readme.md for @git.zone/tsdocker

> □□ The ultimate Docker development toolkit for TypeScript projects — build, test, and ship multi-arch containerized applications with zero friction.

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.

What is tsdocker?

tsdocker is a comprehensive Docker development and build tool that handles everything from testing npm packages in clean environments to building and pushing multi-architecture Docker images across multiple registries — all from a single CLI.

□□ Key Capabilities

- □□ **Smart Docker Builds** — Automatically discover, sort, and build Dockerfiles by dependency
- □□ **True Multi-Architecture** — Build for `amd64` and `arm64` simultaneously with Docker Buildx
- □□ **Multi-Registry Push** — Ship to Docker Hub, GitLab, GitHub Container Registry, and more via OCI Distribution API
- ⚡ **Parallel Builds** — Level-based parallel builds with configurable concurrency
- □□ **Persistent Local Registry** — All images flow through a local OCI registry with persistent storage

- □□ **Build Caching** — Skip unchanged Dockerfiles with content-hash caching
- □□ **Dockerfile Filtering** — Build or push only specific Dockerfiles using glob patterns
- □□ **Resilient Push** — Automatic retry with exponential backoff, timeouts, and token refresh for rock-solid pushes
- □□ **CI-Safe Isolation** — Unique sessions per invocation prevent collisions in parallel CI pipelines
- □□ **Zero Config Start** — Works out of the box, scales with your needs

Installation

```
# Global installation (recommended for CLI usage)
npm install -g @git.zone/tsdocker

# Or project-local installation
pnpm install --save-dev @git.zone/tsdocker
```

Quick Start

□□ Build Docker Images

Got `Dockerfile` files? Build them all with automatic dependency ordering:

```
tsdocker build
```

tsdocker will:

1. □□ Discover all `Dockerfile*` files in your project
2. □□ Analyze `FROM` dependencies between them
3. □□ Sort them topologically
4. □□ Build each image in the correct order
5. □□ Push every image to a persistent local registry (`.nogit/docker-registry/`)

□□ Push to Registries

Ship your images to one or all configured registries:

```
# Push to all configured registries
tsdocker push

# Push to a specific registry
tsdocker push --registry=registry.gitlab.com

# Push without rebuilding (use existing images in local registry)
tsdocker push --no-build
```

Under the hood, `tsdocker push` uses the **OCI Distribution API** to copy images directly from the local registry to remote registries. This means multi-arch manifest lists are preserved end-to-end — no more single-platform-only pushes. Every request is protected with **automatic retry** (up to 6 attempts with exponential backoff) and **5-minute timeouts**, so transient network issues don't kill your push mid-transfer.

□□ Build Only Specific Dockerfiles

Target specific Dockerfiles by name pattern — dependencies are resolved automatically:

```
# Build only the base image
tsdocker build Dockerfile_base

# Build anything matching a glob pattern
tsdocker build Dockerfile_app*

# Push specific images only (skip build phase)
tsdocker push --no-build Dockerfile_api Dockerfile_web
```

CLI Commands

Command	Description
`tsdocker`	Show usage / man page
`tsdocker build`	Build all Dockerfiles with dependency ordering
`tsdocker push`	Build + push images to configured registries
`tsdocker pull <registry>`	Pull images from a specific registry
`tsdocker test`	Build + run container test scripts (`test_*.sh`)

Command	Description
`tsdocker login`	Authenticate with configured registries
`tsdocker list`	Display discovered Dockerfiles and their dependencies
`tsdocker config`	Manage global tsdocker configuration (remote builders, etc.)
`tsdocker clean`	Interactively clean Docker environment

Build Flags

Flag	Description
`<patterns>`	Positional Dockerfile name patterns (e.g. `Dockerfile_base`, `Dockerfile_app*`)
`--platform=linux/arm64`	Override build platform for a single architecture
`--timeout=600`	Build timeout in seconds
`--no-cache`	Force rebuild without Docker layer cache
`--cached`	Skip unchanged Dockerfiles (content-hash based)
`--verbose`	Stream raw `docker build` output
`--parallel`	Enable level-based parallel builds (default concurrency: 4)
`--parallel=8`	Parallel builds with custom concurrency
`--context=mycontext`	Use a specific Docker context

