

Develop and test locally



- [Build and test inside a sandbox](#)
 - [Local services](#)
 - [Accessing services in the sandbox](#)
 - [Accessing host services from a sandbox](#)
-

This page describes local sandboxes, host services, and local port mappings. For cloud endpoints, see [Expose a port](#).

Use a sandbox's private runtime to build images, run tests, and connect local tools to development services across the sandbox boundary.

Build and test inside a sandbox

Agents have sudo access inside the sandbox, so they can install packages, start databases, run test dependencies, and prepare the environment they need. Installed packages persist for the sandbox's lifetime. For repeated setup, use [Customize](#) to package the environment as a template or kit.

Agents can also build Docker images, run containers, and use [Compose](#). Everything runs inside the sandbox's private Docker daemon, so containers started by the agent never appear in your host's `docker ps`. When you remove the sandbox, all images, containers, and volumes inside it are deleted with it.

This pattern works well for tasks where the agent needs to run the project's test suite or inspect a service it started. If you need to reach that service from your host, publish the port when you create the sandbox, or publish it later with `sbx ports`.

Local services

Use this workflow when a sandboxed agent starts a dev server, or when the agent needs to call a service running on your host.

Accessing services in the sandbox

Sandboxes are [network-isolated](#) — your browser or local tools can't reach a server running inside one by default. A port mapping of `8080:3000` publishes sandbox port 3000 on host port 8080.

If you know which ports you need, publish them when you create the sandbox:

```
$ sbx run --publish 8080:3000 --name my-sandbox claude
```

For an existing sandbox, use `sbx ports` to forward traffic from your host. Publishing a port on a stopped local sandbox starts it first.

The common case: an agent has started a dev server or API, and you want to open it in your browser or run tests against it.

```
$ sbx ports my-sandbox --publish 8080:3000
$ open http://localhost:8080
```

To let the OS pick a free host port instead of choosing one yourself, specify only the sandbox port. Then use `sbx ports` to check which host port was assigned:

```
$ sbx ports my-sandbox --publish 3000
$ sbx ports my-sandbox
```

`sbx ls` shows active port mappings alongside each sandbox, and `sbx ports` lists them in detail:

```
$ sbx ls
```

SANDBOX	AGENT	STATUS	PORTS	WORKSPACE	CREATED
my-sandbox	claude	● running	127.0.0.1:8080->3000/tcp4	/home/user/proj	2m

To stop forwarding a port:

```
$ sbx ports my-sandbox --unpublish 8080:3000
```

For a service to be reachable, it must listen on all interfaces inside the sandbox, not only `127.0.0.1`. Bind it to `0.0.0.0` for IPv4 or `:::` for both IPv4 and IPv6. Most dev servers need a flag like `--host 0.0.0.0` to do this.

On the host, a published port binds IPv4 (`127.0.0.1`) unless you name another protocol, so `http://localhost:<port>/` reaches a service listening on IPv4 whichever address your resolver picks for `localhost`. Naming an explicit IPv6 host address, such as `--publish [::1]:8080:3000`, defaults the protocol to `tcp6` instead. To publish on both families use `--publish 8080:3000/tcp`, and for IPv6 alone `/tcp6`. Both of those require the sandboxed service to listen on IPv6 as well — bind it to

`[::]` — or a client arriving over `::1` has its connection accepted and then reset.

Published ports survive restarts: `sbx` re-publishes them when the sandbox or the daemon restarts. Explicit host ports are reused, while a port published with an OS-assigned host port, such as `--publish 3000`, gets a different host port on each start. Check `sbx ports my-sandbox` to find it. If an explicit host port is already in use at restart, the CLI or the dashboard prompts you to choose another. Removing the sandbox releases its ports.

When `sbx run` re-attaches to an existing sandbox, it ignores `--publish`. Use `sbx ports` to publish ports on that sandbox. To stop forwarding, `--unpublish 8080:3000` removes a single mapping, and `--unpublish 3000` removes every host port mapped to sandbox port 3000.

Accessing host services from a sandbox

Services running on your host are reachable from inside a sandbox using the hostname `host.docker.internal`. Use this instead of `127.0.0.1` or your machine's local network IP address, which are not reachable from inside the sandbox.

The sandbox proxy translates `host.docker.internal` to `localhost` before forwarding the request, so you must add the `localhost` address with the specific port to your network policy allowlist:

```
$ sbx policy allow network localhost:11434
```

Then use `host.docker.internal` in any configuration or request that points at the host service. For example, to verify connectivity from a sandbox shell:

```
$ curl http://host.docker.internal:11434
```

```
## #2
```

```
0 ## 2026-10-31 19:02:41 UTC
```

```
0 ## 2026-10-31 19:02:42 UTC
```