@apiclient.xyz/docker

Documentation for @apiclient.xyz/docker

- readme.md for @apiclient.xyz/docker
- changelog.md for @apiclient.xyz/docker

readme.md for @apiclient.xyz/docker

A fully typed TypeScript client for the Docker Engine API. Talk to Docker from Node.js or Deno with a clean, object-oriented interface — containers, images, networks, services, secrets, and image storage all in one package. □□

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.

Install

```
pnpm install @apiclient.xyz/docker
# or
npm install @apiclient.xyz/docker
```

Usage

`DockerHost` is the single entry point. Every Docker resource — containers, images, networks, services, secrets — is managed through it.

```
import { DockerHost } from '@apiclient.xyz/docker';
```

□□ Setting Up the Docker Host

```
// Default: auto-detects Docker socket from common locations
const docker = new DockerHost({});

// Custom socket path
const docker = new DockerHost({
  socketPath: 'http://unix:/var/run/docker.sock:',
});

// Custom image store directory for local image caching
const docker = new DockerHost({
  imageStoreDir: '/tmp/my-image-store',
});

// Start the host (initializes the image store)
await docker.start();

// When done, stop the host
await docker.stop();
```

Socket path resolution order:

1. Explicit `socketPath` constructor option (highest priority)
2. `DOCKER_HOST` environment variable
3. `http://docker:2375/` when running in CI (the `CI` env var is set)
4. `http://unix:/var/run/docker.sock:` as the default

□□ Health Check and Version

```
// Ping the Docker daemon to verify it is accessible
await docker.ping();

// Get Docker daemon version information
const version = await docker.getVersion();
console.log(`Docker ${version.Version} (API ${version.ApiVersion})`);
console.log(`Platform: ${version.Os}/${version.Arch}`);
console.log(`Kernel: ${version.KernelVersion}`);
```

Field	Type	Description
`Version`	string	Docker engine version

Field	Type	Description
`ApiVersion`	string	API version
`MinAPIVersion`	string	Minimum supported API version
`GitCommit`	string	Git commit of the build
`GoVersion`	string	Go compiler version
`Os`	string	Operating system (e.g., `linux`)
`Arch`	string	Architecture (e.g., `amd64`)
`KernelVersion`	string	Host kernel version
`BuildTime`	string	Build timestamp

□□ Authentication

```
// Authenticate with explicit credentials
await docker.auth({
  serveraddress: 'https://index.docker.io/v1/',
  username: 'myuser',
  password: 'mypassword',
});

// Authenticate using credentials stored in ~/.docker/config.json
await docker.getAuthTokenFromDockerConfig('https://registry.gitlab.com');
```

□□ Docker Swarm

```
// Activate swarm with automatic IP detection
await docker.activateSwarm();

// Activate swarm with a specific advertisement address
await docker.activateSwarm('192.168.1.100');
```

□□ Docker Events

Subscribe to real-time Docker daemon events using an RxJS Observable.

```
const eventObservable = await docker.getEventObservable();

const subscription = eventObservable.subscribe((event) => {
  console.log(`Event: ${event.Type} ${event.Action}`);
  console.log(`Actor: ${event.Actor?.ID}`);
});

// Later: unsubscribe to stop listening
subscription.unsubscribe();
```

□□ Containers

Listing Containers

```
const containers = await docker.listContainers();

for (const container of containers) {
  console.log(`${container.Names[0]} - ${container.State} (${container.Status})`);
  console.log(`  Image: ${container.Image}`);
  console.log(`  ID: ${container.Id}`);
}
```

Getting a Container by ID

Returns `undefined` if the container does not exist.

```
const container = await docker.getContainerById('abc123def456');

if (container) {
  console.log(`Found container: ${container.Names[0]}`);
} else {
  console.log('Container not found');
}
```

Creating a Container

```
const container = await docker.createContainer({
  Hostname: 'my-container',
  Domainname: 'example.com',
  networks: ['my-network'],
});
```

Container Lifecycle

```
// Start a container
await container.start();

// Stop a container (with optional timeout in seconds)
await container.stop();
await container.stop({ t: 30 });

// Remove a container
await container.remove();
await container.remove({ force: true, v: true }); // Force removal and delete volumes
```

Inspecting a Container

```
const details = await container.inspect();
console.log(`State: ${details.State.Status}`);
console.log(`PID: ${details.State.Pid}`);
```

Refreshing Container State

Reload the container's properties from the Docker daemon.

```
await container.refresh();
console.log(`Current state: ${container.State}`);
```

Container Logs

```
// Get logs as a string (one-shot)
const logs = await container.logs({
  stdout: true,
  stderr: true,
  timestamps: true,
  tail: 100,         // Last 100 lines
  since: 1609459200, // Unix timestamp
});
console.log(logs);

// Stream logs continuously (follow mode)
const logStream = await container.streamLogs({
  stdout: true,
  stderr: true,
  timestamps: true,
  tail: 50,
});

logStream.on('data', (chunk) => {
  process.stdout.write(chunk.toString());
});

logStream.on('end', () => {
  console.log('Log stream ended');
});
```

Container Stats

```
// Get a single stats snapshot
const stats = await container.stats({ stream: false });
console.log(`CPU usage: ${stats.cpu_stats.cpu_usage.total_usage}`);
console.log(`Memory usage: ${stats.memory_stats.usage}`);
```