Push Flags

Flag	Description
`<patterns>`	Positional Dockerfile name patterns to select which images to push
`--registry=<url>`	Push to a single specific registry instead of all configured
`--no-build`	Skip the build phase; only push existing images from local registry

Config Subcommands

Subcommand	Description
`add-builder`	Add or update a remote builder node

Subcommand	Description
`remove-builder`	Remove a remote builder by name
`list-builders`	List all configured remote builders
`show`	Show the full global configuration

`add-builder` flags:

Flag	Description
`--name=<name>`	Builder name (e.g. `arm64-builder`)
`--host=<user@ip>`	SSH host (e.g. `armbuilder@192.168.1.100`)
`--platform=<p>`	Target platform (e.g. `linux/arm64`)
`--ssh-key=<path>`	SSH key path (optional, uses SSH agent/config by default)

Clean Flags

Flag	Description
`--all`	Include all images and volumes (not just dangling)
`-y`	Auto-confirm all prompts

Configuration

Configure tsdocker in your `.smartconfig.json` under the `@git.zone/tsdocker` key:

```
{
  "@git.zone/tsdocker": {
    "registries": ["registry.gitlab.com", "docker.io"],
    "registryRepoMap": {
      "registry.gitlab.com": "myorg/myproject"
    },
    "buildArgEnvMap": {
      "NODE_VERSION": "NODE_VERSION"
    },
    "platforms": ["linux/amd64", "linux/arm64"],
    "testDir": "./test"
  }
```

```
}
```

Configuration Options

Build & Push Options

Option	Type	Default	Description
`registries`	`string[]`	`[]`	Registry URLs to push to
`registryRepoMap`	`object`	`{}`	Map registries to different repository paths
`buildArgEnvMap`	`object`	`{}`	Map Docker build ARGs to environment variables
`platforms`	`string[]`	`["linux/amd64"]`	Target architectures for multi-arch builds
`testDir`	`string`	`./test`	Directory containing test scripts

Architecture: How tsdocker Works

tsdocker uses a **local OCI registry** as the canonical store for all built images. This design solves fundamental problems with Docker's local daemon, which cannot hold multi-architecture manifest lists.

□□ Build Flow

```
┌──────────────────────────────────────────────────┐
│  tsdocker build                                  │
│                                                  │
│  1. Start local registry (localhost:<dynamic-port>)  │
│     └── Persistent volume: .nogit/docker-registry/  │
│                                                  │
│  2. For each Dockerfile (topological order):         │
│     ├── Multi-platform: buildx --push → registry    │
│     └── Single-platform: docker build → registry    │
│                                                  │
│  3. Stop local registry (data persists on disk)      │
```

```
└──────────────────────────────────────────────────┘
```

□□ Push Flow

```
┌──────────────────────────────────────────────────┐
│  tsdocker push                                   │
│                                                  │
│  1. Start local registry (loads persisted data)  │
│                                                  │
│  2. For each image × each remote registry:       │
│     └── OCI Distribution API copy (with retry):  │
│         ├── Fetch manifest (single or multi-arch)│
│         ├── Copy blobs (skip if already exist)   │
│         ├── Retry up to 6× with exponential backoff│
│         └── Push manifest with destination tag   │
│                                                  │
│  3. Stop local registry                          │
└──────────────────────────────────────────────────┘
```

□□ Why a Local Registry?

Problem	Solution
`docker buildx --load` fails for multi-arch images	`buildx --push` to local registry works for any number of platforms
`docker push` only pushes single-platform manifests	OCI API copy preserves full manifest lists (multi-arch)
Images lost between build and push phases	Persistent storage at `.nogit/docker-registry/` survives restarts
Redundant blob uploads on incremental pushes	HEAD checks skip blobs that already exist on the remote

□□ Resilient Push

The OCI Distribution API client wraps every HTTP request with:

- **Timeouts** — 5-minute timeout for blob operations, 30-second timeout for auth/metadata calls via `AbortSignal.timeout()`
- **Automatic Retry** — Up to 6 attempts with exponential backoff (1s → 2s → 4s → 8s → 16s → 32s)

- **Smart Retry Logic** — Retries on network errors (`ECONNRESET`, `fetch failed`) and 5xx server errors; does NOT retry 4xx client errors
- **Token Refresh** — On 401 responses, the cached auth token is cleared so the next retry re-authenticates automatically

This means transient issues like stale connection pools, brief network blips, or token expiry during long multi-arch pushes (56+ blob operations) are handled gracefully instead of killing the entire transfer.

