Authentication and authorization

- Sign in through your browser
- Authenticate automation with a PAT
- Authenticate agents
- Resource access and permissions
- Authenticate direct API requests
- Authenticate sandbox requests

Use browser sign-in when running an application interactively, or a personal access token (PAT) for automation. The SDK obtains short-lived access tokens and renews them as needed.

You need an active Docker Agentic Platform subscription. Authenticate with the Docker account you used to subscribe.

Sign in through your browser

Use the OAuth helper to sign in with your Docker account:

```
import { oauth, Sandboxes } from '@docker/sandboxes';

const auth = oauth({
  onVerification({ verificationUriComplete, verificationUri, userCode }) {
    console.log(`Open ${verificationUriComplete ?? verificationUri}`);
    console.log(`Verification code: ${userCode}`);
  },
});
await auth.getAccessToken();
const client = new Sandboxes({ auth });
```

Open the printed URL and complete sign-in. Browser sign-in supports single sign-on and two-factor authentication. The `getAccessToken()` call waits for you to finish before the program continues.

The SDK keeps credentials in memory and refreshes them while your application runs. With the default configuration, you sign in again each time you start the application. SDK sign-in is separate from `docker login` and `sbx login`.

Authenticate automation with a PAT

Use a personal access token for CI jobs and unattended applications. When creating the token, select **Allow Cloud Sandboxes access**. That grants the `sandbox:use` scope. The same token can also have a repository permission, for example to push images and use Cloud Sandboxes. Registry permissions alone don't grant Cloud Sandboxes access.

Provide your Docker ID and PAT to the SDK. For example, read them from your application's environment:

```
import { pat, Sandboxes } from '@docker/sandboxes';

const username = process.env.DOCKER_ID;
const personalAccessToken = process.env.DOCKER_PAT;
if (!username || !personalAccessToken) {
  throw new Error('Set DOCKER_ID and DOCKER_PAT');
}

const client = new Sandboxes({
  auth: pat({ username, personalAccessToken }),
});
```

The SDK exchanges the PAT for a short-lived access token and repeats the exchange when needed. If the PAT is invalid or revoked, authentication fails.

Store the PAT in your CI or application's secret store and keep it out of source control and logs.

Authenticate agents

An AI agent needs credentials for its model provider in addition to your Docker sign-in. For example, Claude Code can use an Anthropic API key, and Codex can use an OpenAI API key.

Store the provider key as a secret and attach it when creating the sandbox. For example, with an authenticated `client` and an Anthropic API key in `providerKey`:

```
const secret = await client.secrets.create({
  displayName: 'anthropic-key',
  serviceType: 'anthropic',
  token: { value: providerKey },
});

const sandbox = await client.kits.launchAndWait('claude', {
  storage: { secrets: [secret.name] },
});
```

The secret is attached before the agent runs. Keep the key out of command arguments, source files, and plain environment variables inside the sandbox.

Resource access and permissions

Your credentials determine which account's resources you can access. Cloud uses them to identify the account, so leave the optional `parent` field empty in requests.

Each request also checks whether you have permission for the action on the target resource. For example, creating a sandbox requires `sandboxesCreate`, reading it requires `sandboxesRead`, and deleting it requires `sandboxesDelete`.

Account permissions also control access to optional features. See Supported Cloud options.

Authenticate direct API requests

If you call the REST API without the SDK, obtain and renew access tokens in your application. Exchange your Docker ID and a PAT with the `sandbox:use` scope using the Docker Hub authentication API:

```
$ ACCESS_TOKEN=$(curl --silent --show-error --fail --request POST \
  --url https://hub.docker.com/v2/auth/token \
  --header "Content-Type: application/json" \
```

```
  --data '{"identifier":"<DOCKER_ID>","secret":"<PERSONAL_ACCESS_TOKEN>"}' \
  | jq -er '.access_token')
```

Send the returned access token in the `Authorization` header when calling `https://connect.docker.com/sandboxes`:

```
Authorization: Bearer <access_token>
```

The Sandboxes API accepts the access token returned by the exchange, not the PAT itself.

If you manage access tokens but use the SDK for requests, import `bearer` from `@docker/sandboxes` and configure the client with `auth: bearer(accessToken)`. Create a client with a fresh token when the previous token expires.

Authenticate sandbox requests

The SDK handles authentication when you run commands or transfer files using a sandbox's methods. It obtains a token scoped to that sandbox and the operation, then reuses or renews the token as needed.

For example, running a command requires `sandboxesExec` and obtaining its token requires `sandboxesCredential`. Your account must have both permissions.

If you call a sandbox endpoint directly, use a token issued for that sandbox. Don't send the Docker Hub token used for management requests to a sandbox endpoint. See Management and sandbox endpoints.

□□□ #2
□ □□□ □□□ 2026-10-31 18:54:41 UTC
□ □□□ □□□ 2026-10-31 18:54:43 UTC