Deep dive into GitHub Codespaces

In this article

Understand how GitHub Codespaces works.

GitHub Codespaces is an instant, cloud-based development environment that uses a container to provide you with common languages, tools, and utilities for development. GitHub Codespaces is also configurable, allowing you to create a customized development environment for your project. By configuring a custom development environment for your project, you can have a repeatable codespace configuration for all users of your project.

Creating your codespace

There are a number of entry points to create a codespace.

- From a GitHub template or any template repository on GitHub.com to start a new project
- From a branch in your repository for new feature work
- From an open pull request to explore work-in-progress
- From a commit in a repository's history to investigate a bug at a specific point in time

You can create a codespace on GitHub.com, in Visual Studio Code, or by using GitHub CLI.

Your codespace can be ephemeral if you need to test something or you can return to the same codespace to work on long-running feature work.

For more information, see "Creating a codespace for a repository," "Creating a codespace from a template," and "Opening an existing codespace."

> **Note**: You can create more than one codespace per repository or even per branch. However, there are limits to the number of codespaces you can create, and the number of codespaces you can run at the same time. If you reach the maximum number of codespaces and try to create another, a message is displayed telling you that you must remove an existing codespace before you can create a new one.

The codespace creation process

When you create a codespace, various steps happen in the background before the

codespace is available to you.

Step 1: VM and storage are assigned to your codespace

When you create a codespace, a shallow clone is made of your repository, or of the template repository if you're creating a codespace from a template. The repository is cloned to a Linux virtual machine that is both dedicated and private to you. Having a dedicated VM ensures that you have the entire set of compute resources from that machine available to you. If necessary, this also allows you to have full root access to your container.

Your repository is cloned into the `/workspaces` directory in the codespace. For more information, see "About the directory structure of a codespace" below.

Step 2: Container is created

GitHub Codespaces uses a container as the development environment. This container is created based on configurations that you can define in a `devcontainer.json` file and, optionally, a Dockerfile. If you create a codespace from GitHub's blank template, or from a repository with no `devcontainer.json` file, GitHub Codespaces uses a default image, which has many languages and runtimes available. For more information, see "Introduction to dev containers." For details of what the default image contains, see the `devcontainers/images` repository.

> **Note:** If you want to use Git hooks in your codespace and apply anything in the git template directory to your codespace, then you must set up hooks during step 4 after the container is created.
>
> Since your repository is cloned onto the host VM before the container is created, anything in the git template directory will not apply in your codespace unless you set up hooks in your `devcontainer.json` configuration file using the `postCreateCommand` in step 4. For more information, see "Step 4: Post-creation setup."

Step 3: Connecting to the codespace

When your container has been created and any other initialization has run, you'll be connected to your codespace. You can connect to it by using:

- Your web browser
- Visual Studio Code
- A JetBrains IDE
- GitHub CLI

Step 4: Post-creation setup

Once you are connected to your codespace, your automated setup may continue to build based on the configuration specified in your `devcontainer.json` file. You may see `postCreateCommand` and `postAttachCommand` run.

If you want to use Git hooks in your codespace, set up hooks using the `devcontainer.json` lifecycle scripts, such as `postCreateCommand`. For information about the lifecycle scripts, see the dev containers specification on the Development Containers website.

If you have a public dotfiles repository for GitHub Codespaces, you can enable it for use with new codespaces. When enabled, your dotfiles will be cloned to the container and the install script will be invoked. For more information, see "Personalizing GitHub Codespaces for your account."

Finally, if you created the codespace from a repository, the entire history of the repository is copied down with a full clone. If you created the codespace from a template, the full history of the template repository is not preserved; instead, unless you are using the blank template, you will start with an initial commit for the contents of the template repository.

During post-creation setup you'll still be able to use the integrated terminal and make edits to your files, but take care to avoid any race conditions between your work and the commands that are running.

Codespaces lifecycle

Saving files in your codespace

Save changes to files in the normal way, depending on the editor you are using.