□□ CI-Safe Session Isolation

Every tsdocker invocation gets its own **session** with unique:

- **Session ID** — Random 8-char hex (override with `TSDOCKER_SESSION_ID`)
- **Registry port** — Dynamically allocated (override with `TSDOCKER_REGISTRY_PORT`)
- **Registry container** — Named `tsdocker-registry-<sessionId>`
- **Builder suffix** — In CI, the buildx builder gets a `-<sessionId>` suffix to prevent collisions

This prevents resource conflicts when multiple CI jobs run tsdocker in parallel. Auto-detected CI systems:

Environment Variable	CI System
`GITEA_ACTIONS`	Gitea Actions
`GITHUB_ACTIONS`	GitHub Actions
`GITLAB_CI`	GitLab CI
`CI`	Generic CI

In local dev, no suffix is added — keeping a persistent builder for faster rebuilds.

□□ Docker Context & Topology Detection

tsdocker automatically detects your Docker environment topology:

Topology	Detection	Meaning
`local`	Default	Standard Docker installation on the host
`socket-mount`	`/.dockerenv` exists	Running inside a container with Docker socket mounted
`dind`	`DOCKER_HOST` starts with `tcp://`	Docker-in-Docker setup

Context-aware builder names (`tsdocker-builder-<context>`) prevent conflicts across Docker contexts. Rootless Docker configurations trigger appropriate warnings.

Registry Authentication

Environment Variables

```
# Pipe-delimited format (supports DOCKER_REGISTRY_1 through DOCKER_REGISTRY_10)
export DOCKER_REGISTRY_1="registry.gitlab.com|username|password"
export DOCKER_REGISTRY_2="docker.io|username|password"

# Individual registry format
export DOCKER_REGISTRY_URL="registry.gitlab.com"
export DOCKER_REGISTRY_USER="username"
export DOCKER_REGISTRY_PASSWORD="password"
```

Docker Config Fallback

When pushing, tsdocker will also read credentials from `~/.docker/config.json` if no explicit credentials are provided via environment variables. This means `docker login` credentials work automatically. Docker Hub special cases (`docker.io`, `index.docker.io`, `registry-1.docker.io`) are all recognized.

Login Command

```
tsdocker login
```

Authenticates with all configured registries using the provided environment variables.

Advanced Usage

□□ Multi-Architecture Builds

Build for multiple platforms using Docker Buildx:

```
{
  "@git.zone/tsdocker": {
    "platforms": ["linux/amd64", "linux/arm64"]
  }
}
```

tsdocker automatically:

- Sets up a Buildx builder with `--driver-opt network=host` (so buildx can reach the local registry)
- Pushes multi-platform images to the local registry via `buildx --push`
- Copies the full manifest list (including all platform variants) to remote registries on `tsdocker push`

□□ Native Remote Builders

Instead of relying on slow QEMU emulation for cross-platform builds, tsdocker can use **native remote machines** via SSH as build nodes. For example, use a real arm64 machine for `linux/arm64` builds:

```
# Add a remote arm64 builder
tsdocker config add-builder \
  --name=arm64-builder \
  --host=armbuilder@192.168.1.100 \
  --platform=linux/arm64 \
  --ssh-key=~/.ssh/id_ed25519

# List configured builders
tsdocker config list-builders

# Remove a builder
tsdocker config remove-builder --name=arm64-builder

# Show full global config
tsdocker config show
```

Global configuration is stored at `~/.git.zone/tsdocker/config.json`.

How it works:

When remote builders are configured and the project's `platforms` includes a matching platform, tsdocker automatically:

1. Creates a **multi-node buildx builder** — local node for `linux/amd64`, remote SSH node for `linux/arm64`
2. Opens **SSH reverse tunnels** so the remote builder can push to the local staging registry
3. Builds natively on each platform's hardware — no QEMU overhead
4. Tears down tunnels after the build completes

```
[Local machine]                        [Remote arm64 machine]
  registry:2 on localhost:PORT  <──── SSH reverse tunnel ──── localhost:PORT
  BuildKit (amd64) ──push──>             BuildKit (arm64) ──push──>
     localhost:PORT                         localhost:PORT (tunneled)
```

Prerequisites for the remote machine:

- Docker installed and running
- A user with Docker group access (no sudo needed)
- SSH key access configured

⚡ Parallel Builds

Speed up builds by building independent images concurrently:

```
# Default concurrency (4 workers)
tsdocker build --parallel

# Custom concurrency
tsdocker build --parallel=8

# Works with caching too
tsdocker build --parallel --cached
```

tsdocker groups Dockerfiles into **dependency levels** using topological analysis. Images within the same level have no dependencies on each other and build in parallel. Each level completes before the next begins.

