@git.zone/tsdocker

Documentation for @git.zone/tsdocker

- readme.md for @git.zone/tsdocker
- changelog.md for @git.zone/tsdocker

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.

changelog.md for @git.zone/tsdocker

2026-03-24 - 2.2.4 - fix(config)

migrate configuration loading to smartconfig and update build tooling compatibility

- switch configuration loading and documentation to .smartconfig.json
- upgrade build and test dependencies for tsbuild 4.4.0 and TypeScript 6 compatibility
- remove deprecated tsconfig baseUrl and paths settings and add an lru-cache override to avoid type issues

2026-03-24 - 2.2.3 - fix(config)

update workflow repository URL handling and package config file references

- Switches Gitea workflow repository URLs to use gitlab.com explicitly for authenticated CI git operations.
- Replaces the published config file entry from npmextra.json to .smartconfig.json in package metadata.
- Adds Rust build output directories to .gitignore.
- Refreshes changelog and README formatting to match the current smartconfig-based configuration.

2026-03-24 - 2.2.2 - fix(config)

rename npmextra configuration file to .smartconfig.json

- Moves the existing project metadata and release configuration from npmextra.json to .smartconfig.json without changing its contents.

2026-03-24 - 2.2.1 - fix(config)

switch configuration loading from npmextra to smartconfig

- replace the @push.rocks/npmextra dependency with @push.rocks/smartconfig
- update config initialization to use Smartconfig for the @git.zone/tsdocker settings
- refresh CLI help text to reference smartconfig.json instead of npmextra.json

2026-03-19 - 2.2.0 - feat(cli/buildx)

add pull control for builds and isolate buildx builders per project

- adds a new pull build option with --no-pull CLI support and defaults builds to refreshing base images with --pull
- passes the selected buildx builder explicitly into build commands instead of relying on global docker buildx use state
- generates project-hashed builder suffixes so concurrent runs from different project directories do not share the same local builder
- updates session logging to include project hash and builder suffix for easier build diagnostics

2026-03-15 - 2.1.0 - feat(cli)

add global remote builder configuration and native SSH buildx nodes for multi-platform builds

- adds a new `tsdocker config` command with subcommands to add, remove, list, and show remote builder definitions
- introduces global config support for remote builders stored under `~/.git.zone/tsdocker/config.json`
- builds can now create multi-node buildx setups with remote SSH builders and open reverse SSH tunnels so remote nodes can push to the local staging registry
- updates the README and CLI help to document remote builder configuration and native cross-platform build workflows

2026-03-12 - 2.0.2 - fix(repo)

no changes to commit

2026-03-12 - 2.0.1 - fix(repository)

no changes to commit

2026-03-12 - 2.0.0 - BREAKING CHANGE(cli)

remove legacy container test runner and make the default command show the man page

- Removes legacy testing and VS Code commands, including `runinside`, `vscode`, generated Dockerfile assets, and related configuration fields (`baseImage`, `command`, `dockerSock`, `keyValueObject`)
- Simplifies configuration and dependencies by dropping qenv-based env loading and unused legacy packages
- Updates CLI and documentation to reflect default help output and the current build/push-focused workflow

2026-02-07 - 1.17.4 - fix()

no changes

2026-02-07 - 1.17.3 - fix(registry)

increase default maxRetries in fetchWithRetry from 3 to 6 to improve resilience when fetching registry resources

- Changed default maxRetries from 3 to 6 in ts/classes.registrycopy.ts
- Reduces failures from transient network or registry errors by allowing more retry attempts
- No API or behavior changes besides the increased default retry count

2026-02-07 - 1.17.2 - fix(registry)

improve HTTP fetch retry logging, backoff calculation, and token-cache warning

- Include HTTP method in logs and normalize method to uppercase for consistency
- Log retry attempts with method, URL and calculated exponential backoff delay
- Compute and reuse exponential backoff delay variable instead of inline calculation
- Log error when a 5xx response persists after all retry attempts and when fetch ultimately fails
- Add a warning log when clearing cached token after a 401 response

2026-02-07 - 1.17.1 - fix(registrycopy)

add fetchWithRetry wrapper to apply timeouts, retries with exponential backoff, and token cache handling; use it for registry HTTP requests

