Introduction to dev containers - GitHub Docs

18-22 minutes

When you work in a codespace, the environment you are working in is created using a development container, or dev container, hosted on a virtual machine.

Who can use this feature?

People with write permissions to a repository can create or edit the codespace configuration.

About dev containers

Development containers, or dev containers, are Docker containers that are specifically configured to provide a fully featured development environment. Whenever you work in a codespace, you are using a dev container on a virtual machine.

You can configure the dev container for a repository so that codespaces created for that repository give you a tailored development environment, complete with all the tools and runtimes you need to work on a specific project. If you don't define a configuration in the repository then GitHub Codespaces uses a default configuration, which contains many of the common tools that your team might need for development with your project. See "Using the default dev container configuration."

The configuration files for a dev container are contained in a `.devcontainer` directory in your repository. You can use Visual Studio Code to add configuration files for you. You can choose from a selection of predefined configurations for various project types. You can use these without further configuration, or you can edit the configurations to refine the development environment they produce. See "Using a predefined dev container configuration."

Alternatively, you can add your own custom configuration files. See "Creating a custom dev container configuration."

You can define a single dev container configuration for a repository, different configurations for different branches, or multiple configurations. When multiple configurations are available, users can choose their preferred configuration when they create a codespace. This is particularly useful for large repositories that contain source code in different programming languages or for different projects. You can create a choice of configurations that allow different teams to work in a codespace that's set up appropriately for the work they are doing.

When you create a codespace from a template, you might start with one or more dev container configuration files in your workspace. To configure your environment further, you can add or remove settings from these files and rebuild the container to apply the changes to the codespace you're working in. If you publish your codespace to a repository on GitHub, then any codespaces created from that repository will share the configuration you've defined. See "Applying configuration changes to a codespace" and "Creating a codespace from a template."

devcontainer.json

The primary file in a dev container configuration is the `devcontainer.json` file. You can use this file to determine the environment of codespaces created for your repository. The contents of this file define a dev container that can include frameworks, tools, extensions, and port forwarding. The

`devcontainer.json` file usually contains a reference to a Dockerfile, which is typically located alongside the `devcontainer.json` file.

If you create a codespace from a repository without a `devcontainer.json` file, or if you start from GitHub's blank template, the default dev container configuration is used. See "Using the default dev container configuration."

The `devcontainer.json` file is usually located in the `.devcontainer` directory of your repository. Alternatively, you can locate it directly in the root of the repository, in which case the file name must begin with a period: `.devcontainer.json`.

If you want to have a choice of dev container configurations in your repository, any alternatives to the `.devcontainer/devcontainer.json` (or `.devcontainer.json`) file must be located in their own subdirectory at the path `.devcontainer/SUBDIRECTORY/devcontainer.json`. For example, you could have a choice of two configurations:

- `.devcontainer/database-dev/devcontainer.json`
- `.devcontainer/gui-dev/devcontainer.json`

When you have multiple `devcontainer.json` files in your repository, each codespace is created from only one of the configurations. Settings cannot be imported or inherited between `devcontainer.json` files. If a `devcontainer.json` file in a custom subdirectory has dependent files, such as the Dockerfile or scripts that are run by commands in the `devcontainer.json` file, it's recommended that you co-locate these files in the same subdirectory.

For information about how to choose your preferred dev container configuration when you create a codespace, see "Creating a codespace for a repository."

For information about the settings and properties that you can set in a `devcontainer.json` file, see the Specification on the Development Containers website.

How to use the devcontainer.json

It's useful to think of the `devcontainer.json` file as providing "customization" rather than "personalization." You should only include things that everyone working on your codebase needs as standard elements of the development environment, not things that are personal preferences. Things like linters are good to standardize on, and to require everyone to have installed, so they're good to include in your `devcontainer.json` file. Things like user interface decorators or themes are personal choices that should not be put in the `devcontainer.json` file.

You can personalize your codespaces by using dotfiles and Settings Sync. See "Personalizing GitHub Codespaces for your account."

Dockerfile

You can add a Dockerfile as part of your dev container configuration.

The Dockerfile is a text file that contains the instructions needed to create a Docker container image. This image is used to generate a development container each time someone creates a codespace using the `devcontainer.json` file that references this Dockerfile. The instructions in the Dockerfile typically begin by referencing a parent image on which the new image that will be created is based. This is followed by commands that are run during the image creation process, for example to install software packages.

The Dockerfile for a dev container is typically located in the `.devcontainer` folder, alongside the `devcontainer.json` in which it is referenced.

Note: As an alternative to using a Dockerfile you can use the `image` property in the `devcontainer.json` file to refer directly to an existing image you want to use. The image you specify here must be allowed by any organization image policy that has been set. See "Restricting the base image for codespaces." If neither a Dockerfile nor an image is found then the default container image is used. See "Using the default dev container configuration."

