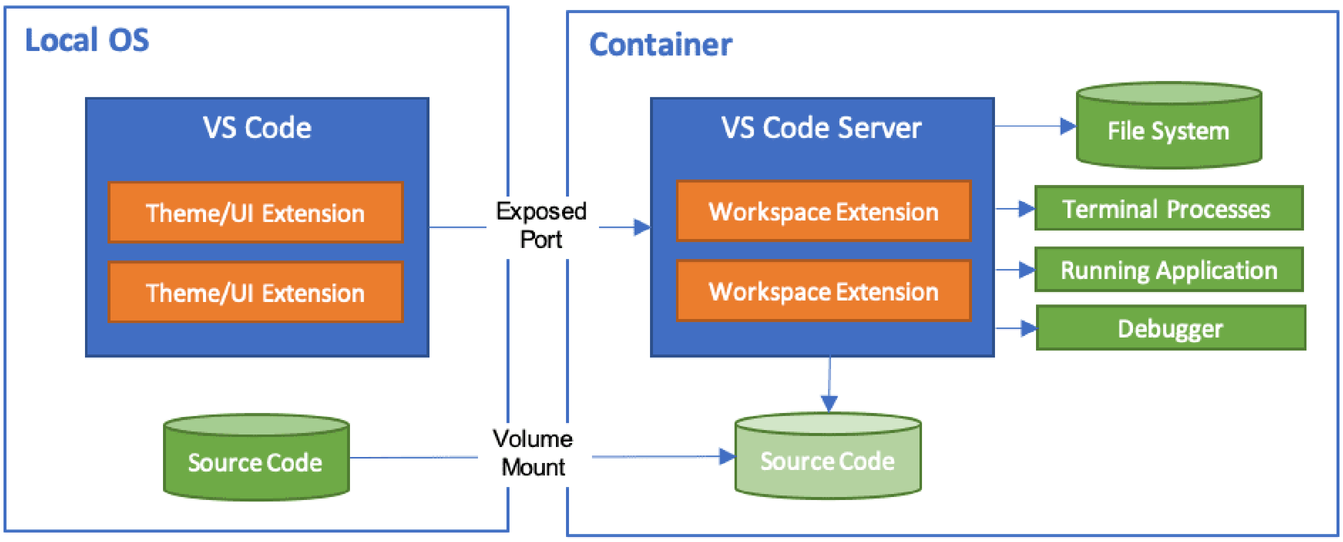
(https://vscode.dev/github/microsoft/vscode-docs/blob/main/docs/devcontainers/containers.md)

Developing inside a Container

The **Visual Studio Code Dev Containers** extension lets you use a container as a full-featured development environment. It allows you to open any folder inside (or mounted into) a container and take advantage of Visual Studio Code's full feature set. A devcontainer.json file in your project tells VS Code how to access (or create) a **development container** with a well-defined tool and runtime stack. This container can be used to run an application or to separate tools, libraries, or runtimes needed for working with a codebase.

Workspace files are mounted from the local file system or copied or cloned into the container. Extensions are installed and run inside the container, where they have full access to the tools, platform, and file system. This means that you can seamlessly switch your entire development environment just by connecting to a different container.



This lets VS Code provide a **local-quality development experience** including full IntelliSense (completions), code navigation, and debugging **regardless of where your tools (or code) are located**.

The Dev Containers extension supports two primary operating models:

- You can use a container as your full-time development environment
- You can attach to a running container (/docs/devcontainers/attach-container) to inspect it.

> **Note**: The Dev Containers extension supports the open Dev Containers Specification, which empowers anyone in any tool to configure a consistent dev environment. You can learn more in our dev container FAQ (/docs/devcontainers/faq#_can-i-use-dev-containers-outside-of-vs-code) and on the specification's site containers.dev (https://containers.dev/).

Getting started

> **Note**: You can learn how to get up-and-running quickly with dev containers in the introductory Dev Containers tutorial (/docs/devcontainers/tutorial).

System requirements

Local / Remote Host:

You can use Docker with the Dev Containers extension in a few ways, including:

- Docker installed locally.
- Docker installed on a remote environment.
- Other Docker compliant CLIs, installed locally or remotely.
    - While other CLIs may work, they are not officially supported. Note that attaching to a Kubernetes cluster (/docs/devcontainers/attach-container#_attach-to-a-container-in-a-kubernetes-cluster) only requires a properly configured kubectl CLI (https://kubernetes.io/docs/reference/kubectl/overview/).

You can learn more in the alternative Docker options doc (/remote/advancedcontainers/docker-options).

Below are some specific ways you can configure Docker on a local or remote host:

- **Windows:** Docker Desktop (https://www.docker.com/products/docker-desktop) 2.0+ on Windows 10 Pro/Enterprise. Windows 10 Home (2004+) requires Docker Desktop 2.3+ and the WSL 2 back-end (https://aka.ms/vscode-remote/containers/docker-wsl2). (Docker Toolbox is not supported. Windows container images are not supported.)
- **macOS**: Docker Desktop (https://www.docker.com/products/docker-desktop) 2.0+.
- **Linux**: Docker CE/EE (https://docs.docker.com/install/#supported-platforms) 18.06+ and Docker Compose (https://docs.docker.com/compose/install) 1.21+. (The Ubuntu snap package is not supported.)
- **Remote hosts:** 1 GB RAM is required, but at least 2 GB RAM and a 2-core CPU is recommended.

Containers:

- x86_64 / ARMv7l (AArch32) / ARMv8l (AArch64) Debian 9+, Ubuntu 16.04+, CentOS / RHEL 7+
- x86_64 Alpine Linux 3.9+

Other `glibc` based Linux containers may work if they have needed Linux prerequisites (/docs/remote/linux).

Installation

To get started, follow these steps:

1. Install and configure Docker (https://www.docker.com/get-started) for your operating system, using one of the paths below or an alternative Docker option (/remote/advancedcontainers/docker-options), like Docker on a remote host or Docker compliant CLI.

   Windows / macOS:

   1. Install Docker Desktop for Windows/Mac (https://www.docker.com/products/docker-desktop).

   2. If you are using WSL 2 on Windows, to ensure the WSL 2 back-end (https://aka.ms/vscode-remote/containers/docker-wsl2) is enabled: Right-click on the Docker taskbar item and select **Settings**. Check **Use the WSL 2 based engine** and verify your distribution is enabled under **Resources** > **WSL Integration**.

   3. When not using the WSL 2 back-end, right-click on the Docker task bar item, select **Settings** and update **Resources** > **File Sharing** with any locations your source code is kept. See tips and tricks (/docs/devcontainers/tips-and-tricks) for troubleshooting.

   Linux:

   1. Follow the official install instructions for Docker CE/EE for your distribution (https://docs.docker.com/install/#supported-platforms). If you are using Docker Compose, follow the Docker Compose directions (https://docs.docker.com/compose/install/) as well.

   2. Add your user to the `docker` group by using a terminal to run: `sudo usermod -aG docker $USER`

   3. Sign out and back in again so your changes take effect.

2. Install Visual Studio Code (https://code.visualstudio.com/) or Visual Studio Code Insiders (https://code.visualstudio.com/insiders/).

3. Install the Dev Containers extension (https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-containers). If you plan to work with other remote extensions in VS Code, you may choose to install the Remote Development extension pack (https://aka.ms/vscode-remote/download/extension).

Working with Git?

Here are two tips to consider:

- If you are working with the same repository both locally in Windows and inside a container, be sure to set up consistent line endings. See tips and tricks (/docs/remote/troubleshooting#_resolving-git-line-ending-issues-in-wsl-resulting-in-many-modified-files) for details.
- If you clone using a Git credential manager, your container should already have access to your credentials! If you use SSH keys, you can also opt in to sharing them. See Sharing Git credentials with your container (/remote/advancedcontainers/sharing-git-credentials) for details.