- Introduces fetchWithRetry(url, options, timeoutMs, maxRetries) to wrap fetch with AbortSignal timeout, exponential backoff retries, and retry behavior only for network errors and 5xx responses
- Replaces direct fetch calls for registry /v2 checks, token requests, and blob uploads with fetchWithRetry (30s for auth/token checks, 300s for blob operations)
- Clears token cache entry when a 401 response is received so the next attempt re-authenticates
- Adds logging on retry attempts and backoff delays to improve robustness and observability

2026-02-07 - 1.17.0 - feat(tsdocker)

add Dockerfile filtering, optional skip-build flow, and fallback Docker config credential loading

- Add TsDockerManager.filterDockerfiles(patterns) to filter discovered Dockerfiles by glob-style patterns and warn when no matches are found
- Allow skipping image build with --no-build (argvArg.build === false): discover Dockerfiles and apply filters without performing build
- Fallback to load Docker registry credentials from ~/.docker/config.json via RegistryCopy.getDockerConfigCredentials when env vars do not provide credentials
- Import RegistryCopy and add info/warn logs when credentials are loaded or missing

2026-02-07 - 1.16.0 - feat(core)

Introduce per-invocation TsDockerSession and session-aware local registry and build orchestration; stream and parse buildx output for improved logging and visibility; detect Docker topology and add CI-safe cleanup; update README with multi-arch, parallel-build, caching, and local registry usage and new CLI flags.

- Add TsDockerSession to allocate unique ports, container names and builder suffixes for concurrent runs (especially in CI).
- Make local registry session-aware: start/stop/use registry container and persistent storage per session; retry on port conflicts.
- Inject session into Dockerfile instances and TsDockerManager; use session.config.registryHost for tagging/pushing and test container naming.
- Stream and parse buildx/docker build output via createBuildOutputHandler for clearer step/platform/CACHED/DONE logging and --progress=plain usage.
- Detect Docker topology (socket-mount, dind, local) in DockerContext and expose it in context info.
- Add manager.cleanup to remove CI-scoped buildx builders and ensure CLI calls cleanup after build/push/test.
- Update interfaces to include topology and adjust many Dockerfile/manager methods to be session-aware.
- Large README improvements: multi-arch flow, persistent local registry, parallel builds, caching, new CLI and clean flags, and examples for CI integration.

2026-02-07 - 1.15.1 - fix(registry)

use persistent local registry and OCI Distribution API image copy for pushes

- Adds RegistryCopy class implementing the OCI Distribution API to copy images (including multi-arch manifest lists) from the local registry to remote registries.
- All builds now go through a persistent local registry at localhost:5234 with volume storage at .nogit/docker-registry/; Dockerfile.startLocalRegistry mounts this directory.
- Dockerfile.push now delegates to RegistryCopy.copyImage; Dockerfile.needsLocalRegistry() always returns true and config.push is now a no-op (kept for backward compat).
- Multi-platform buildx builds are pushed to the local registry (this.localRegistryTag) during buildx --push; code avoids redundant pushes when images are already pushed by buildx.
- Build, cached build, test, push and pull flows now start/stop the local registry automatically to support multi-platform/image resolution.
- Introduces Dockerfile.getDestRepo and support for config.registryRepoMap to control destination repository mapping.

- Breaking change: registry usage and push behavior changed (config.push ignored and local registry mandatory) — bump major version.

2026-02-07 - 1.15.0 - feat(clean)

Make the `clean` command interactive: add smartinteract prompts, docker context detection, and selective resource removal with support for --all and -y auto-confirm

- Adds dependency @push.rocks/smartinteract and exposes it from the plugins module
- Refactors tsdocker.cli.ts clean command to list Docker resources and prompt checkbox selection for running/stopped containers, images, and volumes
- Adds DockerContext detection and logging to determine active Docker context
- Introduces auto-confirm (-y) and --all handling to either auto-accept or allow full-image/volume removal
- Replaces blunt shell commands with safer, interactive selection and adds improved error handling and logging

2026-02-07 - 1.14.0 - feat(build)

add level-based parallel builds with --parallel and configurable concurrency

- Introduces --parallel and --parallel= CLI flags to enable level-based parallel Docker builds (default concurrency 4).
- Adds Dockerfile.computeLevels() to group topologically-sorted Dockerfiles into dependency levels.
- Adds Dockerfile.runWithConcurrency() implementing a bounded-concurrency worker-pool (fast-fail via Promise.all).
- Integrates parallel build mode into Dockerfile.buildDockerfiles() and TsDockerManager.build() for both cached and non-cached flows, including tagging and pushing for dependency resolution after each level.
- Adds options.parallel and options.parallelConcurrency to the build interface and wires them through the CLI and manager.
- Updates documentation (readme.hints.md) with usage examples and implementation notes.