Attaching to a Container

Attach to the container's main process (PID 1) for bidirectional communication.

```
const { stream, close } = await container.attach({
  stdin: true,
  stdout: true,
  stderr: true,
  stream: true,
  logs: true, // Include previous logs
});

// Read output from the container
stream.on('data', (chunk) => {
  process.stdout.write(chunk.toString());
});

// Send input to the container
stream.write('echo hello\n');

// Detach when done
await close();
```

Executing Commands in a Container

Run a command inside a running container with full streaming support. The command argument can be a string (wrapped in `/bin/sh -c`) or an array of strings.

```
// Simple command execution
const { stream, close, inspect } = await container.exec('ls -la /app', {
  tty: true,
});

let output = '';
stream.on('data', (chunk) => {
  output += chunk.toString();
});

stream.on('end', async () => {
  // Check the exit code after the command finishes
  const info = await inspect();
  console.log(`Exit code: ${info.ExitCode}`);
  console.log(`Output:\n${output}`);
```

```
  await close();
});
```

```
// Execute with advanced options
const { stream, close, inspect } = await container.exec(
  ['python', '-c', 'print("hello from python")'],
  {
    tty: false,
    env: ['MY_VAR=hello', 'DEBUG=1'],
    workingDir: '/app',
    user: 'appuser',
    attachStdin: true,
    attachStdout: true,
    attachStderr: true,
  }
);

stream.on('data', (chunk) => {
  console.log(chunk.toString());
});

stream.on('end', async () => {
  const info = await inspect();
  if (info.ExitCode === 0) {
    console.log('Command succeeded');
  } else {
    console.log(`Command failed with exit code ${info.ExitCode}`);
  }
  await close();
});
```

The `inspect()` method on the exec result returns an `IExecInspectInfo` object:

Field	Type	Description
`ExitCode`	number	Exit code of the process (0 = success)
`Running`	boolean	Whether the exec process is still running
`Pid`	number	Process ID
`ContainerID`	string	Container where the exec ran

Field	Type	Description
`ID`	string	Exec instance ID
`OpenStderr`	boolean	Whether stderr is open
`OpenStdin`	boolean	Whether stdin is open
`OpenStdout`	boolean	Whether stdout is open
`CanRemove`	boolean	Whether the exec instance can be removed
`DetachKeys`	string	Detach keys
`ProcessConfig`	object	Process config (tty, entrypoint, arguments, privileged)

Container Properties

Each `DockerContainer` instance exposes these properties:

Property	Type	Description
`Id`	string	Container ID
`Names`	string[]	Container names
`Image`	string	Image name
`ImageID`	string	Image ID
`Command`	string	Command used to start the container
`Created`	number	Creation timestamp
`Ports`	array	Port mappings
`Labels`	object	Key-value label pairs
`State`	string	Container state (running, exited, etc.)
`Status`	string	Human-readable status string
`HostConfig`	object	Host configuration
`NetworkSettings`	object	Network configuration and IP addresses
`Mounts`	any	Volume mounts

□□ Images

Listing Images

```
const images = await docker.listImages();

for (const image of images) {
  console.log(`Tags: ${image.RepoTags?.join(', ')}`);
  console.log(`  Size: ${(image.Size / 1024 / 1024).toFixed(2)} MB`);
  console.log(`  ID: ${image.Id}`);
}
```

Getting an Image by Name

```
const image = await docker.getImageByName('nginx:latest');

if (image) {
  console.log(`Found image: ${image.RepoTags[0]}`);
  console.log(`Size: ${image.Size} bytes`);
}
```

Pulling an Image from a Registry

```
// Pull with explicit tag
const image = await docker.createImageFromRegistry({
  imageUrl: 'nginx',
  imageTag: 'latest',
});

// Pull with tag embedded in the URL
const image = await docker.createImageFromRegistry({
  imageUrl: 'node:20-alpine',
});

// Pull from a private registry (authenticate first)
await docker.auth({
  serveraddress: 'https://registry.gitlab.com',
  username: 'deploy-token',
```

```
  password: 'my-token',
});

const image = await docker.createImageFromRegistry({
  imageUrl: 'registry.gitlab.com/myorg/myapp:v1.2.3',
});
```

Importing an Image from a Tar Stream

```
import * as fs from 'node:fs';

const tarStream = fs.createReadStream('/path/to/image.tar');
const image = await docker.createImageFromTarStream(tarStream, {
  imageUrl: 'myapp:imported',
});
console.log(`Imported: ${image.RepoTags[0]}`);
```

Exporting an Image to a Tar Stream

```
import * as fs from 'node:fs';

const image = await docker.getImageByName('myapp:latest');
const tarStream = await image.exportToTarStream();

const writeStream = fs.createWriteStream('/path/to/output.tar');
tarStream.pipe(writeStream);

writeStream.on('finish', () => {
  console.log('Image exported successfully');
});
```

Pulling the Latest Version of an Image

```
const image = await docker.getImageByName('nginx:latest');
await image.pullLatestImageFromRegistry();
```

Removing an Image

```
const image = await docker.getImageByName('old-image:v1');
await image.remove();

// Force remove (even if referenced by containers)
await image.remove({ force: true });

// Remove without deleting untagged parent images
await image.remove({ noprune: true });
```

Pruning Unused Images

```
// Prune dangling (untagged) images
const result = await docker.pruneImages({ dangling: true });
console.log(`Deleted: ${result.ImagesDeleted?.length || 0} image layers`);
console.log(`Reclaimed: ${(result.SpaceReclaimed / 1024 / 1024).toFixed(2)} MB`);

// Prune images older than 7 days
const result = await docker.pruneImages({
  filters: {
    until: ['168h'],
  },
});

// Prune images matching specific labels
const result = await docker.pruneImages({
  filters: {
    label: ['environment=staging'],
  },
});
```