Picking your quick start

This document includes 3 quick starts - we recommend starting with the one that fits your workflow and interests the best:

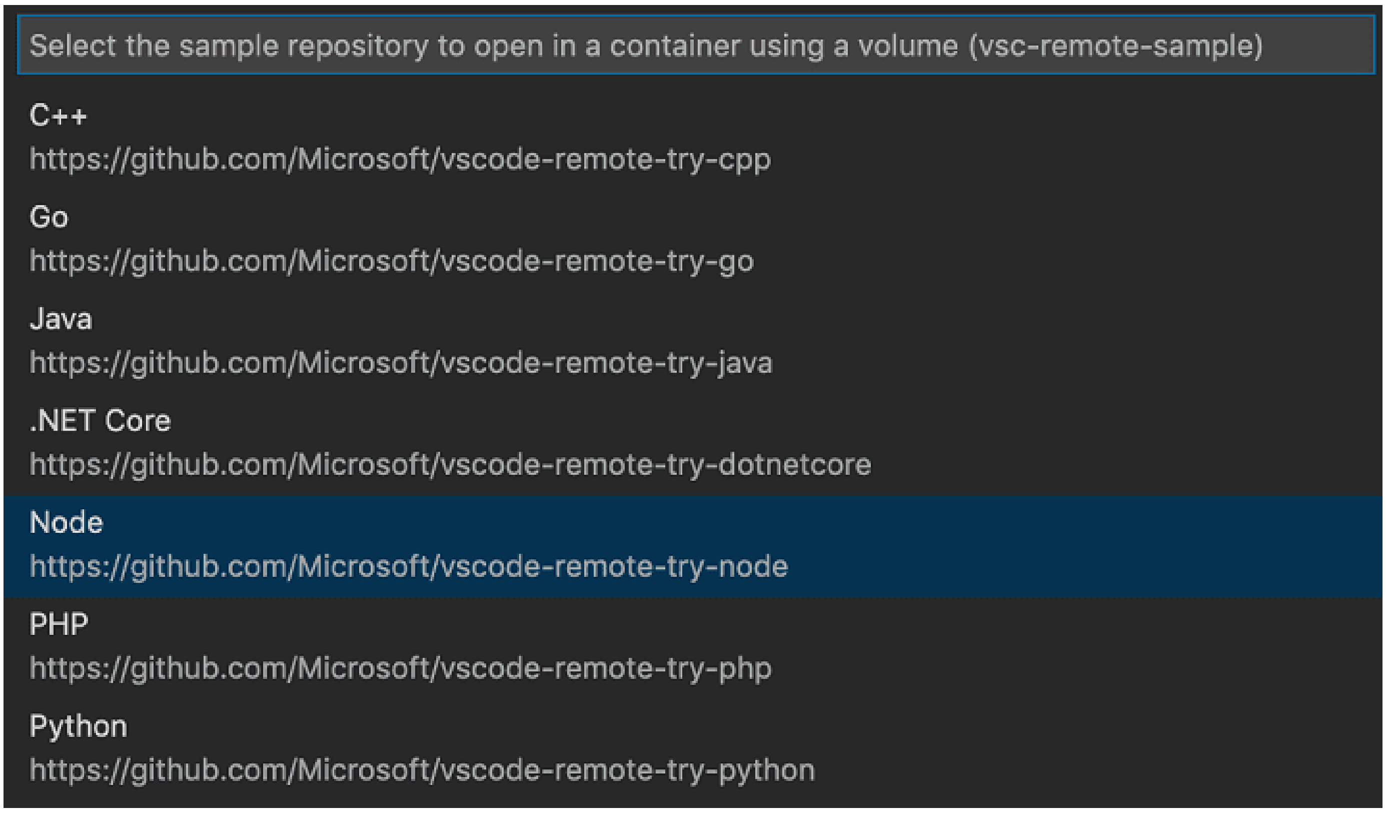
1. Want to try out a dev container in a quick sample repo? Check out Quick start 1: Try a development container.

2. Want to add a dev container to one of your existing locally cloned projects? Check out Quick start 2: Open an existing folder in a container.

3. Want to work with an isolated copy of a repo, i.e. to review a PR or investigate a branch without impacting your local work? Check out Quick start 3: Open a git repo or PR in an isolated container volume.

Quick start: Try a development container

The easiest way to get started is to try one of the sample development containers. The Containers tutorial (/docs/devcontainers/tutorial) will walk you through setting up Docker and the Dev Containers extension and let you select a sample:



Select the sample repository to open in a container using a volume (vsc-remote-sample)

C++
https://github.com/Microsoft/vscode-remote-try-cpp

Go
https://github.com/Microsoft/vscode-remote-try-go

Java
https://github.com/Microsoft/vscode-remote-try-java

.NET Core
https://github.com/Microsoft/vscode-remote-try-dotnetcore

Node
https://github.com/Microsoft/vscode-remote-try-node

PHP
https://github.com/Microsoft/vscode-remote-try-php

Python
https://github.com/Microsoft/vscode-remote-try-python

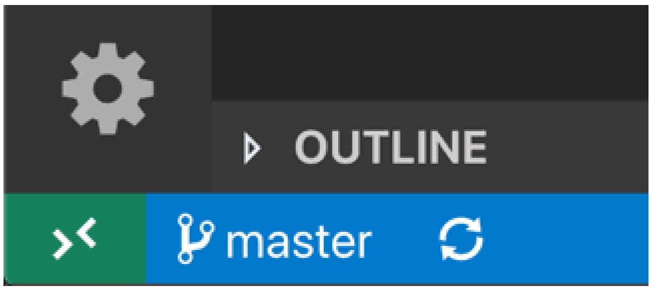
> Note: If you already have VS Code and Docker installed, then you may use open in dev container (https://vscode.dev/redirect?url=vscode://ms-vscode-remote.remote-containers/cloneInVolume?url=https://github.com/microsoft/vscode-remote-try-python). You can learn more about this and how to add it to your repos in the create a dev container guide (/docs/devcontainers/create-dev-container#_add-configuration-files-to-a-repository).

Quick start: Open an existing folder in a container

This quick start covers how to set up a dev container for an existing project to use as your full-time development environment using existing source code on your filesystem. Follow these steps:

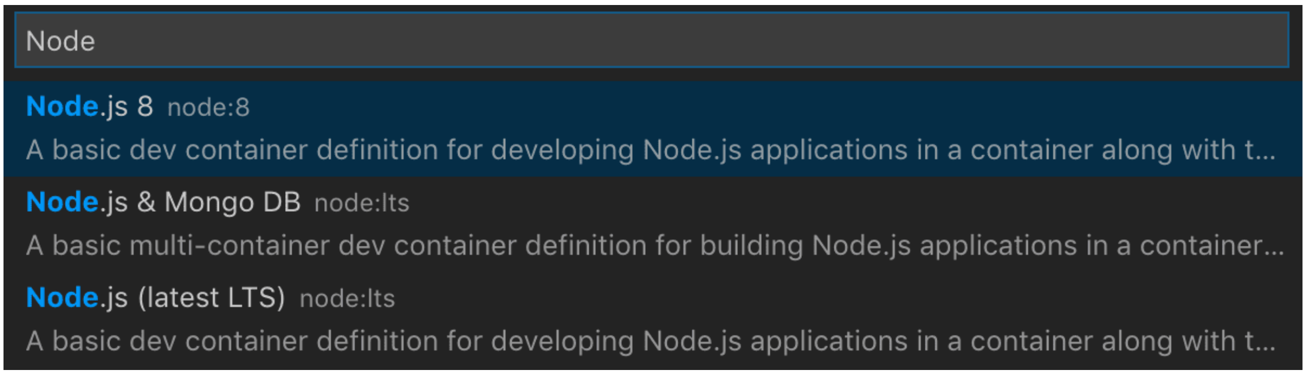
1. Start VS Code, run the **Dev Containers: Open Folder in Container...** command from the Command Palette (`F1`) or quick actions Status bar item, and select the project folder you would like to set up the container for.

   > **Tip:** If you want to edit the container's contents or settings before opening the folder, you can run **Dev Containers: Add Dev Container Configuration Files...** instead.

2. Now pick a starting point for your dev container. You can either select a base **Dev Container Template** from a filterable list, or use an existing Dockerfile (https://docs.docker.com/engine/reference/builder/) or Docker Compose file (https://docs.docker.com/compose/compose-file/#compose-file-structure-and-examples) if one exists in the folder you selected.