□□ Dockerfile Naming Conventions

tsdocker discovers files matching `Dockerfile*`:

File Name	Version Tag
`Dockerfile`	`latest`

File Name	Version Tag
`Dockerfile_v1.0.0`	`v1.0.0`
`Dockerfile_alpine`	`alpine`
`Dockerfile_##version##`	Uses `package.json` version

□□ Dockerfile Filtering

Build or push only the Dockerfiles you need. Positional arguments are matched against Dockerfile basenames as glob patterns:

```
# Build a single Dockerfile
tsdocker build Dockerfile_base

# Glob patterns with * and ? wildcards
tsdocker build Dockerfile_app*

# Multiple patterns
tsdocker build Dockerfile_base Dockerfile_web

# Push specific images without rebuilding
tsdocker push --no-build Dockerfile_api
```

When filtering for `build`, **dependencies are auto-resolved**: if `Dockerfile_app` depends on `Dockerfile_base`, specifying only `Dockerfile_app` will automatically include `Dockerfile_base` in the build order.

□□ Dependency-Aware Builds

If you have multiple Dockerfiles that depend on each other:

```
# Dockerfile_base
FROM node:20-alpine
RUN npm install -g typescript

# Dockerfile_app
FROM myproject:base
COPY . .
RUN npm run build
```

tsdocker automatically detects that `Dockerfile_app` depends on `Dockerfile_base`, builds them in the correct order, and makes the base image available to dependent builds via the local registry (using `--build-context` for buildx).

□□ Container Test Scripts

Create test scripts in your test directory:

```
# test/test_latest.sh
#!/bin/bash
node --version
npm --version
echo "Container tests passed!"
```

Run with:

```
tsdocker test
```

This builds all images, starts the local registry (so multi-arch images can be pulled), and runs each matching test script inside a container.

□□ Build Args from Environment

Pass environment variables as Docker build arguments:

```
{
  "@git.zone/tsdocker": {
    "buildArgEnvMap": {
      "NPM_TOKEN": "NPM_TOKEN",
      "NODE_VERSION": "NODE_VERSION"
    }
  }
}
```

```
ARG NPM_TOKEN
ARG NODE_VERSION=20
FROM node:${NODE_VERSION}
RUN echo "//registry.npmjs.org/:_authToken=${NPM_TOKEN}" > ~/.npmrc
```

□□ Registry Repo Mapping

Use different repository names for different registries:

```
{
  "@git.zone/tsdocker": {
    "registries": ["registry.gitlab.com", "docker.io"],
    "registryRepoMap": {
      "registry.gitlab.com": "mygroup/myproject",
      "docker.io": "myuser/myproject"
    }
  }
}
```

When pushing, tsdocker maps the local repo name to the registry-specific path. For example, a locally built `myproject:latest` becomes `registry.gitlab.com/mygroup/myproject:latest` and `docker.io/myuser/myproject:latest`.

□□ Listing Dockerfiles

Inspect your project's Dockerfiles and their relationships:

```
tsdocker list
```

Output:

```
Discovered Dockerfiles:
=======================

1. /path/to/Dockerfile_base
   Tag: myproject:base
   Base Image: node:20-alpine
   Version: base

2. /path/to/Dockerfile_app
   Tag: myproject:app
   Base Image: myproject:base
   Version: app
   Depends on: myproject:base
```

Examples

Minimal Build & Push

```
{
  "@git.zone/tsdocker": {
    "registries": ["docker.io"],
    "platforms": ["linux/amd64"]
  }
}
```

```
tsdocker push
```