Simple Dockerfile example

The following example uses four instructions:

`ARG` defines a build-time variable.

`FROM` specifies the parent image on which the generated Docker image will be based. If a base image policy has been configured, allowing only certain images to be used, the specified image must match one of the image references in the policy. If it does not, codespaces for this repository will be created in recovery mode. See "Restricting the base image for codespaces."

`COPY` copies a file from the repository and adds it to the filesystem of the codespace.

`RUN` updates package lists and runs a script. You can also use a `RUN` instruction to install software, as shown by the commented out instructions. To run multiple commands, use `&&` to combine the commands into a single `RUN` statement.

Dockerfile

```
ARG VARIANT="16"
FROM mcr.microsoft.com/devcontainers/javascript-node:1-${VARIANT}

RUN apt-get update && export DEBIAN_FRONTEND=noninteractive \
    && apt-get -y install --no-install-recommends bundler

# [Optional] Uncomment if you want to install an additional version
#  of node using nvm
# ARG EXTRA_NODE_VERSION=18
# RUN su node -c "source /usr/local/share/nvm/nvm.sh \
#    && nvm install ${EXTRA_NODE_VERSION}"

COPY ./script-in-your-repo.sh /tmp/scripts/script-in-codespace.sh
RUN apt-get update && bash /tmp/scripts/script-in-codespace.sh
```

Note: In the above example, the script that's copied to the codespace (`script-in-your-repo.sh`) must exist in your repository.

For more information about Dockerfile instructions, see "Dockerfile reference" in the Docker documentation.

Using a Dockerfile

To use a Dockerfile as part of a dev container configuration, reference it in your `devcontainer.json` file by using the `dockerfile` property.

JSONC

```
{
  // ...
  "build": { "dockerfile": "Dockerfile" },
  // ...
}
```

Various options are available to you if you want to use existing container orchestration in your dev container. See the "Orchestration options" section of the Specification on the Development Containers website.

Using the default dev container configuration

If you don't add a dev container configuration to your repository, or if your configuration does not specify a base image to use, then GitHub creates a container from a default Linux image. This Linux image includes a number of runtime versions for popular languages like Python, Node, PHP, Java, Go, C++, Ruby, and .NET Core/C#. The latest or LTS releases of these languages are used. There are also tools to support data science and machine learning, such as JupyterLab and Conda. The default dev container image also includes other developer tools and utilities like Git, GitHub CLI, yarn, openssh, and vim. To see all the languages, runtimes, and tools that are included use the `devcontainer-info content-url` command inside your codespace terminal and follow the URL that the command outputs.

For information about what's included in the default Linux image, see the `devcontainers/images` repository.

The default configuration is a good option if you're working on a small project that uses the languages and tools that GitHub Codespaces provides.

Note: GitHub does not charge for storage of containers built from the default dev container image. For more information about billing for codespace storage, see "About billing for GitHub Codespaces." For information on how to check whether a codespace was built from the default dev container image, see "Getting the most out of your included usage."

Using a predefined dev container configuration

If you use Codespaces in Visual Studio Code, or in a web browser, you can create a dev container configuration for your repository by choosing from a list of predefined configurations. These configurations provide common setups for particular project types, and can help you quickly get started with a configuration that already has the appropriate container options, Visual Studio Code settings, and Visual Studio Code extensions that should be installed.

Using a predefined configuration is a great idea if you need some additional extensibility. You can also start with a predefined configuration and amend it as needed for your project. For more information about the definitions of predefined dev containers, see the `devcontainers/images` repository.

You can add a predefined dev container configuration either while working in a codespace, or while working on a repository locally. To do this in VS Code while you are working locally, and not

connected to a codespace, you must have the "Dev Containers" extension installed and enabled. For more information about this extension, see the VS Code Marketplace. The following procedure describes the process when you are using a codespace. The steps in VS Code when you are not connected to a codespace are very similar.

1. Access the Visual Studio Code Command Palette (`Shift+Command+P` / `Ctrl+Shift+P`), then start typing "add dev". Click **Codespaces: Add Dev Container Configuration Files**.
2. Click **Create a new configuration**.
3. Click **Show All Definitions**.
4. Click the definition you want to use.
5. Follow the prompts to customize your definition.
6. Click **OK**.
7. If you are working in a codespace, apply your changes by clicking **Rebuild now** in the pop-up at the bottom right of the window. For more information about rebuilding your container, see "Applying configuration changes to a codespace."

Adding additional features to your `devcontainer.json` file

Features are self-contained units of installation code and dev container configuration, designed to work across a wide range of base container images. You can use features to quickly add tools, runtimes, or libraries to your codespace image. For more information, see the available features and features specification on the Development Containers website.

