Use environment files in the cloud

- Prepare your environment
- Create the sandbox and add your project
  - Automate the initial upload
- Work with the agent
- Retrieve your work and clean up
  - Push changes to GitHub instead
  - Remove the environment
- Adapt an existing environment file
  - Remove local settings
  - Automate setup with host commands
  - Configure cloud credentials
  - Apply changes and recover interrupted creation

Use an environment file to share a cloud sandbox's agent, tools, environment variables, and resource limits with your project contributors. Run `sbx --cloud env` to apply that configuration on Docker-managed infrastructure.

A cloud environment has its own filesystem. It doesn't mount or synchronize directories from your machine, so leave `workspace` out of the file. Copy your project into the sandbox or clone it there, and copy the results back or push your commits before removing the sandbox. Credentials and network policy are also separate from your local sandbox configuration. Host lifecycle commands still run on your machine, with your privileges.

> `sbx env` is experimental. The command interface and file format may change.

Prepare your environment

Complete the cloud prerequisites, including activating cloud access and signing in with `sbx login`. Cloud compute charges apply.

This walkthrough uses Claude Code to work on a local project named `web-app`. Configure an Anthropic API key in the cloud secret store before creating the sandbox:

```
$ sbx --cloud secret set anthropic
```

The command prompts for the key. For other agents and authentication options, see Authenticate cloud agents. A credential configured for a local sandbox isn't available in the cloud.

If you want the agent to push changes to GitHub, configure a GitHub token before creating the sandbox:

```
$ sbx --cloud secret set github --sandbox cloud-web-app
```

The command prompts for a token. Use one with write access to your repository. The `--sandbox` value must match the environment's `name`. Cloud environments select their credentials at creation: adding a GitHub secret afterward won't make it available to an existing environment. If you decide to push later, copy your work out first, then recreate the environment with the secret.

Save the following as `cloud.sbxenv.yaml` in your project's root directory. You can keep the file elsewhere if you prefer: its location doesn't determine which files are available in the cloud sandbox.

```
schemaVersion: "1"
name: cloud-web-app
agent: claude

env:
  NODE_ENV: test

sandboxOptions:
  cpus: 2
  memory: 4g
```

This selects Claude Code, sets `NODE_ENV` for the sandbox, and requests two CPUs and 4 GiB of memory. Resource limits must match a cloud size. You can also add `kits` to install project tools.

Create the sandbox and add your project

From your project directory, review the plan and create the sandbox without attaching:

```
$ sbx --cloud env plan ./cloud.sbxenv.yaml
$ sbx --cloud env create --ttl 2h --on-timeout delete ./cloud.sbxenv.yaml
```

The plan shows the configuration and inherited cloud credentials. Creation asks you to approve the plan before applying it. This example explicitly deletes the sandbox after two hours, so retrieve your work before then. Without timeout flags, a cloud sandbox expires after one hour, and the service stops sandboxes that can be resumed and deletes the rest. See Configure expiration for other options.

Copy your project into the sandbox's workspace:

```
$ sbx --cloud cp . cloud-web-app:/home/agent/workspace/web-app
```

Use a destination that doesn't already exist for this first copy. Copying a directory to an existing directory nests it under its source name. Review the files you're uploading: `cp .` doesn't respect `.gitignore`. It includes hidden files such as `.env`, Git history, dependencies, and build output. For larger projects, prepare a clean staging directory containing only the files you want to upload, and copy that directory instead. Copying is a one-time transfer: later edits on either side aren't synchronized. Directory uploads skip symbolic links and empty directories and report what was skipped. Install dependencies inside the sandbox rather than copying a local dependency directory.

For a Git workflow, you can instead clone a repository inside the sandbox and push commits to your remote. Configure cloud credentials for private repository access and cloud network policy for the destinations your project needs. Copying includes uncommitted local work; cloning a specific commit gives contributors a reproducible starting point.

Automate the initial upload

To upload your project whenever you create a fresh sandbox, add a `postCreate` command to the environment file. This automates the initial upload as part of setup.

```
lifecycle:
  postCreate:
    - name: Upload project
```

```
      command: sbx --cloud cp . "$SBX_SANDBOX_NAME:/home/agent/workspace/web-app"
```

Here, `.` resolves to the directory containing the environment file. If you keep the file elsewhere or upload from a staging directory, set the command's `workdir` to the project or staging directory. The same file selection rules as the manual copy apply, including copying ignored files and skipping symbolic links and empty directories.

The command runs on your machine after sandbox creation and before `run` attaches to the agent. `SBX_SANDBOX_NAME` contains the sandbox name, including any `--name` override. Reconnecting skips `postCreate`, preserving the agent's work. Later local edits aren't uploaded automatically.

With this block in place, use `sbx --cloud env run ./cloud.sbxenv.yaml` to create the sandbox, upload the project, and attach to the agent. It replaces the manual upload step for future sandbox creation.