If you work on codespaces in Visual Studio Code, you can enable Auto Save to ensure that your changes are always saved.

Closing or stopping your codespace

Your codespace will keep running while you are using it, but will time out after a period of inactivity. File changes from the editor and terminal output are counted as activity, so your codespace will not time out if terminal output is continuing. The default inactivity timeout period is 30 minutes. You can define your personal timeout setting for codespaces you create, but this may be overruled by an organization timeout policy. For more information, see "Setting your timeout period for GitHub Codespaces."

If a codespace times out it will stop running, but you can restart it from the browser tab (if you were using the codespace in the browser), from within VS Code, or from your list of codespaces at https://github.com/codespaces.

To stop your codespace you can

- In the browser: on your list of codespaces at https://github.com/codespaces, click the ellipsis (**...**) to the right of the codespace you want to stop and click **Stop codespace**.
- In VS Code: open the Visual Studio Code Command Palette - for example, by pressing `Ctrl` + `Shift` + `Enter` (Windows/Linux) or `Shift` + `Command` + `P` (Mac) - type `Codespaces: stop` then press `Enter`. For more information, see "Using the Visual Studio Code Command Palette in GitHub Codespaces."
- In the JetBrains client, click the stop button at the top of the GitHub Codespaces tool window. For more information, see the "JetBrains IDEs" tab of "Stopping and starting a codespace."
- In a terminal window: use the GitHub CLI command `gh codespace stop`. For more information, see "Using GitHub Codespaces with GitHub CLI."

If you exit your codespace without running the stop command (for example, by closing the browser tab), or if you leave the codespace running without interaction, the codespace and its running processes will continue for the duration of the inactivity timeout period.

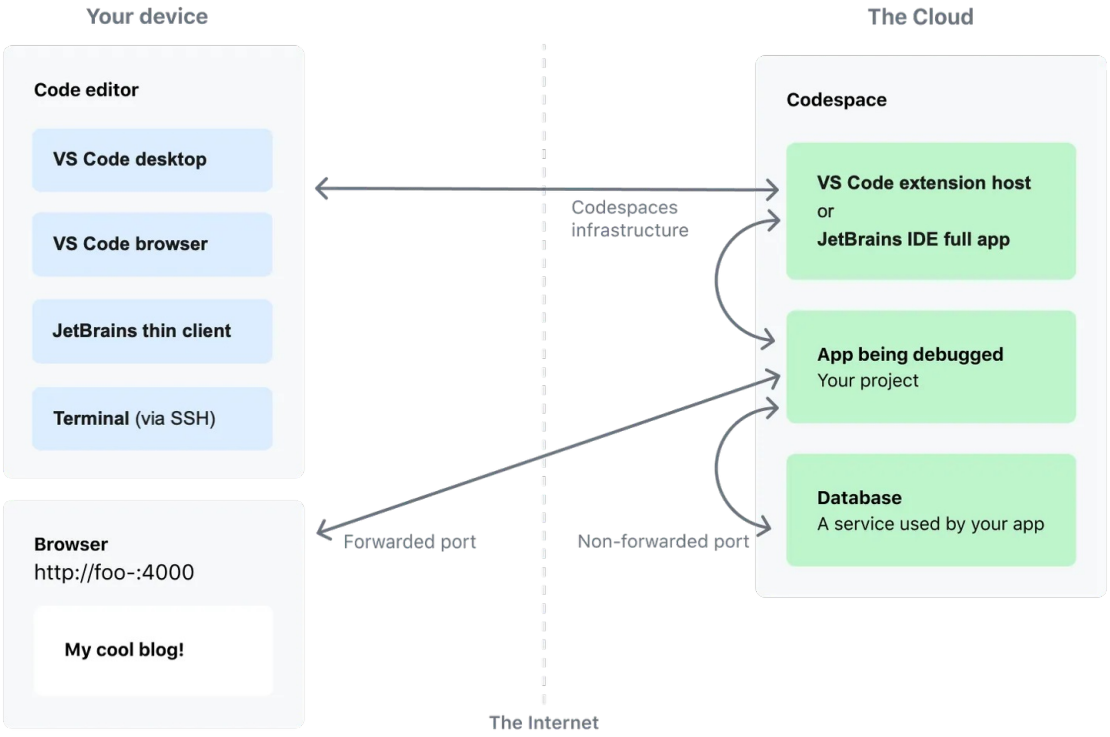
When you close or stop your codespace, all uncommitted changes are preserved until you connect to the codespace again.

