASCII Visualization:

```
BuildKit Workspace:
/var/lib/buildkit/
├── cache/        ← Material storage
├── metadata/     ← Project blueprints
├── snapshots/    ← Work-in-progress photos
└── tmp/          ← Temporary workspace
```

Real-World Analogy: This is BuildKit's main warehouse where all materials, blueprints, and work-in-progress projects are stored.

Insecure Entitlements

```toml
insecure-entitlements = [ "network.host", "security.insecure" ]
```

ASCII Visualization:

```
🔒 Normal Build:   [Container] ──| Security Gate |── [Host Network]
🚫 Insecure Build: [Container] ──────────────────→ [Direct Host Access]
```

Real-World Analogy: Like giving construction workers master keys to the entire building instead of just their work area - powerful but risky.

📝 Logging Configuration

```toml
[log]
format = "text"
```

Real-World Analogy: Choosing between a **newspaper format** (text) vs **data spreadsheet format** (json) for your construction daily reports.

🌐 DNS Configuration

```toml
[dns]
nameservers=["1.1.1.1","8.8.8.8"]
options=["edns0"]
searchDomains=["example.com"]
```

ASCII Visualization:

```
BuildKit needs to find docker.io:
BuildKit → [DNS: 1.1.1.1] → "Where is docker.io?" → [IP: 54.230.75.15] → Download
```

Real-World Analogy:

- **Nameservers**: Like having multiple phone books (Cloudflare, Google)
- **searchDomains**: If you say "go to 'registry'", automatically try "registry.example.com"
- **options**: Special phone book features (like enhanced directory services)

📡 GRPC Communication Settings

```toml
[grpc]
address = [ "tcp://0.0.0.0:1234" ]
debugAddress = "0.0.0.0:6060"
uid = 0
gid = 0
```

ASCII Visualization:

```
Client Commands:
[Docker Build] ──tcp://0.0.0.0:1234──→ [BuildKit Daemon]
                          ↓
Debug/Monitoring:              [Debug Port 6060]
[Profiler Tools] ←──────────────────────────────┘
```

Real-World Analogy:

- **Main address**: Like the main phone number for your construction company
- **debugAddress**: Like a special hotline for supervisors to check on work progress
- **uid/gid**: Like specifying which employee ID has authority to answer calls

TLS Security

```toml
[grpc.tls]
cert = "/etc/buildkit/tls.crt"
key = "/etc/buildkit/tls.key"
ca = "/etc/buildkit/tlsca.crt"
```

ASCII Visualization:

```
Secure Communication:
[Client] ──🔐encrypted──→ [BuildKit]
      ←──🔐encrypted──←
```

Real-World Analogy: Like having encrypted walkie-talkies for construction site communication - only authorized people with the right codes can listen in.

📊 OpenTelemetry (OTEL)

```toml
[otel]
socketPath = "/run/buildkit/otel-grpc.sock"
```

Real-World Analogy: Like having a dedicated data analyst who sits in a specific office and collects performance metrics about your construction projects.

🔌 Container Device Interface (CDI)

```toml
[cdi]
disabled = true
specDirs = ["/etc/cdi", "/var/run/cdi", "/etc/buildkit/cdi"]
```

ASCII Visualization:

```
GPU/Special Hardware Access:
[Build Container] ──🚫 CDI Disabled──→ [No Special Hardware]
[Build Container] ──✅ CDI Enabled───→ [GPU, FPGAs, Special Devices]
```

Real-World Analogy: CDI is like a **specialized equipment rental system**. If disabled, workers only get basic tools. If enabled, they can request special equipment like cranes or specialized machinery.

📚 Build History

```toml
[history]
maxAge = 172800    # 48 hours in seconds
maxEntries = 50
```

ASCII Visualization:

```
Build History Log:
[Build #1] ─── 47h ago ─── ✅ Keep
[Build #2] ─── 49h ago ─── 🗑️ Delete (too old)
[Build #50] ──────────────── ✅ Keep (within limit)
[Build #51] ──────────────── 🗑️ Delete (over limit)
```

Real-World Analogy: Like keeping construction project records for 2 days or up to 50 projects maximum - old paperwork gets shredded automatically.

🏗️ Worker Configuration - OCI Worker

```toml
[worker.oci]
enabled = true
platforms = [ "linux/amd64", "linux/arm64" ]
snapshotter = "auto"
rootless = false
```

ASCII Visualization:

```
Multi-Platform Building:
        [BuildKit]
       /      \
  [AMD64 Builder]  [ARM64 Builder]
     |           |
  [Intel Images]  [ARM Images]
```

Real-World Analogy:

- **OCI Worker**: Like having a **specialized construction crew**
- **platforms**: Crew can build both **skyscrapers** (amd64) and **tiny houses** (arm64)
- **snapshotter**: Method for taking **progress photos** (overlayfs = layered photos, native = simple snapshots)
- **rootless**: Whether crew needs **manager permissions** or can work independently

Process Sandbox

```toml
noProcessSandbox = false
```

ASCII Visualization:

```
With Sandbox:    [Build Process] ──| Isolation Wall |── [Host System]
Without Sandbox: [Build Process] ──────────────────→ [Direct Host Access]
```

Real-World Analogy: Like deciding whether construction workers work in a **fenced construction site** (sandbox) or have **free access to the entire property**.