   > **Note:** When using Alpine Linux containers, some extensions may not work due to `glibc` dependencies in native code inside the extension.

   
   Node
   Node.js 8 node:8
   A basic dev container definition for developing Node.js applications in a container along with t...
   Node.js & Mongo DB node:lts
   A basic multi-container dev container definition for building Node.js applications in a container...
   Node.js (latest LTS) node:lts
   A basic dev container definition for developing Node.js applications in a container along with t...

   The list will be automatically sorted based on the contents of the folder you open.

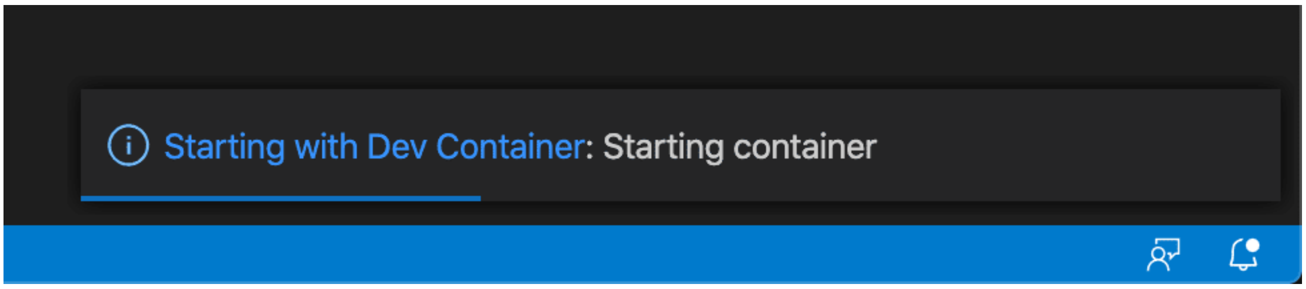
   You may be able to customize your dev container with additional Features, which you can read more about below.

   The dev container Templates displayed come from our first-party and community index (https://containers.dev/templates), which is part of the Dev Container Specification (https://containers.dev/). We host a set of Templates as part of the spec in the devcontainers/templates repository (https://github.com/devcontainers/templates). You can browse the `src` folder of that repository to see the contents of each Template.

   You can also choose to publish and distribute your own dev container Templates using the dev container CLI (https://containers.dev/implementors/templates-distribution/).

3. After picking the starting point for your container, VS Code will add the dev container configuration files to your project (`.devcontainer/devcontainer.json`).

4. The VS Code window will reload and start building the dev container. A progress notification provides status updates. You only have to build a dev container the first time you open it; opening the folder after the first successful build will be much quicker.

   
   Starting with Dev Container: Starting container

5. After the build completes, VS Code will automatically connect to the container.

You can now interact with your project in VS Code just as you could when opening the project locally. From now on, when you open the project folder, VS Code will automatically pick up and reuse your dev container configuration.

> **Tip:** Want to use a remote Docker host? See the section on opening a folder on a remote SSH host in a container for information.

While using this approach to bind mount (https://docs.docker.com/storage/bind-mounts/) the local filesystem into a container is convenient, it does have some performance overhead on Windows and macOS. There are some techniques (/remote/advancedcontainers/improve-performance) that you can apply to improve disk performance, or you can open a repository in a container using an isolated container volume instead.

Open a WSL 2 folder in a container on Windows

If you are using Windows Subsystem for Linux v2 (WSL 2) (https://learn.microsoft.com/windows/wsl/wsl2-about) and have enabled Docker Desktop's WSL 2 back-end (https://aka.ms/vscode-remote/containers/docker-wsl2), you can work with source code stored inside WSL!

Once the WSL 2 engine is enabled, you can either:

- Use the **Dev Containers: Reopen in Container** command from a folder already opened using the WSL (https://aka.ms/vscode-remote/download/wsl) extension.
- Select **Dev Containers: Open Folder in Container...** from the Command Palette (`F1`) and choose a WSL folder using the local `\\wsl$` share (from the Windows side).

The rest of the quick start applies as-is! You can learn more about the WSL extension in its documentation (/docs/remote/wsl).

Open a folder on a remote SSH host in a container

If you are using a Linux or macOS SSH host, you can use the Remote - SSH (/docs/remote/ssh) and Dev Containers extensions together. You do not even need to have a Docker client installed locally.

To do so:

1. Follow the installation (/docs/remote/ssh#_installation) and SSH host setup (/docs/remote/ssh#_ssh-host-setup) steps for the Remote - SSH extension.

2. **Optional:** Set up SSH key based authentication (/docs/remote/troubleshooting#_configuring-key-based-authentication) to the server so you do not need to enter your password multiple times.

3. Install Docker on your SSH host. You do not need to install Docker locally.

4. Follow the quick start (/docs/remote/ssh#_connect-to-a-remote-host) for the Remote - SSH extension to connect to a host and open a folder there.

5. Use the **Dev Containers: Reopen in Container** command from the Command Palette (`F1` , `Ctrl+Shift+P`).

The rest of the Dev Containers quick start applies as-is. You can learn more about the Remote - SSH extension in its documentation (/docs/remote/ssh). You can also see the Develop on a remote Docker host (/remote/advancedcontainers/develop-remote-host) article for other options if this model does not meet your needs.

Open a folder on a remote Tunnel host in a container

You can use the Remote - Tunnels (/docs/remote/tunnels) and Dev Containers extensions together to open a folder on your remote host inside of a container. You do not even need to have a Docker client installed locally. This is similar to the SSH host scenario above, but uses Remote - Tunnels instead.

To do so:

1. Follow the Getting Started (/docs/remote/tunnels#_getting-started) instructions for the Remote - Tunnels extension.

2. Install Docker on your tunnel host. You do not need to install Docker locally.

3. Follow the steps (/docs/remote/tunnels#_remote-tunnels-extension) for the Remote - Tunnels extension to connect to a tunnel host and open a folder there.

4. Use the **Dev Containers: Reopen in Container** command from the Command Palette (`F1` , `Ctrl+Shift+P`).

The rest of the Dev Containers quick start applies as-is. You can learn more about the Remote - Tunnels extension in its documentation (/docs/remote/tunnels). You can also see the Develop on a remote Docker host (/remote/advancedcontainers/develop-remote-host) article for other options if this model does not meet your needs.

Open an existing workspace in a container

You can also follow a similar process to open a VS Code multi-root workspace (/docs/editor/multi-root-workspaces) in a **single container** if the workspace only **references relative paths to sub-folders of the folder the `.code-workspace` file is in (or the folder itself).**

You can either:

- Use the **Dev Containers: Open Workspace in Container...** command.
- Use **File > Open Workspace...** once you have opened a folder that contains a `.code-workspace` file in a container.

Once connected, you may want to **add the `.devcontainer` folder** to the workspace so you can easily edit its contents if it is not already visible.

Also note that, while you cannot use multiple containers for the same workspace in the same VS Code window, you can use multiple Docker Compose managed containers at once (/remote/advancedcontainers/connect-multiple-containers) from separate windows.

Quick start: Open a Git repository or GitHub PR in an isolated container volume

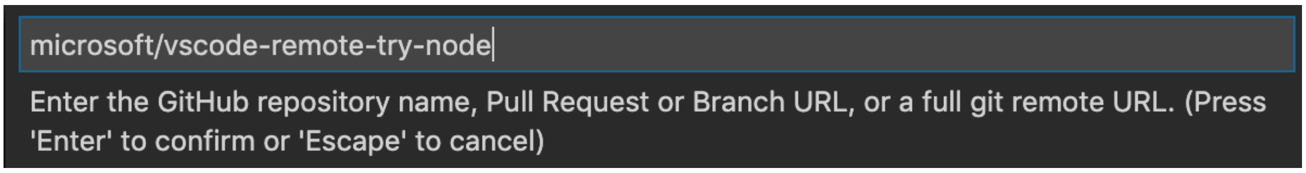
While you can open a locally cloned repository in a container, you may want to work with an isolated copy of a repository for a PR review or to investigate another branch without impacting your work.

