readme.md for @host.today/ht-docker-node

□□ ht-docker-node

> Production-ready Docker images for Node.js with NVM built in, multi-arch support, and modern runtimes (Bun, Deno). Every image ships with **tini** as PID 1 and full **NVM** integration — switch Node versions on the fly, no sourcing required.

Multi-arch • Alpine & Ubuntu • NVM built-in • Bun, Deno & pnpm • tini init • CI/CD optimized

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

□□ Quick Start

```
# Pull and run the full-featured Ubuntu image
docker pull code.foss.global/host.today/ht-docker-node:latest
docker run -it code.foss.global/host.today/ht-docker-node:latest

# Or go lean with Alpine (~200 MB vs ~900 MB)
```

```
docker pull code.foss.global/host.today/ht-docker-node:alpine-node
docker run -it code.foss.global/host.today/ht-docker-node:alpine-node
```

NVM is ready the moment you enter the container — no manual sourcing, no `.bashrc` hacks:

```
$ nvm install 22
$ nvm use 22
$ node -v   # v22.x.x 
```

□□ Available Images

Ubuntu-Based (Full-Featured)

Built on **Ubuntu 24.04**. Maximum compatibility, all build tools included, plus Chromium for Puppeteer/Playwright, and MongoDB 8.0.

Tag	Description	Key Contents
`:latest`	Kitchen-sink Node.js image	Node LTS + NVM + pnpm + Bun + Deno + Chromium + MongoDB 8.0
`:lts`	Alias of `:latest`	Same — explicit LTS naming for clarity
`:szci`	CI/CD workhorse	`:latest` + `@ship.zone/szci` preinstalled
`:fossglobal_preinstalled_<ver>`	Preloaded tooling image	`:szci` + tsrun, tstest, tapbundle, smartfile, and more

Alpine-Based (Lightweight & Multi-Arch) ⚡

40–75 % smaller than Ubuntu. Native performance on **both amd64 and arm64** (Apple Silicon, Graviton, Ampere).

Tag	Description	Size	Architectures
`:alpine-node`	Node.js LTS + NVM + pnpm	~200 MB	amd64, arm64
`:alpine-bun`	Node.js LTS + NVM + Bun	~150 MB	amd64, arm64
`:alpine-deno`	Node.js LTS + NVM + Deno	~180 MB	amd64, arm64

Tag	Description	Size	Architectures
`:alpine-szci`	Alpine Node + szci + build tools	~250 MB	amd64, arm64

> □□ Docker automatically pulls the right arch for your platform. Build on a Mac, deploy on an ARM server — same tag, native speed everywhere.

> **Note:** The Deno Alpine image uses `alpine:edge` to get the official musl-compiled Deno from the community repository.

What every image includes

Feature	Detail
tini	PID 1 init — proper signal forwarding & zombie reaping
NVM	v0.40.1 — works in `RUN`, `docker exec`, CI scripts, interactive shells
Node.js	LTS v24.13.0 (default, switchable)
docker-entrypoint.sh	Loads NVM at runtime so `docker run … bash -c "nvm use 22"` just works

□□ Key Features

□□ NVM — Zero-Config Node Version Management

NVM is pre-wired into every shell context. No manual sourcing required in any of these scenarios:

Dockerfile RUN commands (via the `bash-with-nvm` SHELL wrapper):

```
FROM code.foss.global/host.today/ht-docker-node:latest

# Works directly — no sourcing needed!
```

```
RUN nvm install 22 && nvm use 22 && npm ci
RUN nvm alias default 22          # persists for later RUN steps
```

CI/CD scripts (via `BASH_ENV=/etc/bash.bashrc`):

```
# Gitea / GitLab CI
test:
  image: code.foss.global/host.today/ht-docker-node:latest
  script:
    - nvm install 22 && nvm use 22
    - pnpm ci && pnpm test
```

Interactive shells and `docker exec`:

```
docker exec -it mycontainer bash
$ nvm ls          # lists installed versions
$ nvm install 20  # installs Node 20
$ nvm use 20      # switches immediately
```

> ⚠ **Note on version persistence across RUN steps:** Each Dockerfile `RUN` starts a new shell. Use `nvm alias default <version>` to persist your choice, or chain commands in a single `RUN`.

□□ tini — Proper Init for Containers

All images use tini as PID 1:

```
tini → docker-entrypoint.sh → your command
```

This means:

- □ Signals (SIGTERM, SIGINT) are forwarded correctly to your app
- □ Zombie processes are reaped automatically
- □ Clean container shutdown — no orphaned processes

□□ Chromium (Ubuntu `:latest` only)

Puppeteer and Playwright work out of the box:

```
const browser = await puppeteer.launch(); // uses /usr/bin/chromium-browser
```

Environment variables `PUPPETEER_EXECUTABLE_PATH` and `CHROME_BIN` are pre-set. Multi-arch compatible (amd64 + arm64).

□□ Alpine — Production Optimized

```
FROM code.foss.global/host.today/ht-docker-node:alpine-node

RUN nvm install 22 && nvm use 22
RUN pnpm install && pnpm build
# Result: ~200 MB image
```

Why Alpine?