Getting the Image Version Label

```
const version = await image.getVersion();
console.log(`Image version: ${version}`); // Returns the "version" label value, or "0.0.0"
```

Image Properties

Each `DockerImage` instance exposes these properties:

Property	Type	Description
`Id`	string	Image ID
`RepoTags`	string[]	Repository tags
`RepoDigests`	string[]	Repository digests
`Created`	number	Creation timestamp
`Size`	number	Image size in bytes
`VirtualSize`	number	Virtual size in bytes
`SharedSize`	number	Shared size in bytes
`Labels`	object	Key-value label pairs
`ParentId`	string	Parent image ID
`Containers`	number	Number of containers using the image

□□ Networks

Listing Networks

```
const networks = await docker.listNetworks();

for (const network of networks) {
  console.log(`${network.Name} (${network.Driver}) - ${network.Scope}`);
  console.log(`  ID: ${network.Id}`);
  console.log(`  Attachable: ${network.Attachable}`);
}
```

Getting a Network by Name

```
const network = await docker.getNetworkByName('my-overlay-network');
```

```
if (network) {
  console.log(`Network: ${network.Name}`);
  console.log(`Driver: ${network.Driver}`);
  console.log(`Scope: ${network.Scope}`);
}
```

Creating a Network

```
// Simple overlay network (default driver)
const network = await docker.createNetwork({
  Name: 'my-overlay',
});

// Bridge network with custom IPAM configuration
const network = await docker.createNetwork({
  Name: 'custom-bridge',
  Driver: 'bridge',
  IPAM: {
    Config: [{
      Subnet: '172.20.0.0/16',
      Gateway: '172.20.0.1',
      IPRange: '172.20.10.0/24',
    }],
  },
  Labels: { environment: 'production' },
});

// Internal network (no external access)
const network = await docker.createNetwork({
  Name: 'internal-net',
  Driver: 'overlay',
  Internal: true,
  Attachable: true,
  EnableIPv6: false,
});
```

Network creation options:

Field	Type	Default	Description

`Name`	string	(required)	Network name
`Driver`	string	`'overlay'`	Network driver: `bridge`, `overlay`, `host`, `none`, `macvlan`
`Attachable`	boolean	`true`	Whether non-service containers can attach
`Labels`	object	--	Key-value label pairs
`IPAM`	object	--	IP Address Management configuration
`Internal`	boolean	`false`	Restrict external access to the network
`EnableIPv6`	boolean	`false`	Enable IPv6

IPAM Config entries support: `Subnet` (CIDR), `Gateway`, `IPRange`, `AuxiliaryAddresses`.

Listing Containers on a Network

```
const network = await docker.getNetworkByName('my-overlay');
const containers = await network.listContainersOnNetwork();

for (const container of containers) {
  console.log(`${container.Name}: ${container.IPv4Address}`);
}
```

Getting Containers for a Specific Service on a Network

```
const network = await docker.getNetworkByName('my-overlay');
const service = await docker.getServiceByName('web');
const serviceContainers = await network.getContainersOnNetworkForService(service);

for (const container of serviceContainers) {
  console.log(`${container.Name}: ${container.IPv4Address}`);
}
```

Removing a Network

```
const network = await docker.getNetworkByName('old-network');
await network.remove();
```

Network Properties

Property	Type	Description
`Id`	string	Network ID
`Name`	string	Network name
`Created`	string	Creation timestamp
`Scope`	string	Network scope (local, swarm, global)
`Driver`	string	Network driver
`EnableIPv6`	boolean	Whether IPv6 is enabled
`Internal`	boolean	Whether the network is internal-only
`Attachable`	boolean	Whether non-service containers can attach
`Ingress`	boolean	Whether the network is an ingress network
`IPAM`	object	IP Address Management configuration

⚙ Services (Swarm Mode)

Services are only available when the Docker daemon is running in Swarm mode.

Listing Services

```
const services = await docker.listServices();

for (const service of services) {
  console.log(`${service.Spec.Name} - ${service.Spec.TaskTemplate.ContainerSpec.Image}`);
  console.log(`  ID: ${service.ID}`);
}
```

Getting a Service by Name

```
const service = await docker.getServiceByName('my-web-service');
console.log(`Service: ${service.Spec.Name}`);
console.log(`Image: ${service.Spec.TaskTemplate.ContainerSpec.Image}`);
```

Creating a Service

Services support both string references and resource instances for images, networks, and secrets.

```
// Using string references
const service = await docker.createService({
  name: 'web-app',
  image: 'nginx:latest',
  labels: { environment: 'production', team: 'platform' },
  networks: ['frontend-network'],
  networkAlias: 'web',
  secrets: ['tls-certificate'],
  ports: ['80:80', '443:443'],
  resources: {
    memorySizeMB: 512,
  },
});

// Using resource instances directly
const image = await docker.getImageByName('nginx:latest');
const network = await docker.getNetworkByName('frontend-network');
const secret = await docker.getSecretByName('tls-certificate');

const service = await docker.createService({
  name: 'web-app',
  image: image,
  labels: { environment: 'production' },
  networks: [network],
  networkAlias: 'web',
  secrets: [secret],
  ports: ['80:80'],
  accessHostDockerSock: false,
  resources: {
    memorySizeMB: 1024,
    volumeMounts: [
```

```
      {
        containerFsPath: '/data',
        hostFsPath: '/mnt/storage/data',
      },
    ],
  },
});
```