Repository Containers use isolated, local Docker volumes instead of binding to the local filesystem. In addition to not polluting your file tree, local volumes have the added benefit of improved performance on Windows and macOS. (See Advanced Configuration Improve disk performance (/remote/advancedcontainers/improve-performance) article for information on how to use these types of volumes in other scenarios.)

For example, follow these steps to open one of the "try" repositories in a Repository Container:

1. Start VS Code and run **Dev Containers: Clone Repository in Container Volume...** from the Command Palette (`F1`).

2. Enter `microsoft/vscode-remote-try-node` (or one of the other "try" repositories), a Git URI, a GitHub branch URL, or a GitHub PR URL in the input box that appears and press `Enter` .



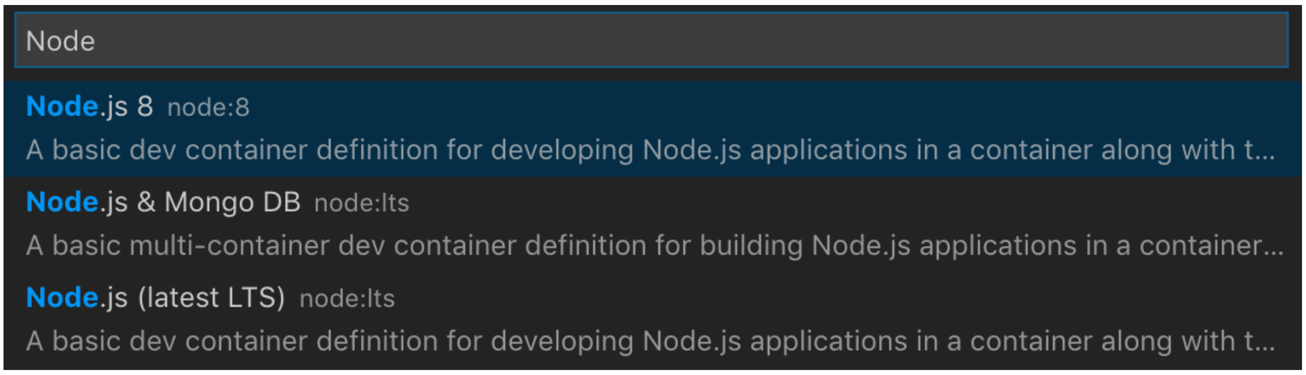
microsoft/vscode-remote-try-node

Enter the GitHub repository name, Pull Request or Branch URL, or a full git remote URL. (Press 'Enter' to confirm or 'Escape' to cancel)

Tip: If you choose a private repository, you may want to setup a credential manager or add your SSH keys to your SSH agent. See Sharing Git credentials with your container (/remote/advancedcontainers/sharing-git-credentials).

3 If your repository does not have a `.devcontainer/devcontainer.json` file in it, you'll be asked to pick a starting point from a filterable list or an existing Dockerfile (https://docs.docker.com/engine/reference/builder/) or Docker Compose file (https://docs.docker.com/compose/compose-file/#compose-file-structure-and-examples) (if one exists).

Note: When using Alpine Linux containers, some extensions may not work due to `glibc` dependencies in native code inside the extension.


Node

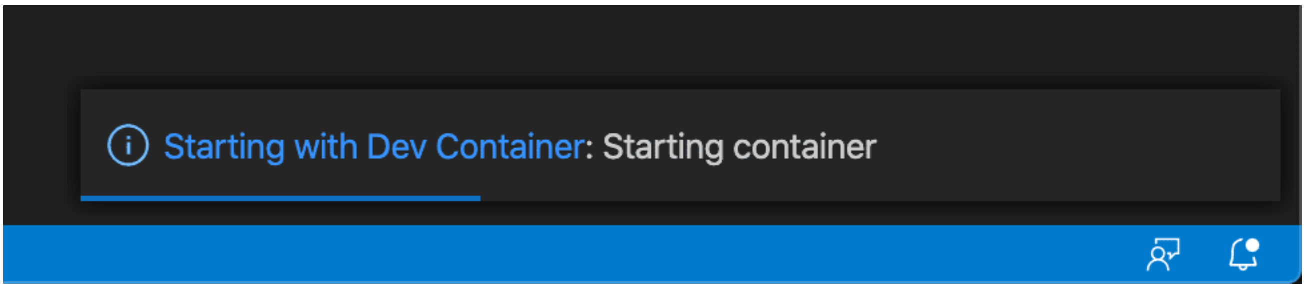
Node.js 8 node:8
A basic dev container definition for developing Node.js applications in a container along with t...

Node.js & Mongo DB node:lts
A basic multi-container dev container definition for building Node.js applications in a container...

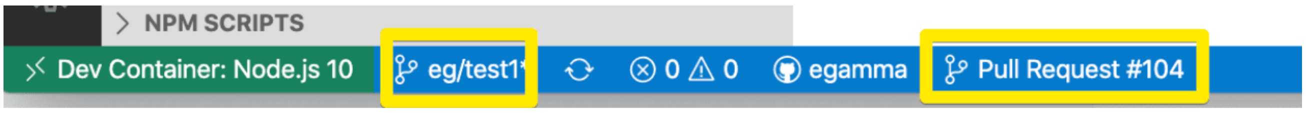
Node.js (latest LTS) node:lts
A basic dev container definition for developing Node.js applications in a container along with t...

The list will be automatically sorted based on the contents of the folder you open. The dev container Templates displayed come from our first-party and community index (https://containers.dev/templates), which is part of the Dev Container Specification (https://containers.dev/). We host a set of Templates as part of the spec in the devcontainers/templates repository (https://github.com/devcontainers/templates). You can browse the `src` folder of that repository to see the contents of each Template.

4 The VS Code window (instance) will reload, clone the source code, and start building the dev container. A progress notification provides status updates.


Starting with Dev Container: Starting container

If you pasted in a GitHub pull request URL in step 2, the PR will be automatically checked out and the GitHub Pull Requests (https://marketplace.visualstudio.com/items?itemName=GitHub.vscode-pull-request-github) extension will be installed in the container. The extension provides additional PR related features like a PR explorer, interacting with PR comments inline, and status bar visibility.



5. After the build completes, VS Code will automatically connect to the container. You can now work with the repository source code in this independent environment as you would if you had cloned the code locally.

Note that if the container fails to come up due to something like a Docker build error, you can select **Reopen in Recovery Container** in the dialog that appears to go into a "recovery container" that allows you to edit your Dockerfile or other content. This opens the docker volume with the cloned repository in a minimal container and shows you the creation log. Once you are done fixing, use **Reopen in Container** to retry.

> **Tip:** Want to use a remote Docker host? See the section on opening a folder on a remote SSH host in a container for information.

Trusting your Workspace

Visual Studio Code takes security seriously and wants to help you safely browse and edit code no matter the source or original authors. The Workspace Trust feature (/docs/editing/workspaces/workspace-trust) lets you decide whether your project folders should allow or restrict automatic code execution.

The Dev Containers extension has adopted Workspace Trust. Depending on how you open and interact with your source code, you'll be prompted to decide if you trust the code you're editing or executing at different points.

Reopen folder in container

Setting up a dev container for an existing project requires trusting the local (or WSL) folder. You will be asked to trust the local (or WSL) folder before the window reloads.