Garbage Collection Settings

```toml
gc = true
reservedSpace = "30%"
maxUsedSpace = "60%"
minFreeSpace = "20GB"
```

ASCII Visualization:

```
Disk Space Management (100GB total disk):
├── Reserved (30GB) │██████████│ ← Never touch this
├── Working (30GB) │██████████│ ← BuildKit can use this
└── Free (40GB)     │████████████████████│ ← Must keep at least 20GB free
```

Real-World Analogy:

- **reservedSpace**: Like keeping 30% of warehouse space always empty for emergencies
- **maxUsedSpace**: Never use more than 60% of total storage
- **minFreeSpace**: Always keep at least 20GB free, like keeping loading dock space clear

Performance Settings

```toml
max-parallelism = 4
cniPoolSize = 16
```

ASCII Visualization:

```
Parallel Work:
Project 1: [Worker A] ───building───→
Project 2: [Worker B] ───building───→
Project 3: [Worker C] ───building───→
Project 4: [Worker D] ───building───→
Project 5: [Waiting...] (max-parallelism = 4)

Network Pool:
[Net1][Net2][Net3]...[Net16] ← Pre-made network connections
```

Real-World Analogy:

- **max-parallelism**: Maximum number of **construction crews working simultaneously**
- **cniPoolSize**: Like having **16 pre-setup work sites** so crews don't waste time setting up/tearing down each time

Container Runtime

```toml
```

```
binary = ""
apparmor-profile = ""
```

Real-World Analogy:

- **binary**: Like choosing which **type of construction equipment** to use (bulldozer vs excavator)
- **apparmor-profile**: Like assigning **specific safety protocols** that workers must follow

---

🏷️ Worker Labels and GC Policies

```toml
[worker.oci.labels]
"foo" = "bar"

[[worker.oci.gcpolicy]]
reservedSpace = "512MB"
maxUsedSpace = "1GB"
minFreeSpace = "10GB"
keepDuration = "48h"
filters = [ "type==source.local", "type==exec.cachemount", "type==source.git.checkout"]
```

ASCII Visualization:

```
Garbage Collection Rules:
Rule 1: Local files   ── keep 48h ── then delete if space needed
Rule 2: Cache mounts  ── keep 48h ── then delete if space needed
Rule 3: Git checkouts ── keep 48h ── then delete if space needed

Space Management:
[512MB Protected] [488MB Working] [Rest Must Stay Free]
```

Real-World Analogy:

- **Labels**: Like putting **name tags** on different work crews
- **GC Policies**: Like having **different cleanup rules** - keep blueprints for 48 hours, but throw away construction debris immediately

---

🐳 Containerd Worker

```toml
[worker.containerd]
address = "/run/containerd/containerd.sock"
enabled = true
namespace = "buildkit"
defaultCgroupParent = "buildkit"
```

ASCII Visualization:

```
BuildKit ← → Containerd Connection:
[BuildKit] ——unix socket——→ [/run/containerd/containerd.sock] ←—— [Containerd]
```

Real-World Analogy:

- **Containerd worker**: Like having a **different construction company** (containerd) as a subcontractor
- **namespace**: Like working in a **specific section** of a larger construction site
- **defaultCgroupParent**: Like having all projects report to the **same project manager**

🗄️ Registry Configuration

```toml
[registry."docker.io"]
mirrors = ["yourmirror.local:5000", "core.harbor.domain/proxy.docker.io"]
http = true
insecure = true
ca=["/etc/config/myca.pem"]
```

ASCII Visualization:

```
Registry Mirror System:
Need ubuntu image:
[BuildKit] → [Mirror 1: yourmirror.local:5000] —— if available ——→ [Image]
        ↓ if not available
        → [Mirror 2: core.harbor.domain] —— if available ——→ [Image]
        ↓ if not available
        → [docker.io] ————————————————————————————————→ [Image]
```

Real-World Analogy:

- **Registry**: Like **hardware stores** where you buy construction materials
- **Mirrors**: Like having **local suppliers** who stock the same items as the main store
- **http/insecure**: Like choosing between **secure courier delivery** vs **regular mail**

🚪 Frontend Control

```toml
[frontend."dockerfile.v0"]
enabled = true

[frontend."gateway.v0"]
enabled = true
allowedRepositories = []
```

ASCII Visualization:

```
Build Frontend Types:
[Dockerfile] ──dockerfile.v0──→ [BuildKit] ──→ [Image]
[Custom Script] ──gateway.v0──→ [BuildKit] ──→ [Image]
```

Real-World Analogy:

- **Frontends**: Like different **types of blueprint formats** your construction company accepts
- **dockerfile.v0**: Standard architectural blueprints
- **gateway.v0**: Custom engineering plans (with security restrictions on who can submit them)

⚙️ System Settings

```toml
[system]
platformsCacheMaxAge = "1h"
```

Real-World Analogy: Like **refreshing your catalog** of available construction equipment types every hour - checking if new machinery models are available or if old ones are discontinued.

🎯 Key Takeaways

BuildKit TOML Configuration is like...

1. 📋 **Operations Manual**: Tells the construction company how to work
2. 🏢 **Office Setup**: Where to store files, how to communicate
3. 👷 **Crew Management**: How many workers, what tools they use
4. 🧹 **Cleanup Rules**: When to throw away old materials
5. 🏪 **Supplier Relationships**: Where to buy materials, security protocols
6. 📐 **Blueprint Standards**: What types of plans are accepted

The configuration balances **performance** (parallel work), **security** (isolation, certificates), **storage management** (garbage collection), and **flexibility** (multiple workers, registries) to create an efficient Docker image building system.