Running your application

Port forwarding gives you access to TCP ports running within your codespace. For

example, if you're running a web application on port 4000 within your codespace, you can automatically forward that port to make the application accessible from your browser.

Port forwarding determines which ports are made accessible to you from the remote machine. Even if you do not forward a port, that port is still accessible to other processes running inside the codespace itself.



When an application running inside GitHub Codespaces outputs a port to the console, GitHub Codespaces detects the localhost URL pattern and automatically forwards the port. You can click on the URL in the terminal, or the link in the "toast" notification message that pops up at the bottom right corner of VS Code, to open the port in a browser. By default, GitHub Codespaces forwards the port using HTTP. For more information on port forwarding, see "Forwarding ports in your codespace."

While ports can be forwarded automatically, they are not publicly accessible to the internet. By default, all ports are private, but you can manually make a port available to your organization or public, and then share access through a URL. For more information, see "Forwarding ports in your codespace."

Running your application when you first land in your codespace can make for a fast inner dev loop. As you edit, your changes are automatically saved and available on your forwarded port. To view changes, go back to the running application tab in your browser and refresh it.

Committing and pushing your changes

Git is installed by default in your codespace and so you can rely on your existing Git workflow. You can work with Git in your codespace either via the Terminal or by using the source control features of VS Code or JetBrains.

If you're working with an existing repository, you can create a codespace from any branch, commit, or pull request in the repository, or you can switch to a new or existing branch from within your active codespace. Because GitHub Codespaces is designed to be ephemeral, you can use it as an isolated environment to experiment, check a teammate's pull request, or fix merge conflicts.

If you only have read access to a repository, then you can create a codespace for the

repository as long as you can fork it. When you make a commit from the codespace, or push a new branch, GitHub Codespaces either automatically creates a fork of the repository for you, or links the codespace to an existing fork if you already have one for the upstream repository.

If you're working in a codespace created from a template, Git is installed by default, but you will need to publish your codespace to a remote repository to persist your work and to share it with others. If you start from GitHub's blank template, you first need to initialize your workspace as a Git repository (for example by entering `git init`) to start using source control within the codespace.

For more information, see "Using source control in your codespace."

Note: Commits from your codespace will be attributed to the name and public email configured at https://github.com/settings/profile. A token scoped to the repository, included in the environment as `GITHUB_TOKEN`, and your GitHub credentials will be used to authenticate.

Personalizing your codespace with extensions or plugins

You can add plugins and extensions within a codespace to personalize your experience in JetBrains and VS Code respectively.