Service creation descriptor fields:

Field	Type	Description
`name`	string	Service name
`image`	string | DockerImage	Image tag string or DockerImage instance
`labels`	object	Key-value label pairs
`networks`	(string | DockerNetwork)[]	Network names or DockerNetwork instances
`networkAlias`	string	DNS alias for the service on the network
`secrets`	(string | DockerSecret)[]	Secret names or DockerSecret instances
`ports`	string[]	Port mappings in `"hostPort:containerPort"` format
`accessHostDockerSock`	boolean	Mount the Docker socket inside the service container
`resources.memorySizeMB`	number	Memory limit in megabytes (default: 1000)
`resources.volumeMounts`	array	Array of `{ containerFsPath, hostFsPath }` mounts

Checking if a Service Needs an Update

```
const service = await docker.getServiceByName('web-app');
const needsUpdate = await service.needsUpdate();

if (needsUpdate) {
  console.log('A newer image version is available');
}
```

Removing a Service

```
const service = await docker.getServiceByName('old-service');
await service.remove();
```

Service Properties

Property	Type	Description
`ID`	string	Service ID
`Version`	object	Version info with `Index` number
`CreatedAt`	string	Creation timestamp
`UpdatedAt`	string	Last update timestamp
`Spec`	object	Full service specification
`Endpoint`	object	Endpoint specification and VirtualIPs

□□ Secrets (Swarm Mode)

Secrets are only available when the Docker daemon is running in Swarm mode.

Listing Secrets

```
const secrets = await docker.listSecrets();

for (const secret of secrets) {
  console.log(`${secret.Spec.Name} (ID: ${secret.ID})`);
}
```

Getting a Secret

```
// By name
const secret = await docker.getSecretByName('my-api-key');
```

```
// By ID
const secret = await docker.getSecretById('abc123secretid');
```

Creating a Secret

```
const secret = await docker.createSecret({
  name: 'database-credentials',
  version: '1.0.0',
  contentArg: JSON.stringify({
    host: 'db.example.com',
    username: 'admin',
    password: 'secret-password',
  }),
  labels: { environment: 'production', team: 'backend' },
});

console.log(`Secret created: ${secret.Spec.Name} (ID: ${secret.ID})`);
```

Field	Type	Description
`name`	string	Secret name
`version`	string	Version label for the secret
`contentArg`	any	Secret content (will be base64-encoded)
`labels`	object	Key-value label pairs

Updating a Secret

```
const secret = await docker.getSecretByName('database-credentials');
await secret.update(JSON.stringify({
  host: 'new-db.example.com',
  username: 'admin',
  password: 'new-secret-password',
}));
```

Removing a Secret

```
const secret = await docker.getSecretByName('old-secret');
await secret.remove();
```

Secret Properties

Property	Type	Description
`ID`	string	Secret ID
`Spec`	object	Contains `Name` and `Labels`
`Version`	object	Version info with `Index` string

□□ Image Store

Built-in image storage for caching Docker images locally or in S3-compatible object storage. Useful for backup, environment transfer, or air-gapped deployments.

Storing an Image

```
const image = await docker.getImageByName('myapp:latest');
const tarStream = await image.exportToTarStream();
await docker.storeImage('myapp:latest', tarStream);
```

Retrieving an Image

```
const tarStream = await docker.retrieveImage('myapp:latest');

// Import the retrieved image back into Docker
const image = await docker.createImageFromTarStream(tarStream, {
  imageUrl: 'myapp:latest',
});
```

S3 Storage Backend

Add S3-compatible object storage for longer-term image persistence.

```
await docker.addS3Storage({
  endpoint: 's3.amazonaws.com',
  accessKey: 'YOUR_ACCESS_KEY',
  accessSecret: 'YOUR_SECRET_KEY',
  bucketName: 'docker-image-backups',
  directoryPath: 'images',
});

// Now storeImage() persists images to S3
const image = await docker.getImageByName('myapp:v2.0.0');
const tarStream = await image.exportToTarStream();
await docker.storeImage('myapp:v2.0.0', tarStream);
```

□□ Complete DockerHost API Reference

Lifecycle and Daemon

Method	Signature	Description
`start`	`() => Promise<void>`	Initialize the host and image store
`stop`	`() => Promise<void>`	Shut down the host and image store
`ping`	`() => Promise<void>`	Ping the Docker daemon; throws if unavailable
`getVersion`	`() => Promise<VersionInfo>`	Get Docker daemon version information
`auth`	`(authData) => Promise<void>`	Authenticate against a Docker registry
`getAuthTokenFromDockerConfig`	`(registryUrl: string) => Promise<void>`	Authenticate using ~/.docker/config.json
`activateSwarm`	`(advertiseIp?: string) => Promise<void>`	Initialize Docker Swarm mode
`getEventObservable`	`() => Promise<Observable<any>>`	Subscribe to real-time Docker events

Containers

Method	Signature	Description
`listContainers`	`() => Promise<DockerContainer[]>`	List all containers
`getContainerById`	`(id: string) => Promise<DockerContainer | undefined>`	Get a container by ID
`createContainer`	`(descriptor) => Promise<DockerContainer>`	Create a new container

Images

Method	Signature	Description
`listImages`	`() => Promise<DockerImage[]>`	List all images
`getImageByName`	`(name: string) => Promise<DockerImage | undefined>`	Get an image by tag name
`createImageFromRegistry`	`(descriptor) => Promise<DockerImage>`	Pull an image from a registry
`createImageFromTarStream`	`(stream, descriptor) => Promise<DockerImage>`	Import an image from a tar stream
`pruneImages`	`(options?) => Promise<PruneResult>`	Remove unused images