2026-02-07 - 1.13.0 - feat(docker)

add Docker context detection, rootless support, and context-aware buildx registry handling

- Introduce DockerContext class to detect current Docker context and rootless mode and to log warnings and context info
- Add IDockerContextInfo interface and a new context option on build/config to pass explicit Docker context
- Propagate --context CLI flag into TsDockerManager.prepare so CLI commands can set an explicit Docker context
- Make buildx builder name context-aware (tsdocker-builder-) and log builder name/platforms
- Pass isRootless into local registry startup and build pipeline; emit rootless-specific warnings and registry reachability hint

2026-02-06 - 1.12.0 - feat(docker)

add detailed logging for buildx, build commands, local registry, and local dependency info

- Log startup of local registry including a note about buildx dependency bridging
- Log constructed build commands and indicate whether buildx or standard docker build is used (including platforms and --push/--load distinctions)
- Emit build mode summary at start of build phase and report local base-image dependency mappings
- Report when --no-cache is enabled and surface buildx setup readiness with configured platforms
- Non-functional change: purely adds informational logging to improve observability during builds

2026-02-06 - 1.11.0 - feat(docker)

start temporary local registry for buildx dependency resolution and ensure buildx builder uses host network

- Introduce a temporary local registry (localhost:5234) with start/stop helpers and push support to expose local images for buildx
- Add Dockerfile.needsLocalRegistry to decide when a local registry is required (local base dependencies + multi-platform or platform option)
- Push built images to the local registry and set localRegistryTag on Dockerfile instances for BuildKit build-context usage
- Tag built images in the host daemon for dependent Dockerfiles to resolve local FROM references
- Integrate registry lifecycle into Dockerfile.buildDockerfiles and TsDockerManager build flows (start before builds, stop after)

- Ensure buildx builder is created with --driver-opt network=host and recreate existing builder if it lacks host network to allow registry access from build containers

2026-02-06 - 1.10.0 - feat(classes.dockerfile)

support using a local base image as a build context in buildx commands

- Adds --build-context flag mapping base image to docker-image://<localTag> when localBaseImageDependent && localBaseDockerfile are set
- Appends the build context flag to both single-platform and multi-platform docker buildx commands
- Logs an info message indicating the local build context mapping

2026-02-06 - 1.9.0 - feat(build)

add verbose build output, progress logging, and timing for builds/tests

- Add 'verbose' option to build/test flows (interfaces, CLI, and method signatures) to allow streaming raw docker build output or run silently
- Log per-item progress for build and test phases (e.g. (1/N) Building/Testing) and report individual durations
- Return elapsed time from Dockerfile.build() and Dockerfile.test() and aggregate total build/test times in manager
- Introduce formatDuration(ms) helper in logging module to format timings
- Switch from console.log to structured logger calls across cache, manager, dockerfile and push paths
- Use silent exec variants when verbose is false and stream exec when verbose is true

2026-02-06 - 1.8.0 - feat(build)

add optional content-hash based build cache to skip rebuilding unchanged Dockerfiles

- Introduce TsDockerCache to compute SHA-256 of Dockerfile content and persist cache to .nogit/tsdocker_support.json
- Add ICacheEntry and ICacheData interfaces and a cached flag to IBuildCommandOptions

- Integrate cached mode in TsDockerManager: skip builds on cache hits, verify image presence, record builds on misses, and still perform dependency tagging
- Expose --cached option in CLI to enable the cached build flow
- Cache records store contentHash, imageId, buildTag and timestamp

2026-02-06 - 1.7.0 - feat(cli)

add CLI version display using commitinfo

- Imported commitinfo from './00_commitinfo_data.js' and called tsdockerCli.addVersion(commitinfo.version) to surface package/commit version in the Smartcli instance
- Change made in ts/tsdocker.cli.ts — small user-facing CLI enhancement; no breaking changes