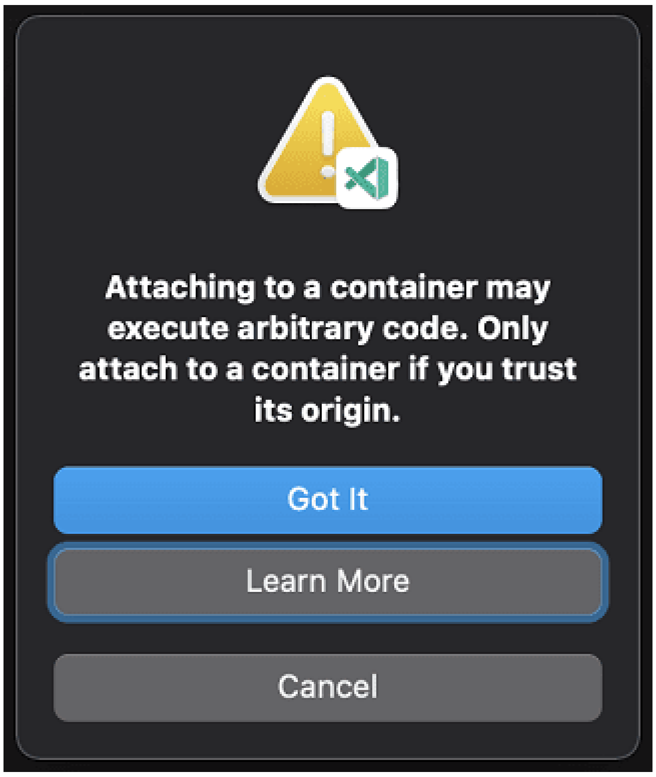
There are a couple of exceptions to this flow:

1. When clicking on a recent entry.

2. Using the **Open Folder in Container** command will ask for trust after the window reloads, if trust is not already given.

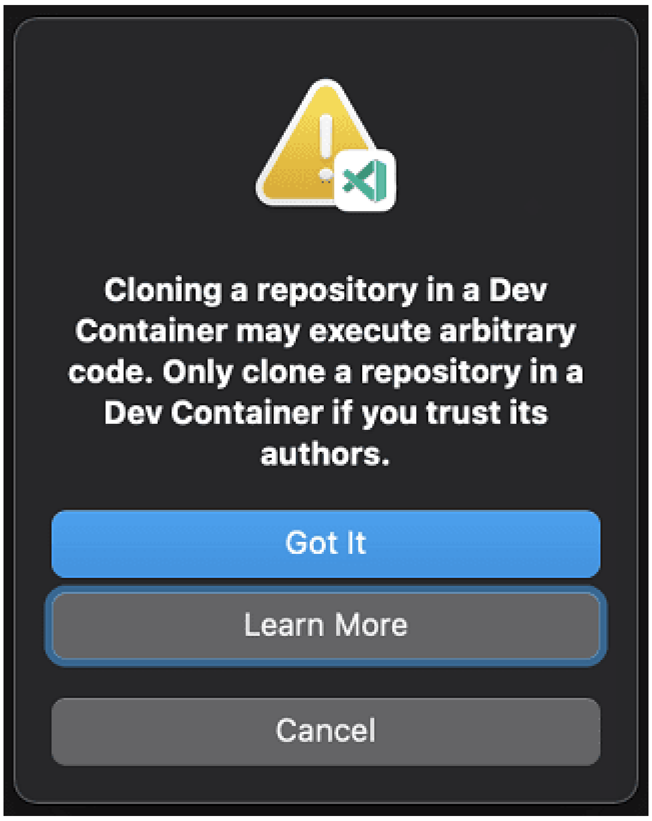
Attach to existing container

When attaching to an existing container (/docs/devcontainers/attach-container), you will be asked to confirm that attaching means you trust the container. This is only confirmed once.


Attaching to a container may execute arbitrary code. Only attach to a container if you trust its origin.

Clone repository in a volume

When cloning a repository in a container volume, you are asked to confirm that cloning a repository means you trust the repository. This is only confirmed once.


Cloning a repository in a Dev Container may execute arbitrary code. Only clone a repository in a Dev Container if you trust its authors.

Inspect volume

Inspecting a volume starts in Restricted Mode (/docs/editing/workspaces/workspace-trust#_restricted-mode), and you can trust the folder inside the container.

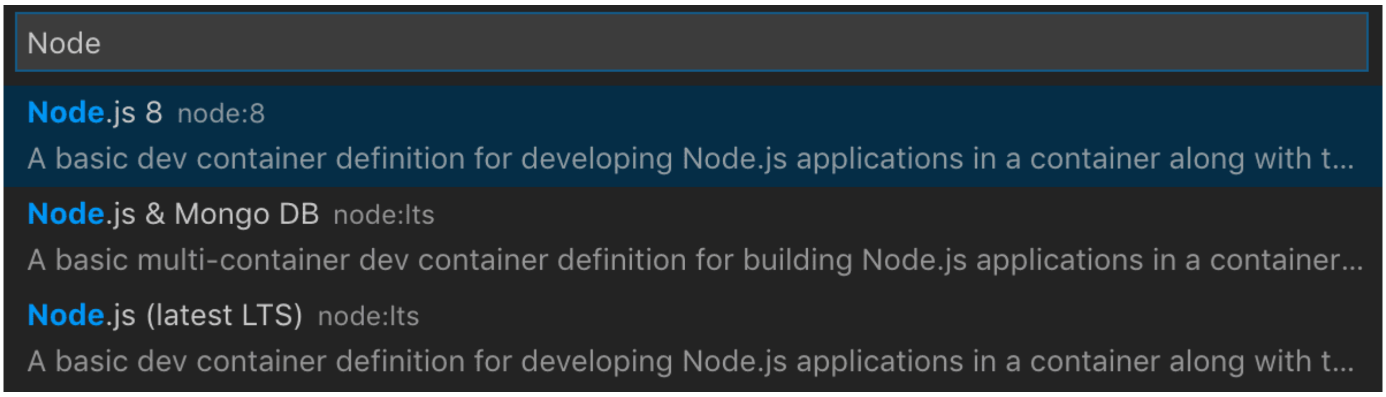
Docker daemon running remotely

This implies trusting the machine the Docker daemon runs on (/remote/advancedcontainers/develop-remote-host). There are no additional prompts to confirm (only those listed for the local/WSL case above).

Create a devcontainer.json file

VS Code's container configuration is stored in a devcontainer.json (https://containers.dev/implementors/json_reference) file. This file is similar to the `launch.json` file for debugging configurations, but is used for launching (or attaching to) your development container instead. You can also specify any extensions to install once the container is running or post-create commands to prepare the environment. The dev container configuration is either located under `.devcontainer/devcontainer.json` or stored as a `.devcontainer.json` file (note the dot-prefix) in the root of your project.

Selecting the **Dev Containers: Add Dev Container Configuration Files...** command from the Command Palette (`F1`) will add the needed files to your project as a starting point, which you can further customize for your needs. The command lets you pick a pre-defined container configuration from a list based on your folder's contents, reuse an existing Dockerfile, or reuse an existing Docker Compose file.


Node
Node.js 8 node:8
A basic dev container definition for developing Node.js applications in a container along with t...
Node.js & Mongo DB node:lts
A basic multi-container dev container definition for building Node.js applications in a container...
Node.js (latest LTS) node:lts
A basic dev container definition for developing Node.js applications in a container along with t...

You can also create a devcontainer.json by hand and use any image, Dockerfile, or set of Docker Compose files as a starting point. Here is a simple example that uses one of the pre-built Development Container images (https://github.com/devcontainers/images/tree/main/src/typescript-node):

Copy

```
{
  "image": "mcr.microsoft.com/devcontainers/typescript-node",
  "forwardPorts": [3000],
  "customizations": {
    // Configure properties specific to VS Code.
    "vscode": {
      // Add the IDs of extensions you want installed when the container is created.
      "extensions": ["streetsidesoftware.code-spell-checker"]
    }
  }
}
```