Networks

Method	Signature	Description
`listNetworks`	`() => Promise<DockerNetwork[]>`	List all networks
`getNetworkByName`	`(name: string) => Promise<DockerNetwork | undefined>`	Get a network by name
`createNetwork`	`(descriptor) => Promise<DockerNetwork>`	Create a new network

Services (Swarm)

Method	Signature	Description
`listServices`	`() => Promise<DockerService[]>`	List all services
`getServiceByName`	`(name: string) => Promise<DockerService>`	Get a service by name
`createService`	`(descriptor) => Promise<DockerService>`	Create a new service

Secrets (Swarm)

Method	Signature	Description
`listSecrets`	`() => Promise<DockerSecret[]>`	List all secrets
`getSecretByName`	`(name: string) => Promise<DockerSecret | undefined>`	Get a secret by name
`getSecretById`	`(id: string) => Promise<DockerSecret | undefined>`	Get a secret by ID
`createSecret`	`(descriptor) => Promise<DockerSecret>`	Create a new secret

Image Store

Method	Signature	Description
`storeImage`	`(name: string, tarStream: Readable) => Promise<void>`	Store an image tar stream
`retrieveImage`	`(name: string) => Promise<Readable>`	Retrieve a stored image as a tar stream
`addS3Storage`	`(options) => Promise<void>`	Configure S3 backend for image storage

□□ Complete Example

A full workflow: connect to Docker, pull an image, create a network and service, then clean up.

```
import { DockerHost } from '@apiclient.xyz/docker';

async function main() {
  // Connect to Docker
  const docker = new DockerHost({});
  await docker.start();

  // Check Docker availability
  await docker.ping();
  const version = await docker.getVersion();
  console.log(`Connected to Docker ${version.Version}`);

  // Initialize Swarm (if not already active)
  await docker.activateSwarm();
```

```
// Pull an image
const image = await docker.createImageFromRegistry({
  imageUrl: 'nginx',
  imageTag: 'latest',
});
console.log(`Pulled image: ${image.RepoTags[0]}`);

// Create a network
const network = await docker.createNetwork({
  Name: 'web-network',
  Driver: 'overlay',
  Attachable: true,
});
console.log(`Created network: ${network.Name}`);

// Create a secret
const secret = await docker.createSecret({
  name: 'web-config',
  version: '1.0.0',
  contentArg: JSON.stringify({ port: 8080 }),
  labels: {},
});
console.log(`Created secret: ${secret.Spec.Name}`);

// Create a service
const service = await docker.createService({
  name: 'web-server',
  image: image,
  labels: { app: 'web' },
  networks: [network],
  networkAlias: 'web',
  secrets: [secret],
  ports: ['8080:80'],
  resources: {
    memorySizeMB: 256,
  },
});
console.log(`Created service: ${service.Spec.Name}`);
```

```
  // List running containers
  const containers = await docker.listContainers();
  for (const container of containers) {
    console.log(`Container: ${container.Names[0]} - ${container.State}`);
  }

  // Clean up
  await service.remove();
  await secret.remove();
  await network.remove();

  // Prune unused images
  const pruneResult = await docker.pruneImages({ dangling: true });
  console.log(`Reclaimed ${(pruneResult.SpaceReclaimed / 1024 / 1024).toFixed(2)} MB`);

  await docker.stop();
}

main().catch(console.error);
```

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 @apiclient.xyz/docker

2026-03-28 - 5.1.2 - fix(deps)

upgrade core tooling dependencies and adapt Docker client internals for compatibility

- replace removed smartfile filesystem APIs with node:fs and SmartFileFactory usage
- update imagestore archive handling for smartarchive v5 and smartbucket v4 overwrite behavior
- improve Docker resource creation and stream handling with stricter null checks, cleanup, and timeout safeguards
- adjust tests and runtime behavior for Deno and newer dependency constraints

2026-03-16 - 5.1.1 - fix(paths)

use the system temp directory for nogit storage and add release metadata

- Changes nogitDir to resolve under the OS temporary directory instead of a local .nogit folder
- Adds @git.zone/cli release and module metadata to npmextra.json for npm publishing configuration

2025-11-25 - 5.1.0 - feat(host)

Add DockerHost version & image-prune APIs, extend network creation options, return exec inspect info, and improve image import/store and streaming

- Add DockerHost.getVersion() to retrieve Docker daemon version and build info
- Add DockerHost.pruneImages() with dangling and filters support (calls /images/prune)
- Extend INetworkCreationDescriptor and DockerNetwork._create() to accept Driver, IPAM, EnableIPv6, Attachable and Labels

- Enhance DockerContainer.exec() to return an inspect() helper and introduce IExecInspectInfo to expose exec state and exit code
- Improve DockerImage._createFromTarStream() parsing of docker-load output and error messages when loaded image cannot be determined
- Implement DockerImageStore.storeImage() to persist, repackage and upload images (local processing and s3 support)
- Various streaming/request improvements for compatibility with Node streams and better handling of streaming endpoints
- Update tests to cover new features (network creation, exec inspect, etc.)