2026-02-06 - 1.6.0 - feat(docker)

add support for no-cache builds and tag built images for local dependency resolution

- Introduce IBuildCommandOptions.noCache to control --no-cache behavior
- Propagate noCache from CLI (via cache flag) through TsDockerManager to Dockerfile.build
- Append --no-cache to docker build/buildx commands when noCache is true
- After building an image, tag it with full base image references used by dependent Dockerfiles so their FROM lines resolve to the locally-built image
- Log tagging actions and execute docker tag via smartshellInstance

2026-02-06 - 1.5.0 - feat(build)

add support for selective builds, platform override and build timeout

- Introduce IBuildCommandOptions with patterns, platform and timeout to control build behavior
- Allow manager.build() to accept options and build only matching Dockerfiles (including dependencies) preserving topological order
- Add CLI parsing for build/push to accept positional Dockerfile patterns and --platform/--timeout flags
- Support single-platform override via docker buildx and multi-platform buildx detection
- Implement streaming exec with timeout to kill long-running builds and surface timeout errors

2026-02-04 - 1.4.3 - fix(dockerfile)

fix matching of base images to local Dockerfiles by stripping registry prefixes when comparing image references

- Added Dockerfile.extractRepoVersion(imageRef) to normalize image references by removing registry prefixes (detects registries containing '.' or ':' or 'localhost').
- Use extractRepoVersion when checking tagToDockerfile and when mapping local base dockerfiles to ensure comparisons use repo:tag keys rather than full registry-prefixed references.
- Prevents mismatches when baseImage includes a registry (e.g. "host.today/repo:version") so it correctly matches a local cleanTag like "repo:version".

2026-01-21 - 1.4.2 - fix(classes.dockerfile)

use a single top-level fs import instead of requiring fs inside methods

- Added top-level import: import * as fs from 'fs' in ts/classes.dockerfile.ts
- Removed inline require('fs') calls and replaced with the imported fs in constructor and test() to keep imports consistent
- No behavioral change expected; this is a cleanup/refactor to standardize module usage

2026-01-20 - 1.4.1 - fix(docs)

update README: expand usage, installation, quick start, features, troubleshooting and migration notes

- Expanded README content: new Quick Start, Installation examples, and detailed Features section (containerized testing, smart Docker builds, multi-registry push, multi-architecture support, zero-config start)
- Added troubleshooting and performance tips including registry login guidance and circular dependency advice
- Updated migration notes from legacy npmdocker to @git.zone/tsdocker (command and config key changes, ESM guidance)
- Documentation-only change — no source code modified

2026-01-20 - 1.4.0 - feat(tsdocker)

add multi-registry and multi-arch Docker build/push/pull manager, registry storage, Dockerfile handling, and new CLI commands

- Introduce TsDockerManager orchestrator to discover, sort, build, test, push and pull Dockerfiles
- Add Dockerfile class with dependency-aware build order, buildx support, push/pull and test flows (new large module)
- Add DockerRegistry and RegistryStorage classes to manage registry credentials, login/logout and environment loading
- Add CLI commands: build, push, pull, test, login, list (and integrate TsDockerManager into CLI)
- Extend configuration (ITsDockerConfig) with registries, registryRepoMap, buildArgEnvMap, platforms, push and testDir; re-export as IConfig for backwards compatibility
- Add @push.rocks/lik to dependencies and import it in tsdocker.plugins
- Remove legacy speedtest command and related package.json script
- Update README and readme.hints with new features, configuration examples and command list

2026-01-19 - 1.3.0 - feat(packaging)

Rename package scope to @git.zone and migrate to ESM; rename CLI/config keys, update entrypoints and imports, bump Node requirement to 18, and adjust scripts/dependencies

- Package renamed to @git.zone/tsdocker (scope change) — consumers must update package reference.
- Configuration key changed from 'npmdocker' to '@git.zone/tsdocker' in npmextra.json; update project config accordingly.
- CLI command renamed from 'npmdocker' to 'tsdocker' and entrypoint/entrypoint binary references updated.
- Project migrated to ESM: imports now use .js extensions, package main/typings point to dist_ts, and ts source uses ESM patterns — Node >=18 required.
- Build/test scripts changed to use tsx and updated test task names; CI/workflow and npmextra release registries updated.
- Dependencies/devDependencies bumped; smartfs, smartcli and tsbuild versions updated.
- Docker build command now uses '--load' and default base images/installation behavior adjusted (global install of tsdocker in image).