VS Code extensions

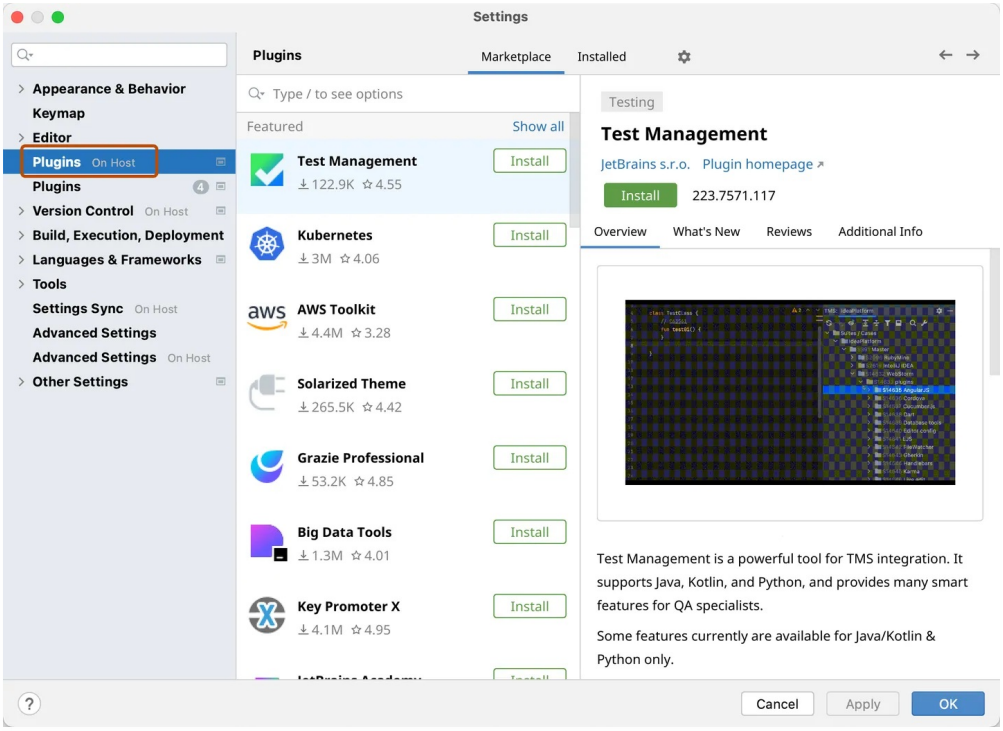
If you work on your codespaces in the VS Code desktop application, or the web client, you can add any extensions you need from the Visual Studio Code Marketplace. For information on how extensions run in GitHub Codespaces, see Supporting Remote Development and GitHub Codespaces in the VS Code documentation.

If you already use VS Code, you can use Settings Sync to automatically sync extensions, settings, themes, and keyboard shortcuts between your local instance and any codespaces you create.

JetBrains plugins

If you work on your codespaces in a JetBrains IDE, you can add plugins from the JetBrains Marketplace.

1. Click **JetBrains Client**, then click **Preferences**.
2. In the Preferences dialog, click either **Plugins On Host** to install a plugin in the full JetBrains IDE that's running remotely, or **Plugins** to install a plugin on the local client, for example to change the user interface theme.
3. Click the **Marketplace** tab.



4. Click **Install** beside the required plugin.

About the directory structure of a codespace

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. For more information on the `/tmp` directory, see "Persisting environment variables and temporary files."

Clearing the directories outside `/workspaces` helps to ensure the rebuilt container is in the same state as it would be in a newly created codespace. If you're rebuilding a container to apply configuration changes to the codespace you're working in, you can be confident that any configuration changes you have made will work the same for users creating new codespaces with the same configuration. For more information, see "Introduction to dev containers."

If you want to make changes to your codespace that will be more robust over rebuilds and across different codespaces, you have several options.

- To install programs and tools in all codespaces created from a repository, in your dev container configuration, you can use lifecycle command properties such as `postCreateCommand` to run custom installation commands, or you can choose from pre-written installation commands called "features." For more information, see the

dev containers specification on the Development Containers website and "Adding features to a devcontainer.json file."

- To install tools or customize your setup in every codespace you create, such as configuring your `bash` profile, you can link GitHub Codespaces with a dotfiles repository. The dotfiles repository is also cloned into the persistent `/workspaces` directory. For more information, see "Personalizing GitHub Codespaces for your account."
- If you want to preserve specific files over a rebuild, you can use a `devcontainer.json` file to create a symlink between the files and a persistent directory within `/workspaces`. For more information, see "Rebuilding the container in a codespace."

Further reading

- "Enabling or disabling GitHub Codespaces for your organization"
- "Managing the cost of GitHub Codespaces in your organization"
- "Adding a dev container configuration to your repository"
- "Understanding the codespace lifecycle"