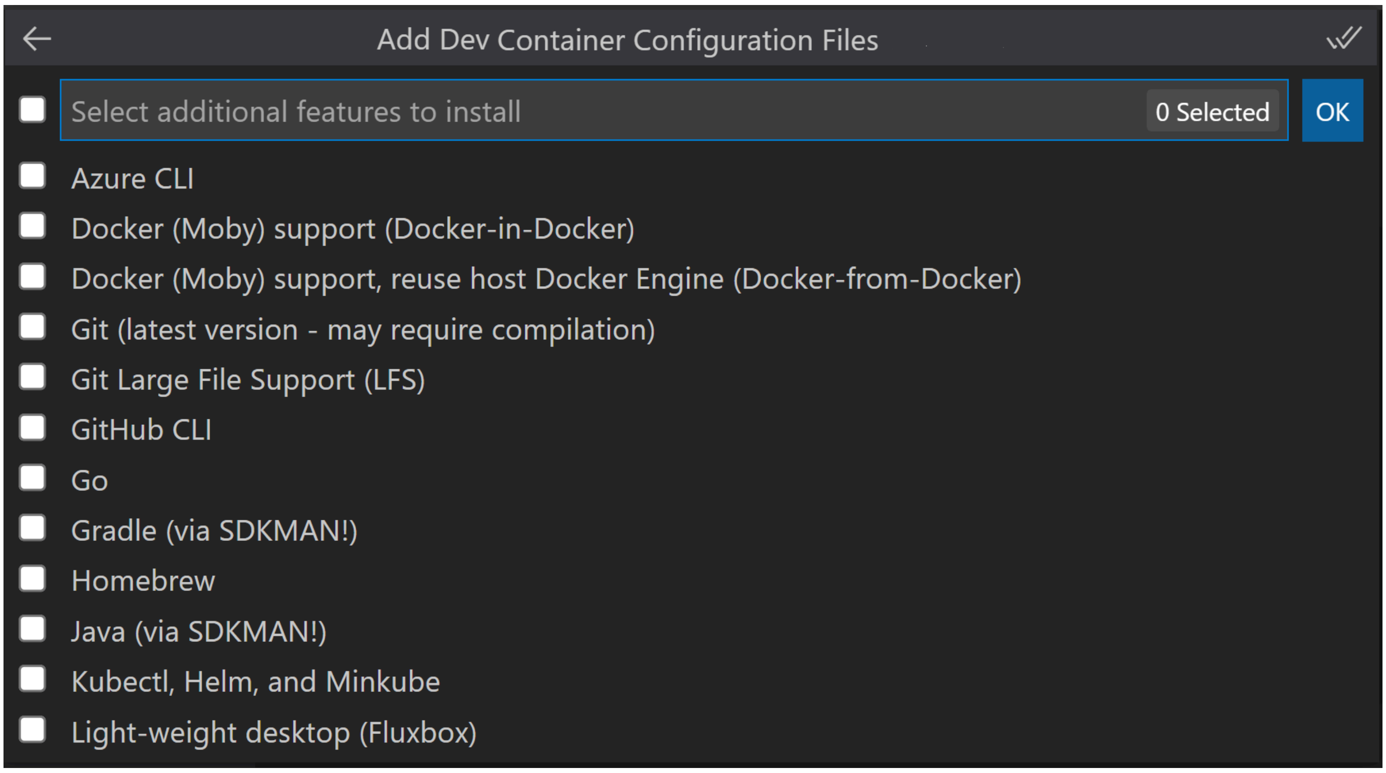
Note: Additional configuration will already be added to the container based on what's in the base image. For example, we add the `streetsidesoftware.code-spell-checker` extension above, and the container will also include `"dbaeumer.vscode-eslint"` as that's part of `mcr.microsoft.com/devcontainers/typescript-node` (https://github.com/devcontainers/images/blob/main/src/javascript-node/.devcontainer/devcontainer.json#L27). This happens automatically when pre-building using `devcontainer.json`, which you may read more about in the pre-build section.

To learn more about creating `devcontainer.json` files, see Create a Development Container (/docs/devcontainers/create-dev-container).

Dev Container Features

Development container "Features" are self-contained, shareable units of installation code and dev container configuration. The name comes from the idea that referencing one of them allows you to quickly and easily add more tooling, runtime, or library "Features" into your development container for use by you or your collaborators.

When you use **Dev Containers: Add Dev Container Configuration Files**, you're presented a list of scripts to customize the existing dev container configurations, such as installing Git or the Azure CLI:

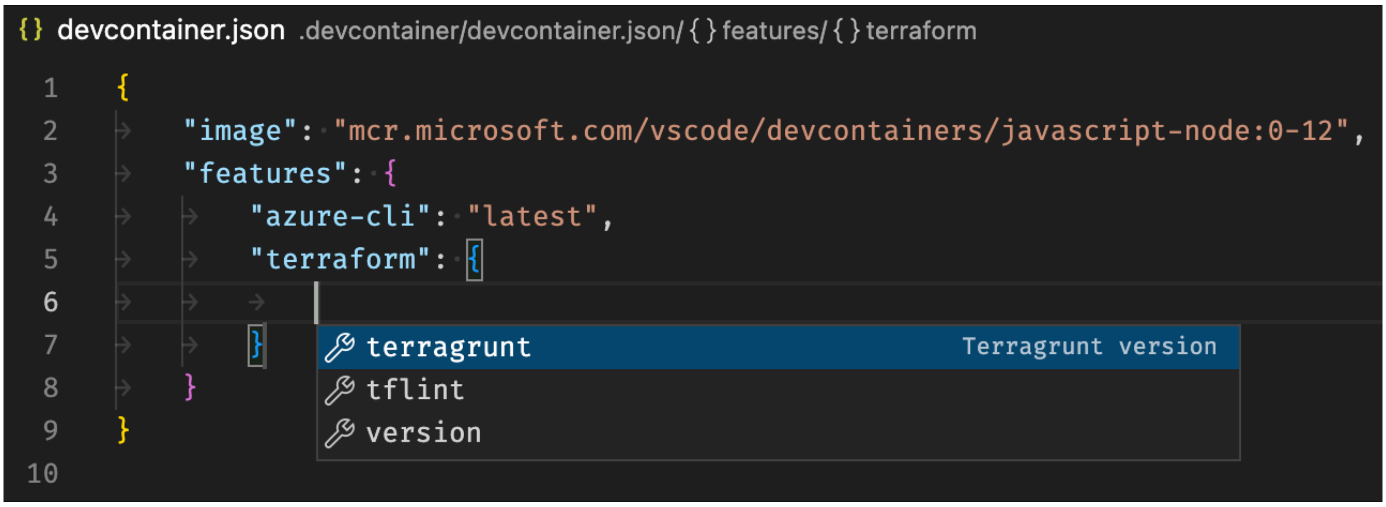

Add Dev Container Configuration Files
Select additional features to install
0 Selected
OK

Azure CLI
Docker (Moby) support (Docker-in-Docker)
Docker (Moby) support, reuse host Docker Engine (Docker-from-Docker)
Git (latest version - may require compilation)
Git Large File Support (LFS)
GitHub CLI
Go
Gradle (via SDKMAN!)
Homebrew
Java (via SDKMAN!)
Kubectl, Helm, and Minkube
Light-weight desktop (Fluxbox)

When you rebuild and reopen in your container, the Features you selected will be available in your `devcontainer.json`:

```
"features": {
    "ghcr.io/devcontainers/features/github-cli:1": {
        "version": "latest"
    }
}
```

You'll get IntelliSense when editing the `"features"` property in the `devcontainer.json` directly:



{} devcontainer.json .devcontainer/devcontainer.json/ { } features/ { } terraform

```
{
    "image": "mcr.microsoft.com/vscode/devcontainers/javascript-node:0-12",
    "features": {
        "azure-cli": "latest",
        "terraform": {
            
        }
    }
}
```

The **Dev Containers: Configure Container Features** command allows you to update an existing configuration.

The Features sourced in VS Code UI now come from a central index, which you can also contribute to. See the Dev Containers specification site (https://containers.dev/features) for the current list, and to learn how to publish and distribute Features (https://containers.dev/implementors/features-distribution/).

"Always installed" Features

Similar to how you can set extensions to always be installed in your dev container, you can use the `dev.containers.defaultFeatures (vscode://settings/dev.containers.defaultFeatures)` User setting (/docs/configure/settings) to set Features you'd always like installed:

```
"dev.containers.defaultFeatures": {
    "ghcr.io/devcontainers/features/github-cli:1": {}
},
```

Creating your own Feature

It's also easy to create and publish your own Dev Container Features. Published Features can be stored and shared as OCI Artifacts (https://github.com/opencontainers/artifacts) from any supporting public or private container registry. You can see the list of current published Features on containers.dev (https://containers.dev/features).

A Feature is a self contained entity in a folder with at least a `devcontainer-feature.json` and `install.sh` entrypoint script:

```
+-- feature
|    +-- devcontainer-feature.json
|    +-- install.sh
|    +-- (other files)
```

Check out the feature/starter (https://github.com/devcontainers/feature-starter) repository for instructions on using the dev container CLI to publish your own public or private Features.