2025-12-13 - 1.2.43 - fix(packaging)

Rename package scope to @git.zone and migrate deps/CI; pin pnpm and enable ESM packaging

- Rename npm package scope from @gitzone/tsdocker to @git.zone/tsdocker (package.json, commitinfo, README, npmextra)
- Migrate devDependencies from @gitzone/_ to @git.zone/_ and ensure runtime packages use @push.rocks/* where applicable
- Replace smartfile usage with smartfs and update code to use async smartfs.file(...).write()/delete() patterns
- Add packageManager pin for pnpm, set type: "module", add files array and pnpm.overrides in package.json
- Add tsconfig.json with NodeNext/ES2022 settings and other ESM-related adjustments
- Add Gitea CI workflows (.gitea/workflows/default_tags.yaml and default_nottags.yaml) for test, audit and release flows
- Update assets Dockerfile to reference @git.zone/tsdocker and other packaging/CI related scripts
- Update .gitignore to consolidate dist patterns and add AI/tooling excludes (.claude/, .serena/)
- Update README, readme.hints.md and changelog to document the scope rename, dependency migrations and SmartFS migration

2025-11-22 - 1.2.42 - fix(package.json)

Add packageManager field to package.json to pin pnpm version

- Add packageManager: "pnpm@10.18.1+sha512.77a884a165cbba2d8d1c19e3b4880eee6d2fcabd0d879121e282196b80042351d5eb3ca0935fa599da1dc51265cc68816ad2bddd2a2de5ea9fdf92adbec7cd34" to package.json to lock pnpm CLI version and integrity

2025-11-22 - 1.2.41 - fix(core)

Migrate to @git.zone / @push.rocks packages, replace smartfile with smartfs and adapt filesystem usage; update dev deps and remove CI/lint config

- Updated devDependencies from @gitzone/_ to @git.zone/_ (tsbuild, tsrun, tstest) and bumped versions
- Re-scoped runtime dependencies from @pushrocks/_ to @push.rocks/_ and updated package versions
- Replaced deprecated smartfile usage with new async smartfs API; added SmartFs instance in ts/tsdocker.plugins.ts
- Switched sync filesystem calls to Node fs where appropriate (fs.existsSync, fs.mkdirSync) and updated code to await smartfs.file(...).write()/delete()
- Made buildDockerFile async and awaited file write/delete operations to ensure correct async flow
- Updated CLI bootstrap to require @git.zone/tsrun in cli.ts.js
- Removed tslint.json and cleaned up CI configuration (.gitlab-ci.yml content removed)
- Added readme.hints.md describing the migration and dependency changes

2021-09-30 - 1.2.40 - release (no code changes)

Routine release tag with no recorded source changes.

- Tagged release only (no changelogged changes).

2021-09-30 - 1.2.39 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2019-05-28 - 1.2.38 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2019-05-27 - 1.2.37 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2019-05-27 - 1.2.36 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2019-05-21 - 1.2.35 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2019-05-21 - 1.2.34 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2019-05-12 - 1.2.33 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2019-05-12 - 1.2.32 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2019-05-12 - 1.2.31 - fix(bin name)

Rename of the published CLI binary.

- Changed published binary name from "npmdocker" to "tsdocker".

2019-05-10 - 1.2.30 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2019-05-10 - 1.2.29 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2019-05-10 - 1.2.28 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2019-05-09 - 1.2.27 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2018-10-29 - 1.2.26 - fix(ci)

CI build process change.

- Removed "npmts" from the build process.

2018-10-29 - 1.2.25 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2018-10-28 - 1.2.24 - fix(clean)

Improved image cleanup.

- Images are now cleaned in a more thorough way.

2018-09-16 - 1.2.23 - fix(core)

Core maintenance updates.

- Internal core updates and maintenance.

2018-09-16 - 1.2.22 - fix(dependencies)

Dependency updates.