2025-11-24 - 5.0.2 - fix(DockerContainer)

Fix getContainerById to return undefined for non-existent containers

- Prevented creation of an invalid DockerContainer from Docker API error responses when a container does not exist.
- Changed DockerContainer._fromId to use the list+find pattern and return Promise<DockerContainer | undefined>.
- Updated DockerHost.getContainerById to return Promise<DockerContainer | undefined> for type safety and consistent behavior.
- Added tests to verify undefined is returned for non-existent container IDs and that valid IDs return DockerContainer instances.
- Bumped package version to 5.0.1 and updated changelog and readme hints to document the fix.

2025-11-24 - 5.0.0 - BREAKING CHANGE(DockerHost)

Rename array-returning get* methods to list* on DockerHost and related resource classes; update docs, tests and changelog

- Renamed public DockerHost methods: getContainers → listContainers, getNetworks → listNetworks, getServices → listServices, getImages → listImages, getSecrets → listSecrets.
- Renamed DockerNetwork.getContainersOnNetwork → DockerNetwork.listContainersOnNetwork and updated usages (e.g. getContainersOnNetworkForService).

- Updated internal/static method docs/comments to recommend dockerHost.list*() usage and adjusted implementations accordingly.
- Updated README, readme.hints.md, tests (test.nonci.node+deno.ts) and changelog to reflect the new list* method names.
- Bumped package version to 4.0.0.
- Migration note: replace calls to get*() with list*() for methods that return multiple items (arrays). Single-item getters such as getContainerById or getNetworkByName remain unchanged.

2025-11-24 - 5.0.1 - fix(DockerContainer)

Fix getContainerById() to return undefined instead of invalid container object when container doesn't exist

Bug Fixed:

- `getContainerById()` was creating a DockerContainer object from error responses when a container didn't exist
- The error object `{ message: "No such container: ..." }` was being passed to the constructor
- Calling `.logs()` on this invalid container returned "[object Object]" instead of logs

Solution:

- Changed `DockerContainer._fromId()` to use the list+filter pattern (consistent with all other resource getters)
- Now returns `undefined` when container is not found (matches DockerImage, DockerNetwork, DockerService, DockerSecret behavior)
- Updated return type to `Promise<DockerContainer | undefined>` for type safety
- Added tests to verify undefined is returned for non-existent containers

Migration: No breaking changes - users should already be checking for undefined/null based on TypeScript types and documentation.

2025-11-24 - 4.0.0 - BREAKING CHANGE: Rename list methods for

consistency

Breaking Changes:

- Renamed all "get*" methods that return arrays to "list*" methods for better clarity:
  - `getContainers()` → `listContainers()`
  - `getNetworks()` → `listNetworks()`
  - `getServices()` → `listServices()`
  - `getImages()` → `listImages()`
  - `getSecrets()` → `listSecrets()`
  - `getContainersOnNetwork()` → `listContainersOnNetwork()` (on DockerNetwork class)

Migration Guide: Update all method calls from `get*()` to `list*()` where the method returns an array of resources. Single-item getters like `getContainerById()`, `getNetworkByName()`, etc. remain unchanged.

Rationale: The `list*` naming convention more clearly indicates that these methods return multiple items (arrays), while `get*` methods are reserved for retrieving single items by ID or name. This follows standard API design patterns and improves code readability.

2025-11-24 - 3.0.2 - fix(readme)

Update README to document 3.0.0+ changes: architecture refactor, streaming improvements, health check and circular dependency fixes

- Documented major refactor to a Clean OOP / Facade pattern with DockerHost as the single entry point
- Added/clarified real-time container streaming APIs: streamLogs(), attach(), exec()
- Clarified support for flexible descriptors (accept both string references and class instances)
- Documented complete container lifecycle API (start, stop, remove, logs, inspect, stats)
- Documented new ping() health check method to verify Docker daemon availability
- Noted fix for circular dependency issues in Node.js by using type-only imports
- Mentioned improved TypeScript definitions and expanded examples, migration guides, and real-world use cases

2025-11-24 - 3.0.1 - fix(classes.base)

Use type-only import for DockerHost in classes.base to avoid runtime side-effects

- Changed the import in ts/classes.base.ts to a type-only import: import type { DockerHost } from './classes.host.js';
- Prevents a runtime import of classes.host when only the type is needed, reducing risk of circular dependencies and unintended side-effects during module initialization.
- No behavior changes to the public API — TypeScript-only change; intended to improve bundling and runtime stability.

2025-11-24 - 3.0.0 - BREAKING CHANGE(DockerHost)

Refactor public API to DockerHost facade; introduce DockerResource base; make resource static methods internal; support flexible descriptors and stream compatibility

- Refactored architecture: DockerHost is now the single public entry point (Facade) for all operations; direct static calls like DockerImage.createFromRegistry(...) are now internal and replaced by DockerHost.createImageFromRegistry(...) and similar factory methods.
- Introduced DockerResource abstract base class used by all resource classes (DockerContainer, DockerImage, DockerNetwork, DockerSecret, DockerService) with a required refresh() method and standardized dockerHost property.
- Static methods on resource classes were renamed / scoped as internal (prefixed with _): _list, _fromName/_fromId, _create, _createFromRegistry, _createFromTarStream, _build, etc. Consumers should call DockerHost methods instead.
- Creation descriptor interfaces (container, service, etc.) now accept either string identifiers or resource instances (e.g. image: string | DockerImage, networks: (string | DockerNetwork)[], secrets: (string | DockerSecret)[]). DockerHost resolves instances internally.
- DockerImageStore imageStore has been made private on DockerHost; new public methods DockerHost.storeImage(name, stream) and DockerHost.retrieveImage(name) provide access to the image store.
- Streaming compatibility: updated requestStreaming to convert web ReadableStreams (smartrequest v5+) to Node.js streams via smartstream.nodewebhelpers, preserving backward compatibility for existing streaming APIs (container logs, attach, exec, image import/export, events).
- Container enhancements: added full lifecycle and streaming/interactive APIs on DockerContainer: refresh(), inspect(), start(), stop(), remove(), logs(), stats(), streamLogs(), attach(), exec().
- Service creation updated: resolves image/network/secret descriptors (strings or instances); adds labels.version from image; improved resource handling and port/secret/network resolution.