Features specification and distribution

Features are a key part of the open-source Development Containers Specification (https://containers.dev). You can review more information about how Features work (https://containers.dev/implementors/features) and their distribution (https://containers.dev/implementors/features-distribution).

Pre-building dev container images

We recommend pre-building images with the tools you need rather than creating and building a container image each time you open your project in a dev container. Using pre-built images will result in a faster container startup, simpler configuration, and allows you to pin to a specific version of tools to improve supply-chain security and avoid potential breaks. You can automate pre-building your image by scheduling the build using a DevOps or continuous integration (CI) service like GitHub Actions.

Even better - pre-built images can contain Dev Container metadata so when you reference an image, settings will be pulled across automatically.

We recommend using the Dev Container CLI (/docs/devcontainers/devcontainer-cli) (or other specification (https://containers.dev) supporting utilities like the GitHub Action (https://github.com/marketplace/actions/devcontainers-ci)) to pre-build your images since it is kept in sync with the Dev Containers extension's latest capabilities - including dev container Features. Once you've built your image, you can push it to a container registry (like the Azure Container Registry (https://learn.microsoft.com/azure/container-registry/container-registry-get-started-docker-cli?tabs=azure-cli), GitHub Container Registry (https://docs.github.com/packages/working-with-a-github-packages-registry/working-with-the-container-registry#pushing-container-images), or Docker Hub (https://docs.docker.com/engine/reference/commandline/push)) and reference it directly.

You can use the GitHub Action in the devcontainers/ci (https://github.com/devcontainers/ci) repository to help you reuse dev containers in your workflows.

Go to the dev container CLI article on pre-building images (/docs/devcontainers/devcontainer-cli#_prebuilding) for more information.

Inheriting metadata

You can include Dev Container configuration and Feature metadata in prebuilt images via image labels (https://docs.docker.com/config/labels-custom-metadata/). This makes the image self-contained since these settings are automatically picked up when the image is referenced - whether directly, in a `FROM` in a referenced

Dockerfile, or in a Docker Compose file. This helps prevent your Dev Container config and image contents from getting out of sync, and allows you to push updates of the same configuration to multiple repositories through a simple image reference.

This metadata label is **automatically added** when you pre-build using the Dev Container CLI (/docs/devcontainers/devcontainer-cli) (or other specification (https://containers.dev) supporting utilities like the GitHub Action (https://github.com/marketplace/actions/devcontainers-ci) or Azure DevOps task (https://marketplace.visualstudio.com/items?itemName=devcontainers.ci)) and includes settings from `devcontainer.json` and any referenced Dev Container Features.

This allows you to have a separate **more complex** `devcontainer.json` you use to pre-build your image, and then a dramatically **simplified one** in one or more repositories. The contents of the image will be merged with this simplified `devcontainer.json` content at the time you create the container (go to the spec (https://containers.dev/implementors/spec/#merge-logic) for information on merge logic). But at its simplest, you can just reference the image directly in `devcontainer.json` for the settings to take effect:

Copy

```
{
  "image": "mcr.microsoft.com/devcontainers/go:1"
}
```

Note that you can also opt to manually add metadata to an image label instead. These properties will be picked up even if you didn't use the Dev Container CLI to build (and can be updated by the CLI even if you do). For example, consider this Dockerfile snippet:

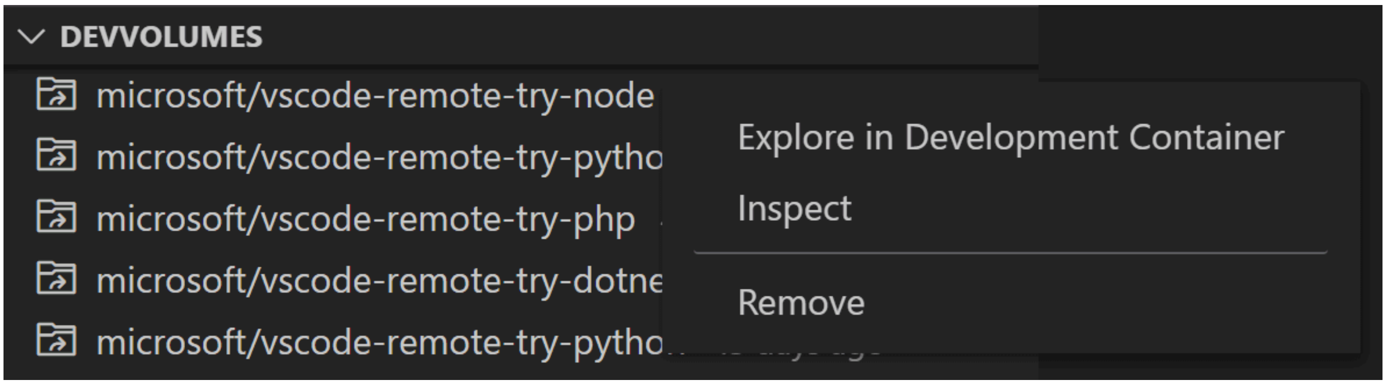
Copy

```
LABEL devcontainer.metadata='[{ \
  "capAdd": [ "SYS_PTRACE" ], \
  "remoteUser": "devcontainer", \
  "postCreateCommand": "yarn install" \
}]'
```

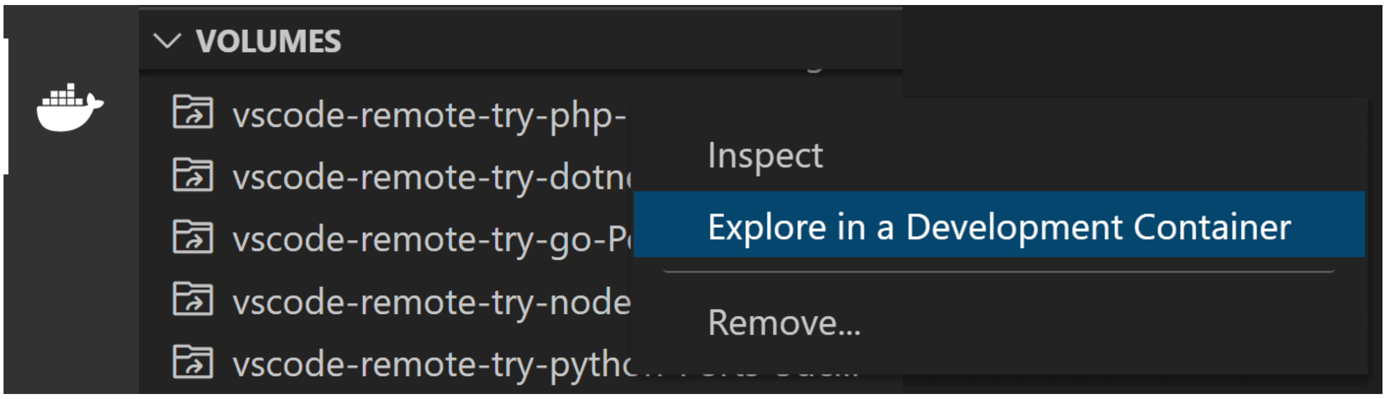
Inspecting volumes

Occasionally you may run into a situation where you are using a Docker named volume that you want to inspect or make changes in. You can use VS Code to work with these contents without creating or modifying `devcontainer.json` file by selecting the **Dev Containers: Explore a Volume in a Dev Container...** from the Command Palette (`F1`).

You can also inspect your volumes in the Remote Explorer. Make sure you have Containers selected in the dropdown, then you'll notice a **Dev Volumes** section. You can right-click on a volume to inspect its creation information, like when the volume was created, what repository was cloned into it, and the mountpoint. You can also explore it in a dev container.