If the upload fails, the sandbox has already been created and remains available. Inspect it and finish transferring files manually, or remove it with `sbx --cloud env rm ./cloud.sbxenv.yaml` before trying a fresh creation.

See Automate setup with host commands for more about lifecycle commands and approval.

Work with the agent

Attach to Claude Code using the same environment file:

```
$ sbx --cloud env run ./cloud.sbxenv.yaml
```

For example, ask the agent:

```
Work in /home/agent/workspace/web-app. Find the test and lint commands in
package.json, install the dependencies, and run those checks. Fix any failures
and explain your changes. Ask before changing dependencies.
```

Use `run` again to reconnect to the existing sandbox. It creates a sandbox if one doesn't exist. Pass `--ttl` and `--on-timeout` only when creating a sandbox: `run` rejects those flags when attaching to an existing one.

Press `Ctrl+\` to detach while leaving the agent running. You can also run a command without an agent session, for example to inspect the working tree of a copied Git project:

```
$ sbx --cloud env exec ./cloud.sbxenv.yaml -- \
    git -C /home/agent/workspace/web-app diff --stat
```

For subsequent environment commands, keep the same machine, cloud endpoint, Docker account, and ordered environment file paths. The CLI records the cloud sandbox's identity locally to manage it across invocations.

Retrieve your work and clean up

After the agent finishes, copy the project back to a separate local directory so you can review it before updating your original files:

```
$ sbx --cloud cp cloud-web-app:/home/agent/workspace/web-app ../web-app-result
```

Use a destination that doesn't already exist.

Push changes to GitHub instead

If you configured the GitHub secret before creating the environment, you can retrieve changes through Git instead of copying files. This works with a repository cloned inside the sandbox or a copied Git repository whose `.git` directory contains the repository history. Use an HTTPS GitHub remote that your token can push to.

For example, ask the agent:

```
In /home/agent/workspace/web-app, review the diff, create a branch named
cloud-fixes, commit the changes for this task, and push that branch to origin.
```

Then, from your local checkout, fetch the branch and check it out for review:

```
$ git fetch origin
$ git switch --track origin/cloud-fixes
```

The commits are available on GitHub even after the sandbox is removed.

Remove the environment

When you've saved your work, remove the environment:

```
$ sbx --cloud env rm ./cloud.sbxenv.yaml
```

Removal asks for approval and deletes the sandbox and secrets provisioned by this environment. Inherited credentials, such as the Anthropic key configured earlier, remain. Global bindings remain

unless you pass `--prune-bindings`. Files stored only in the sandbox are lost on deletion.

For unattended commands, use `--force` with `create`, `run`, or `rm` to approve the plan for that invocation. Use `run --detach` when no interactive terminal is available.

Adapt an existing environment file

Cloud environments support agents, kits, environment variables, CPU and memory limits, credentials for supported providers, and host lifecycle commands. They support a subset of the environment file fields. Use the following checks before reusing a local file.

Remove local settings

Remove these declarations:

- `workspace`, `additionalWorkspaces`, and clone options
- `ports`, `registries`, and `mcp.servers`
- `sandboxOptions.template`, `profile`, `pullPolicy`, `skills`, `display`, `gpu`, `usb`, and `writableEnvFiles`

Unsupported settings are rejected before host commands or provisioning run. Pass the cloud file path explicitly to avoid loading local settings from `~/.sbxenv.yaml`. For services and integrations, use the cloud commands to expose ports and load MCP servers.

Automate setup with host commands

Lifecycle commands connect the cloud environment's lifecycle to work on your machine. They're useful when setup or cleanup depends on local files, tools, or credentials. Including those steps in the environment file makes the workflow repeatable, even when it spans your machine and the cloud sandbox.

Use `initialize` to prepare files before creation and `preRemove` to retrieve results when you run `sbx --cloud env rm`. `preRemove` doesn't run when the cloud service deletes an expired sandbox, and a failed command doesn't prevent removal. Save work you need to keep before removing the environment.

Host commands appear in the plan and require approval. See Lifecycle commands for phase timing, working directories, and approval options.

Configure cloud credentials

The plan shows inherited cloud credentials. Sandbox-scoped credentials override account defaults, and credentials declared in `secrets` override both. Use `snapshot: true` to resolve a host command or vault reference once and upload the value. Snapshots don't refresh. Dynamic secret sources and custom credential providers aren't supported. Credential bindings require cloud support for kit credentials and explicit approval of the kit's injection domains.

Apply changes and recover interrupted creation

Changes to secrets, bindings, kits, or resource limits require recreating the sandbox. Updated `env` values apply to subsequent sessions. Rejoining a running agent keeps that process's environment.

If creation is interrupted, retry the same command and unchanged declaration within 23 hours. Unresolved requests prevent removal. Follow the recovery message and retain its journal until the original requests are resolved.

□□□ #2
□ □□□ □□□ 2026-10-31 18:58:12 UTC
□ □□□ □□□ 2026-10-31 18:58:14 UTC