- Updated dependencies (maintenance).

2018-07-21 - 1.2.21 - fix(update to latest standards)

Standards/update alignment.

- Updated codebase to latest standards (general maintenance).

2018-05-18 - 1.2.20 - release (no code changes)

Tagged release with no recorded source changes.

- Tagged release only (no changelogged changes).

2018-05-18 - 1.2.19 - fix(ci)

CI improvements.

- Added a build command to package.json to support CI builds.

2018-05-18 - 1.2.18 - fix(package)

Packaging change for scoped publish.

- Include npmdocker under the @git.zone npm scope.

2018-01-24 - 1.2.18 - update

Documentation update.

- Updated package description.

2017-10-13 - 1.2.17 - fix(cleanup)

Cleanup behavior fix.

- Now cleans up correctly after operations.

2017-10-13 - 1.2.16 - update

Miscellaneous updates.

- General maintenance and updates.

2017-10-13 - 1.2.15 - fix(test)

Testing improvements.

- Fixed Docker testing.

2017-10-07 - 1.2.14 - ci

CI improvements.

- Updated CI configuration.

2017-10-07 - 1.2.13 - update(analytics)

Analytics integration.

- Updated Analytics integration.

2017-10-07 - 1.2.12 - update(dependencies)

Dependency updates.

- Updated dependencies.

2017-07-16 - 1.2.11 - update

Dependency and greeting update.

- Updated dependencies and changed greeting text.

2017-04-21 - 1.2.10 - feature

Added analytics.

- Now includes SmartAnalytics.

2017-04-02 - 1.2.8 - docs & ci

Docs and CI updates.

- Updated README and CI configuration.

2017-04-02 - 1.2.7 - fix(command)

Command execution fix.

- Fixed command execution behavior.

2017-03-28 - 1.2.6 - ci

CI configuration update.

- Updated .gitlab-ci.yml for correct images/steps.

2017-03-28 - 1.2.5 - ci

CI improvements.

- Further CI updates.

2017-03-28 - 1.2.4 - perf

Performance improvements.

- Now runs asynchronously and is significantly faster.

2017-02-12 - 1.2.3 - feature

New cleanup and diagnostics features.

- Added speedtest utility.
- Added removal of volumes.

2017-02-11 - 1.2.2 - feature

Cleanup enhancement.

- Added "clean --all" option to remove more artifacts.

2017-02-11 - 1.2.1 - maintenance

Docs and dependency updates.

- Updated README and dependencies.

2016-08-04 - 1.2.0 - maintenance

Dependency cleanup.

- Removed unnecessary dependencies.

2016-07-29 - 1.1.6 - feature

Environment support.

- Added support for qenv.

2016-07-29 - 1.1.5 - fix

Container cleanup improvements.

- Now also removes old running containers.

2016-07-29 - 1.1.4 - fix

Namespace conflict avoidance.

- Removes previous containers to avoid name-space conflicts after errors.

2016-07-29 - 1.1.3 - ci

CI image configuration.

- Added correct images for GitLab CI.

2016-07-29 - 1.1.2 - ci

CI fixes.

- Fixed GitLab CI configuration.

2016-07-28 - 1.1.1 - ci

CI fixes and configuration.

- Fixed gitlab.yml and CI issues.

2016-07-28 - 1.1.0 - feature

Docker-in-Docker support.

- Improved support for Docker-in-Docker scenarios.

2016-07-28 - 1.0.5 - feature & ci

Docker socket option and CI update.

- Added dockerSock option.
- Updated .gitlab-ci.yml.

2016-07-19 - 1.0.4 - release (no code changes)

Tagged release with no recorded source changes.

- Tagged release only (no changelogged changes).

2016-07-19 - 1.0.3 - feature

Environment tagging.

- Added environment tag support.

2016-07-19 - 1.0.2 - milestone

CLI and stability improvements.

- Wired up CLI usage.

- Marked as fully working.

2016-07-19 - 1.0.1 - initial improvements

Early project refinements and Docker integration.

- Added/updated Docker integration and configuration.
- Improved config handling and path management.
- Updated Docker handling and removed test artifacts.

2016-07-13 - 1.0.0 - initial

Initial release.

- Added README and initial project scaffolding.