- Network and Secret classes updated to extend DockerResource and to expose refresh(), remove() and lookup methods via DockerHost (createNetwork/listNetworks/getNetworkByName, createSecret/listSecrets/getSecretByName/getSecretById).
- Tests and docs updated: migration guide and examples added (readme.hints.md, README); test timeout reduced from 600s to 300s in package.json.
- BREAKING: Public API changes require consumers to migrate away from direct resource static calls and direct imageStore access to the new DockerHost-based factory methods and storeImage/retrieveImage APIs.

2025-11-18 - 2.1.0 - feat(DockerHost)

Add DockerHost.ping() to check Docker daemon availability and document health-check usage

- Add DockerHost.ping() method that issues a GET to /_ping and throws an error if the response status is not 200
- Update README: show ping() in Quick Start, add health check examples (isDockerHealthy, waitForDocker) and mention Health Checks in Key Concepts

2025-11-18 - 2.0.0 - BREAKING CHANGE(DockerHost)

Rename DockerHost constructor option 'dockerSockPath' to 'socketPath' and update internal socket path handling

- Breaking: constructor option renamed from 'dockerSockPath' to 'socketPath' — callers must update their code.
- Constructor now reads the provided 'socketPath' option first, then falls back to DOCKER_HOST, CI, and finally the default unix socket.
- README examples and documentation updated to use 'socketPath'.

2025-11-17 - 1.3.6 - fix(streaming)

Convert smartrequest v5 web ReadableStreams to Node.js streams and update deps for streaming compatibility

- Upgrade @push.rocks/smartrequest to ^5.0.1 and bump @git.zone dev tooling (@git.zone/tsbuild, tsrun, tstest).
- requestStreaming now uses response.stream() (web ReadableStream) and converts it to a Node.js Readable via plugins.smartstream.nodewebhelpers.convertWebReadableToNodeReadable for backward compatibility.
- Updated consumers of streaming responses (DockerHost.getEventObservable, DockerImage.createFromTarStream, DockerImage.exportToTarStream) to work with the converted Node.js stream and preserve event/backpressure semantics (.on, .pause, .resume).
- Added readme.hints.md documenting the smartrequest v5 migration, conversion approach, modified files, and test/build status (type errors resolved and Node.js tests passing).
- Removed project metadata file (.serena/project.yml) from the repository.

2025-08-19 - 1.3.5 - fix(core)

Stabilize CI/workflows and runtime: update CI images/metadata, improve streaming requests and image handling, and fix tests & package metadata

- Update CI workflows and images: switch workflow IMAGE to code.foss.global/host.today/ht-docker-node:npmci, fix NPMCI_COMPUTED_REPOURL placeholders, and replace @shipzone/npmci with @ship.zone/npmci in workflows
- Update npmextra.json gitzone metadata (githost -> code.foss.global, gitscope -> apiclient.xyz, npmPackagename -> @apiclient.xyz/docker) and npmdocker.baseImage -> host.today/ht-docker-node:npmci
- Adjust package.json repository/bugs/homepage to code.foss.global, add pnpm overrides entry and normalize package metadata
- Improve DockerHost streaming and request handling: reduce requestStreaming timeout to 30s, enable autoDrain for streaming requests, improve response parsing for streaming vs JSON endpoints to avoid hangs
- Enhance DockerImage and DockerImageStore stream handling and tar processing: more robust import/export parsing, safer stream-to-file writes, repackaging steps, and error handling
- Unskip and update tests: re-enable DockerImageStore integration test, change stored image name to 'hello2', add formatting fixes and ensure cleanup stops the test DockerHost
- Miscellaneous code and docs cleanup: numerous formatting fixes and trailing-comma normalization across README and TS sources, update commitinfo and logger newline fixes, and add local tool ignores (.claude/.serena) to .gitignore

2025-08-19 - 1.3.4 - fix(test)

Increase test timeout, enable DockerImageStore test, update test image name, bump smartrequest patch, and add local claude settings

- Increase tstest timeout from 120s to 600s in package.json to accommodate longer-running integration tests.
- Unskip the DockerImageStore integration test and change stored image name from 'hello' to 'hello2' in test/test.nonci.node.ts.
- Bump dependency @push.rocks/smartrequest from ^4.3.0 to ^4.3.1.
- Add .claude/settings.local.json to allow local agent permissions for running tests and related tooling.

2025-08-19 - 1.3.3 - fix(classes.host)

Adjust requestStreaming timeout and autoDrain; stabilize tests

- Reduced requestStreaming timeout from 10 minutes to 30 seconds to avoid long-running hanging requests.
- Enabled autoDrain for streaming requests to ensure response streams are properly drained and reduce resource issues.
- Marked the DockerImageStore S3 integration test as skipped to avoid CI dependence on external S3 and added a cleanup test to stop the test DockerHost.
- Added local tool settings file (.claude/settings.local.json) with local permissions (development-only).