Full Production Setup

```
{
  "@git.zone/tsdocker": {
    "registries": ["registry.gitlab.com", "ghcr.io", "docker.io"],
    "registryRepoMap": {
      "registry.gitlab.com": "myorg/myapp",
      "ghcr.io": "myorg/myapp",
      "docker.io": "myuser/myapp"
    },
    "buildArgEnvMap": {
      "NPM_TOKEN": "NPM_TOKEN"
    },
    "platforms": ["linux/amd64", "linux/arm64"],
    "testDir": "./docker-tests"
  }
}
```

CI/CD Integration

GitLab CI:

```
build-and-push:
  stage: build
  script:
    - npm install -g @git.zone/tsdocker
    - tsdocker push
  variables:
    DOCKER_REGISTRY_1: 'registry.gitlab.com|$CI_REGISTRY_USER|$CI_REGISTRY_PASSWORD'
```

GitHub Actions:

```
- name: Build and Push
  run: |
    npm install -g @git.zone/tsdocker
    tsdocker login
    tsdocker push
  env:
    DOCKER_REGISTRY_1: 'ghcr.io|${{ github.actor }}|${{ secrets.GITHUB_TOKEN }}'
```

Gitea Actions:

```
- name: Build and Push
  run: |
    npm install -g @git.zone/tsdocker
    tsdocker push
  env:
    DOCKER_REGISTRY_1: 'gitea.example.com|${{ secrets.REGISTRY_USER }}|${{
secrets.REGISTRY_PASSWORD }}'
```

tsdocker auto-detects all three CI systems and enables session isolation automatically — no extra configuration needed.

TypeScript API

tsdocker can also be used programmatically:

```
import { TsDockerManager } from '@git.zone/tsdocker/dist_ts/classes.tsdockermanager.js';
import type { ITsDockerConfig } from '@git.zone/tsdocker/dist_ts/interfaces/index.js';
```

```
const config: ITsDockerConfig = {
  registries: ['docker.io'],
  platforms: ['linux/amd64', 'linux/arm64'],
};

const manager = new TsDockerManager(config);
await manager.prepare();
await manager.build({ parallel: true });
await manager.push();
```

Environment Variables

CI & Session Control

Variable	Description
`TSDOCKER_SESSION_ID`	Override the auto-generated session ID (default: random 8-char hex)
`TSDOCKER_REGISTRY_PORT`	Override the dynamically allocated local registry port
`CI`	Generic CI detection (also `GITHUB_ACTIONS`, `GITLAB_CI`, `GITEA_ACTIONS`)

Registry Credentials

Variable	Description
`DOCKER_REGISTRY_1` through `DOCKER_REGISTRY_10`	Pipe-delimited: `registry|username|password`
`DOCKER_REGISTRY_URL`	Registry URL for single-registry setup
`DOCKER_REGISTRY_USER`	Username for single-registry setup
`DOCKER_REGISTRY_PASSWORD`	Password for single-registry setup

Requirements

- **Docker** — Docker Engine 20+ or Docker Desktop
- **Node.js** — Version 18 or higher (for native `fetch` and ESM support)

- **Docker Buildx** — Required for multi-architecture builds (included in Docker Desktop)

Troubleshooting

"docker not found"

Ensure Docker is installed and in your PATH:

```
docker --version
```

Multi-arch build fails

Make sure Docker Buildx is available. tsdocker will set up the builder automatically, but you can verify:

```
docker buildx version
```

Registry authentication fails

Check your environment variables are set correctly:

```
echo $DOCKER_REGISTRY_1
tsdocker login
```

tsdocker also falls back to `~/.docker/config.json` — ensure you've run `docker login` for your target registries.

Push fails with "fetch failed"

tsdocker automatically retries failed requests up to 6 times with exponential backoff. If pushes still fail:

- Check network connectivity to the target registry
- Verify your credentials haven't expired
- Look for retry log messages (`fetch failed (attempt X/6)`) to diagnose the pattern
- Large layers may need longer timeouts — the default 5-minute timeout per request should cover most cases

Circular dependency detected

Review your Dockerfiles' `FROM` statements — you have images depending on each other in a loop.

Build context too large

Use a `.dockerignore` file to exclude `node_modules`, `.git`, `.nogit`, and other large directories:

```
node_modules
.git
.nogit
dist_ts
```

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

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.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #5
Created 2026-03-28 10:49:33 UTC by foss.global Team
Updated 2026-03-28 12:15:31 UTC by foss.global Team