- □ **60–75 % smaller** → Faster pulls, faster deploys
- □ **Reduced attack surface** → Fewer packages = fewer CVEs
- □ **Native musl builds** → No glibc compatibility layer
- □ **Multi-arch** → Same tag works on x64 and ARM64

□□ Usage Examples

Basic Node.js App

```
FROM code.foss.global/host.today/ht-docker-node:alpine-node

WORKDIR /app
COPY package*.json ./
RUN pnpm install

COPY . .
RUN pnpm build

EXPOSE 3000
CMD ["node", "dist/index.js"]
```

Multi-Version Testing

```
FROM code.foss.global/host.today/ht-docker-node:latest

WORKDIR /app
COPY package*.json ./

RUN nvm install 20 && nvm use 20 && npm ci && npm test
RUN nvm install 22 && nvm use 22 && npm ci && npm test

# Ship with Node 22
RUN nvm alias default 22 && npm run build
```

Deno Application

```
FROM code.foss.global/host.today/ht-docker-node:alpine-deno

WORKDIR /app
COPY . .

# Deno and Node.js are both available
CMD ["deno", "run", "--allow-net", "main.ts"]
```

Bun for Ultra-Fast Installs

```
FROM code.foss.global/host.today/ht-docker-node:alpine-bun

WORKDIR /app
COPY package.json bun.lockb ./
RUN bun install

COPY . .
RUN bun run build
CMD ["bun", "run", "start"]
```

TypeScript Multi-Stage Build

```
# Build stage
FROM code.foss.global/host.today/ht-docker-node:alpine-node AS builder
WORKDIR /app
COPY package*.json ./
RUN pnpm install
COPY tsconfig.json ./
COPY src ./src
RUN pnpm build

# Production stage — only runtime deps
FROM code.foss.global/host.today/ht-docker-node:alpine-node
WORKDIR /app
COPY package*.json ./
RUN pnpm install --prod
COPY --from=builder /app/dist ./dist
EXPOSE 3000
CMD ["node", "dist/index.js"]
```

Production-Hardened Setup

```
FROM code.foss.global/host.today/ht-docker-node:alpine-node

# Non-root user
RUN addgroup -g 1001 -S nodejs && adduser -S nodejs -u 1001

WORKDIR /app
COPY package*.json ./
RUN pnpm install --frozen-lockfile && pnpm cache clean

COPY --chown=nodejs:nodejs . .
RUN pnpm build

USER nodejs
EXPOSE 3000
CMD ["node", "dist/index.js"]
```

□□ NVM Cheat Sheet

```
# Install a specific version
nvm install 22.5.0

# Use a version (current shell)
nvm use 22

# Set default (persists across shells / RUN steps)
nvm alias default 22

# Install and switch to latest LTS
nvm install --lts && nvm use --lts

# List installed versions
nvm ls

# Chain in a single Dockerfile RUN
RUN nvm install 22 && nvm use 22 && npm ci && npm test
```

□□ Building the Images

This project uses @git.zone/tsdocker for Docker image management.

```
# Install tsdocker
pnpm install -g @git.zone/tsdocker@latest

# Discover all Dockerfiles and their tags
tsdocker list

# Build all images (multi-arch: amd64 + arm64)
tsdocker build

# Run all test scripts
tsdocker test
```

```
# Push to a specific registry
tsdocker push code.foss.global
```

Manual Build (single image)

```
docker buildx build \
  --platform linux/amd64,linux/arm64 \
  -f Dockerfile_alpine-node \
  -t your-registry/your-image:alpine-node \
  --push .
```

Image Dependency Chain

Some images depend on others being in the registry first:

```
Dockerfile (:latest) ──► Dockerfile_lts (:lts)
                     ──► Dockerfile_szci (:szci)
                          ──► Dockerfile_fossglobal_preinstalled_*
(:fossglobal_preinstalled_<ver>)

Dockerfile_alpine-node (:alpine-node)
                     ──► Dockerfile_alpine-szci (:alpine-szci)
```

The standalone Alpine images (`:alpine-bun`, `:alpine-deno`) have no registry dependencies.

□□ Image Comparison

Feature	Ubuntu `:latest`	Alpine `:alpine-node`
Base Size	~900 MB	~200 MB
Build Tools	□ Full (gcc, g++, make, python3)	⚠ Install separately (`apk add build-base`)
Chromium	□ Pre-installed	□
MongoDB	□ 8.0	□
Runtimes	Node + Bun + Deno + pnpm	Node + pnpm

Feature	Ubuntu `:latest`	Alpine `:alpine-node`
Compatibility	□ Maximum (glibc)	□ Good (musl)
Multi-arch	□ amd64, arm64	□ amd64, arm64
tini init	□	□
Best for	Complex builds, E2E tests, full-stack dev	Production, CI/CD, microservices

---

□□ Troubleshooting

NVM command not found

Shouldn't happen in our images, but if it does:

```
export NVM_DIR="/usr/local/nvm"
[ -s "$NVM_DIR/nvm.sh" ] && . "$NVM_DIR/nvm.sh"
```

Alpine native module build failures

Some npm packages require native build tools:

```
FROM code.foss.global/host.today/ht-docker-node:alpine-node
RUN apk add --no-cache python3 make g++
RUN pnpm install
```

Or use `:alpine-szci` which ships with build tools pre-installed.

Version not persisting across RUN steps

Each Dockerfile `RUN` creates a new shell. Use `nvm alias default`:

```
RUN nvm install 22 && nvm alias default 22
RUN node -v   # □ v22.x.x
```

Links

- **Source Code:** code.foss.global/host.today/ht-docker-node
- **NVM:** github.com/nvm-sh/nvm
- **tini:** github.com/krallin/tini
- **tsdocker:** code.foss.global/git.zone/tsdocker
- **Alpine Linux:** alpinelinux.org
- **Node.js Unofficial Builds:** unofficial-builds.nodejs.org (musl support)

License and Legal Information

Trademarks

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

Capital GmbH of any derivative works.

Revision #4
Created 2026-03-28 11:09:27 UTC by foss.global Team
Updated 2026-03-28 12:15:39 UTC by foss.global Team