2025-08-18 - 1.3.2 - fix(package.json)

Fix test script timeout typo, update dependency versions, and add typings & project configs

- Fix test script: correct 'tineout' -> 'timeout' for npm test command and set timeout to 120s
- Add 'typings': 'dist_ts/index.d.ts' to package.json

- Bump dependencies to newer compatible versions (notable packages: @push.rocks/lik, @push.rocks/smartarchive, @push.rocks/smartbucket, @push.rocks/smartfile, @push.rocks/smartlog, @push.rocks/smartpromise, @push.rocks/smartstream, rxjs)
- Add project/config files: .serena/project.yml and .claude/settings.local.json (editor/CI metadata)
- Include generated cache/metadata files (typescript document symbols cache) — not source changes but tooling/cache artifacts

2025-08-18 - 1.3.1 - fix(test)

Update test setup and devDependencies; adjust test import and add package metadata

- Update test script to run with additional flags: --verbose, --logfile and --tineout 120
- Bump devDependencies: @git.zone/tsbuild -> ^2.6.7, @git.zone/tsrun -> ^1.3.3, @git.zone/tstest -> ^2.3.5, @push.rocks/qenv -> ^6.1.3
- Change test import from @push.rocks/tapbundle to @git.zone/tstest/tapbundle
- Add typings field (dist_ts/index.d.ts)
- Add packageManager field for pnpm@10.14.0 with integrity hash

2024-12-23 - 1.3.0 - feat(core)

Initial release of Docker client with TypeScript support

- Provides easy communication with Docker's remote API from Node.js
- Includes implementations for managing Docker services, networks, secrets, containers, and images

2024-12-23 - 1.2.8 - fix(core)

Improved the image creation process from tar stream in DockerImage class.

- Enhanced `DockerImage.createFromTarStream` method to handle streamed response and parse imported image details.
- Fixed the dependency version for `@push.rocks/smartarchive` in package.json.

2024-10-13 - 1.2.7 - fix(core)

Prepare patch release with minor fixes and improvements

2024-10-13 - 1.2.6 - fix(core)

Minor refactoring and code quality improvements.

2024-10-13 - 1.2.5 - fix(dependencies)

Update dependencies for stability improvements

- Updated @push.rocks/smartstream to version ^3.0.46
- Updated @push.rocks/tapbundle to version ^5.3.0
- Updated @types/node to version 22.7.5

2024-10-13 - 1.2.4 - fix(core)

Refactored DockerImageStore constructor to remove DockerHost dependency

- Adjusted DockerImageStore constructor to remove dependency on DockerHost
- Updated ts/classes.host.ts to align with DockerImageStore's new constructor signature

2024-08-21 - 1.2.3 - fix(dependencies)

Update dependencies to the latest versions and fix image export test

- Updated several dependencies to their latest versions in package.json.
- Enabled the previously skipped 'should export images' test.

2024-06-10 - 1.2.1-1.2.2 - Core/General

General updates and fixes.

- Fix core update

2024-06-10 - 1.2.0 - Core

Core updates and bug fixes.

- Fix core update

2024-06-08 - 1.2.0 - General/Core

Major release with core enhancements.

- Processing images with extraction, retagging, repackaging, and long-term storage

2024-06-06 - 1.1.4 - General/Imagestore

Significant feature addition.

- Add feature to process images with extraction, retagging, repackaging, and long-term storage

2024-05-08 - 1.0.112 - Images

Add new functionality for image handling.

- Can now import and export images
- Start work on local 100% JS OCI image registry

2024-06-05 - 1.1.0-1.1.3 - Core

Regular updates and fixes.

- Fix core update

2024-02-02 - 1.0.105-1.0.110 - Core

Routine core updates and fixes.

- Fix core update

2022-10-17 - 1.0.103-1.0.104 - Core

Routine core updates.

- Fix core update

2020-10-01 - 1.0.99-1.0.102 - Core

Routine core updates.

- Fix core update

2019-09-22 - 1.0.73-1.0.78 - Core

Routine updates and core fixes.

- Fix core update

2019-09-13 - 1.0.60-1.0.72 - Core

Routine updates and core fixes.

- Fix core update

2019-08-16 - 1.0.43-1.0.59 - Core

Routine updates and core fixes.

- Fix core update

2019-08-15 - 1.0.37-1.0.42 - Core

Routine updates and core fixes.

- Fix core update

2019-08-14 - 1.0.31-1.0.36 - Core

Routine updates and core fixes.

- Fix core update

2019-01-10 - 1.0.27-1.0.30 - Core

Routine updates and core fixes.

- Fix core update

2018-07-16 - 1.0.23-1.0.24 - Core

Routine updates and core fixes.

- Fix core shift to new style

2017-07-16 - 1.0.20-1.0.22 - General

Routine updates and fixes.

- Update node_modules within npmdocker

2017-04-02 - 1.0.18-1.0.19 - General

Routine updates and fixes.

- Work with npmdocker and npmts 7.x.x
- CI updates

2016-07-31 - 1.0.17 - General

Enhancements and fixes.

- Now waiting for response to be stored before ending streaming request
- Cosmetic fix

2016-07-29 - 1.0.14-1.0.16 - General

Multiple updates and features added.

- Fix request for change observable and add npmdocker
- Add request typings

2016-07-28 - 1.0.13 - Core

Fixes and preparations.

- Fixed request for newer docker
- Prepare for npmdocker

2016-06-16 - 1.0.0-1.0.2 - General

Initial sequence of releases, significant feature additions and CI setups.

- Implement container start and stop
- Implement list containers and related functions
- Add tests with in docker environment

2016-04-12 - unknown - Initial Commit

Initial project setup.

- Initial commit