You can add features to a `devcontainer.json` file from VS Code or from your repository on GitHub. See "Adding features to a devcontainer.json file."

Creating a custom dev container configuration

If none of the predefined configurations meets your needs, you can create a custom configuration by writing your own `devcontainer.json` file.

- If you're adding a single `devcontainer.json` file that will be used by everyone who creates a codespace from your repository, create the file within a `.devcontainer` directory at the root of the repository.
- If you want to offer users a choice of configuration, you can create multiple custom `devcontainer.json` files, each located within a separate subdirectory of the `.devcontainer` directory.

  Notes:

  - You can't locate your `devcontainer.json` files in directories more than one level below `.devcontainer`. For example, a file at `.devcontainer/teamA/devcontainer.json` will work, but `.devcontainer/teamA/testing/devcontainer.json` will not.
  - When users create codespaces from the **Use this template** button in a template repository, they will not be given a choice between configurations. The codespace will be built based on the default configuration defined in

`.devcontainer/devcontainer.json`, or in `.devcontainer.json` at the root of your repository. See "Setting up a template repository for GitHub Codespaces."

If multiple `devcontainer.json` files are found in the repository, they are listed in the **Dev container configuration** dropdown on the codespace creation options page. See "Creating a codespace for a repository."

Adding a `devcontainer.json` file

If you don't already have a `devcontainer.json` file in your repository, you can quickly add one from GitHub.

1. Navigate to your repository and click the **Code** dropdown.
2. In the **Codespaces** tab, click the ellipsis (**...**), then select **Configure dev container**.

A new `.devcontainer/devcontainer.json` file will open in the editor. The file will contain some initial properties, including a `features` object to which you can add new tools, libraries, or runtimes. See "Adding features to a devcontainer.json file."

If your repository already contains one or more `devcontainer.json` files, then clicking **Configure dev container** will open the existing `devcontainer.json` file with the highest precedence according to the specification on the Development Containers website.

Default configuration selection during codespace creation

If `.devcontainer/devcontainer.json` or `.devcontainer.json` exists, it will be the default selection in the list of available configuration files when you create a codespace. If neither file exists, the default dev container configuration will be selected by default.

In the following screenshot, the repository does not contain `.devcontainer/devcontainer.json` or `.devcontainer.json` files, so the default dev container configuration is selected. However, two alternative configuration files have been defined in subdirectories of the `.devcontainer` directory, so these are listed as options.

Editing the devcontainer.json file

You can add and edit the supported configuration keys in the `devcontainer.json` file to specify aspects of the codespace's environment, like which VS Code extensions will be installed. For information about the settings and properties that you can set in a `devcontainer.json` file, see the Specification on the Development Containers website.

The `devcontainer.json` file is written using the JSONC (JSON with comments) format. This allows you to include comments within the configuration file. See "Editing JSON with VS Code" in the VS Code documentation.

Note: If you use a linter to validate the `devcontainer.json` file, make sure it is set to JSONC and not JSON or comments will be reported as errors.

Interface settings for VS Code

You can configure the interface settings for VS Code, with three scopes: User, Remote [Codespaces], and Workspace. You can view these scopes in the VS Code Settings editor.

To display the Setting editor, use the keyboard shortcut `Command+,` (Mac) / `Ctrl+,` (Linux/Windows).

If a setting is defined in multiple scopes, Workspace settings take priority, then Remote [Codespaces], then User.

You can define default interface settings for VS Code in two places.

- Interface settings defined in the `.vscode/settings.json` file in your repository are applied as Workspace-scoped settings in the codespace.
- Interface settings defined in the `settings` key in the `devcontainer.json` file are applied as Remote [Codespaces]-scoped settings in the codespace.

Applying configuration changes to a codespace

Changes to a configuration will be applied the next time you create a codespace. However, you can apply your changes to an existing codespace by rebuilding the container. You can do this within a codespace in the VS Code web client or desktop application, or you can use GitHub CLI.

Note: When you rebuild the container in a codespace, changes you have made outside the `/workspaces` directory are cleared. Changes you have made inside the `/workspaces` directory, which includes the clone of the repository or template from which you created the codespace, are preserved over a rebuild. For more information, see "Deep dive into GitHub Codespaces."

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

1. Access the VS Code Command Palette (`Shift+Command+P` / `Ctrl+Shift+P`), then start typing "rebuild". Click **Codespaces: Rebuild Container**.

   Tip: You may occasionally want to perform a full rebuild to clear your cache and rebuild your container with fresh images. For more information, see "Rebuilding the container in a codespace."

2. 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 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`.

Further reading

- "Prebuilding your codespaces"