Rebuilding the container in a codespace

In this article

You can rebuild a container to apply configuration changes to the codespaces you are working in. From time to time, you may want to perform a full rebuild.

About rebuilding a container

When you work in a codespace, your development environment is a Docker container that runs on a virtual machine. If you make changes to your dev container configuration from within a codespace, and you want to apply those changes to the current codespace, you need to rebuild the container.

By default, when you rebuild a container, GitHub Codespaces will speed up the build process by reusing cached images from previous builds of the container. This is usually the quickest way to implement changes to your dev container configuration, for the following reasons.

- GitHub Codespaces can reuse images in your cache rather than repulling them from container registries.
- The parts of your dev container configuration that define how the container is built, such as dev container features and Dockerfile instructions, may have already been implemented in image layers in your cache, so you won't need to wait for these processes to run again. (However, commands in your configuration that run after the container is built, such as `onCreateCommand`, will run again.)

Occasionally, you may want to perform a full rebuild of your container. With a full rebuild, GitHub Codespaces cleans all Docker containers, images, and volumes from the cache, then rebuilds your container with newly pulled images. All the setup defined in your configuration will run again, generating new image layers. You may want to perform a full rebuild after many iterations of rebuilding your container with cached images, in situations such as the following.

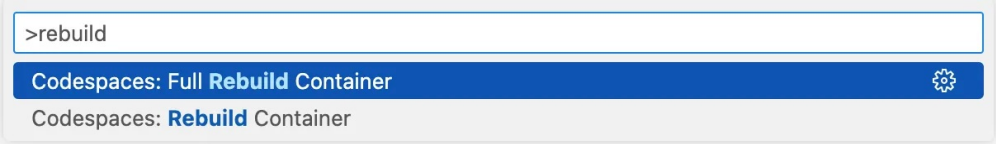
- You want to ensure that the setup defined in your configuration is not dependent on cached images, and will run as required when someone creates a new codespace based on the configuration. For example, a dependency may have been removed from the base image since it was last pulled into your codespace.
- You want to free up the disk space used by your cache, for example if you are low on disk space or want to minimize storage charges. Your image cache might be using a significant amount of disk space if you've changed your base image multiple times, if you've made a large number of iterative changes to your configuration, or if you're running multiple containers with Docker Compose.

Rebuilding a container

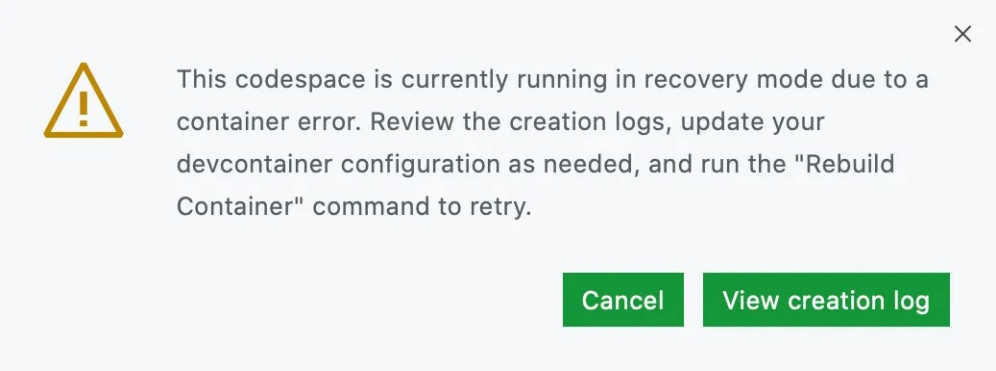
You can rebuild a container within a codespace in the VS Code web client or desktop application, or you can use GitHub CLI.

Rebuilding the dev container in the VS Code web client or desktop application

1. Access the VS Code Command Palette with `Shift` + `Command` + `P` (Mac) or `Ctrl` + `Shift` + `P` (Windows/Linux).
2. Start typing "Rebuild" and select **Codespaces: Rebuild Container** or **Codespaces: Full Rebuild Container**.



3. If changes to your dev container configuration cause a container error, your codespace will run in recovery mode, and you will see an error message.



   - To diagnose the error by reviewing the creation logs, click **View creation log**.
   - To fix the errors identified in the logs, update your `devcontainer.json` file.
   - To apply the changes, rebuild your container.

Using GitHub CLI to rebuild a dev container

If you've changed a dev container configuration outside of VS Code (for example, on GitHub.com or in a JetBrains IDE), you can use GitHub CLI to rebuild the dev container for an existing codespace.

1. In a terminal, enter the following command.

   ```
   gh codespace rebuild
   ```

   Your codespaces are listed.

2. Use the arrow keys on your keyboard to highlight the required codespace, then press `Enter`.

To perform a full rebuild with GitHub CLI, you can use the `gh codespace rebuild --full`

command.

Persisting data over a rebuild

When you create a codespace, your repository is cloned into the `/workspaces` directory in your codespace. This is a persistent directory that is mounted into the container. Any changes you make inside this directory, including editing, adding, or deleting files, are preserved when you stop and start the codespace, and when you rebuild the container in the codespace.

Outside the `/workspaces` directory, your codespace contains a Linux directory structure that varies depending on the image used to build your codespace. You can add files or make changes to files outside the `/workspaces` directory: for example, you can install new programs, or you can set up your shell configuration in a file such as `~/.bashrc`. As a non-root user, you may not automatically have write access to certain directories, but most images allow root access to these directories with the `sudo` command.

Outside `/workspaces`, with the exception of the `/tmp` directory, the directories in a codespace are tied to the lifecycle of the container. This means any changes you make are preserved when you stop and start your codespace, but are not preserved when you rebuild of the container.

If you want to preserve files outside the `/workspaces` directory over a rebuild, you can create, at the desired location in the container, a symbolic link (symlink) to the persistent directory. For example, in your `/workspaces/.devcontainer` directory, you can create a `config` directory that will be preserved across a rebuild. You can then symlink the `config` directory and its contents as a `postCreateCommand` in your `devcontainer.json` file.

```
{
    "image": "mcr.microsoft.com/vscode/devcontainers/base:alpine",
    "postCreateCommand": ".devcontainer/postCreate.sh"
}
```

In the example `postCreate.sh` file below, the contents of the `config` directory are symbolically linked to the home directory.

```
#!/bin/bash
ln -sf $PWD/.devcontainer/config $HOME/config && set +x
```

Further reading

- "Introduction to dev containers"