DEVVOLUMES
microsoft/vscode-remote-try-node
microsoft/vscode-remote-try-pytho
microsoft/vscode-remote-try-php
microsoft/vscode-remote-try-dotne
microsoft/vscode-remote-try-pytho
Explore in Development Container
Inspect
Remove

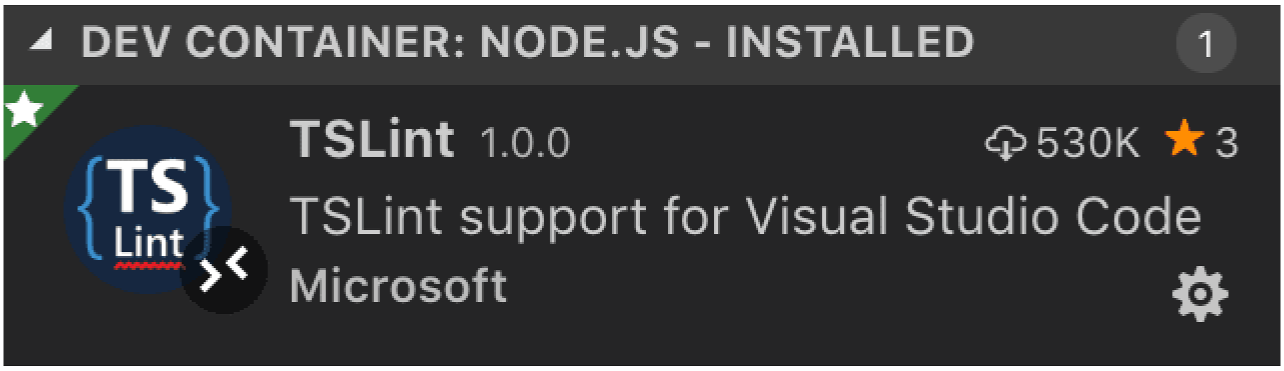
If you have the [Docker extension (https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-docker)](https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-docker) installed, you can right-click on a volume in the **Volumes** section of the **Docker Explorer** and select **Explore in a Development Container**.

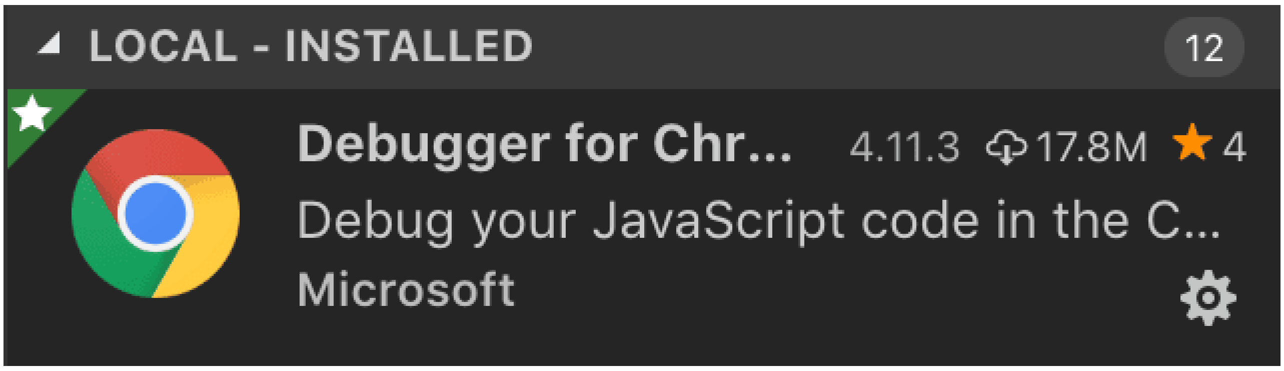


Managing extensions

VS Code runs extensions in one of two places: locally on the UI / client side, or in the container. While extensions that affect the VS Code UI, like themes and snippets, are installed locally, most extensions will reside inside a particular container. This allows you to install only the extensions you need for a given task in a container and seamlessly switch your entire tool-chain just by connecting to a new container.

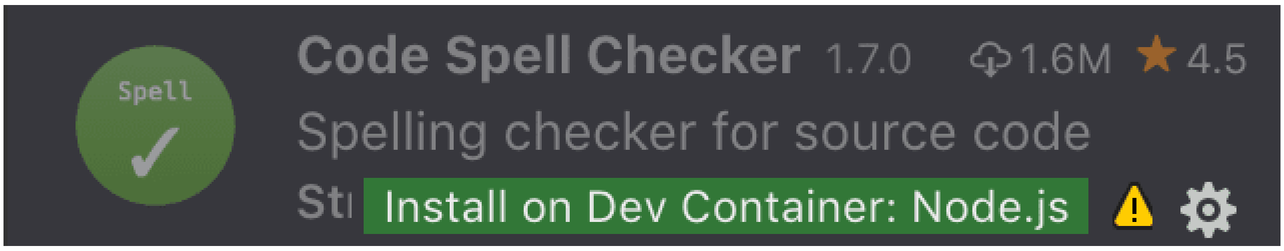
If you install an extension from the Extensions view, it will automatically be installed in the correct location. You can tell where an extension is installed based on the category grouping. There will be a **Local - Installed** category and also one for your container.


DEV CONTAINER: NODE.JS - INSTALLED
1
TSLint 1.0.0
530K
3
TSLint support for Visual Studio Code
Microsoft

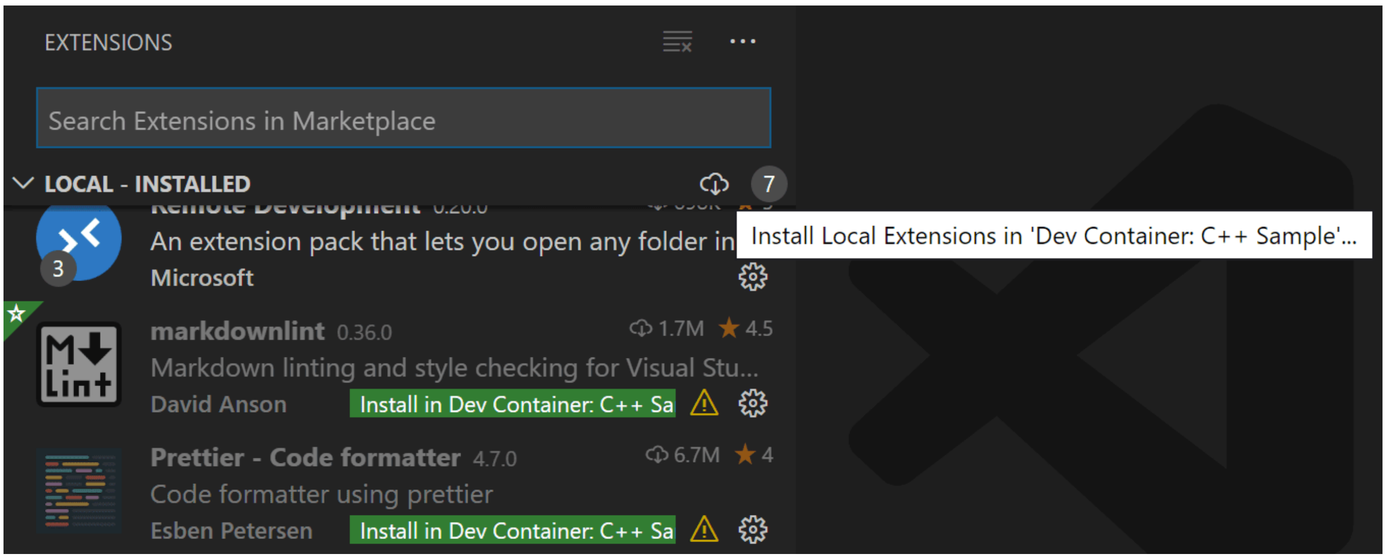

LOCAL - INSTALLED 12
Debugger for Chr... 4.11.3 17.8M 4
Debug your JavaScript code in the C...
Microsoft

> **Note:** If you are an extension author and your extension is not working properly or installs in the wrong place, see Supporting Remote Development (/api/advanced-topics/remote-extensions) for details.

Local extensions that actually need to run remotely will appear **Disabled** in the **Local - Installed** category. Select **Install** to install an extension on your remote host.


Code Spell Checker 1.7.0 1.6M 4.5
Spelling checker for source code
Install on Dev Container: Node.js

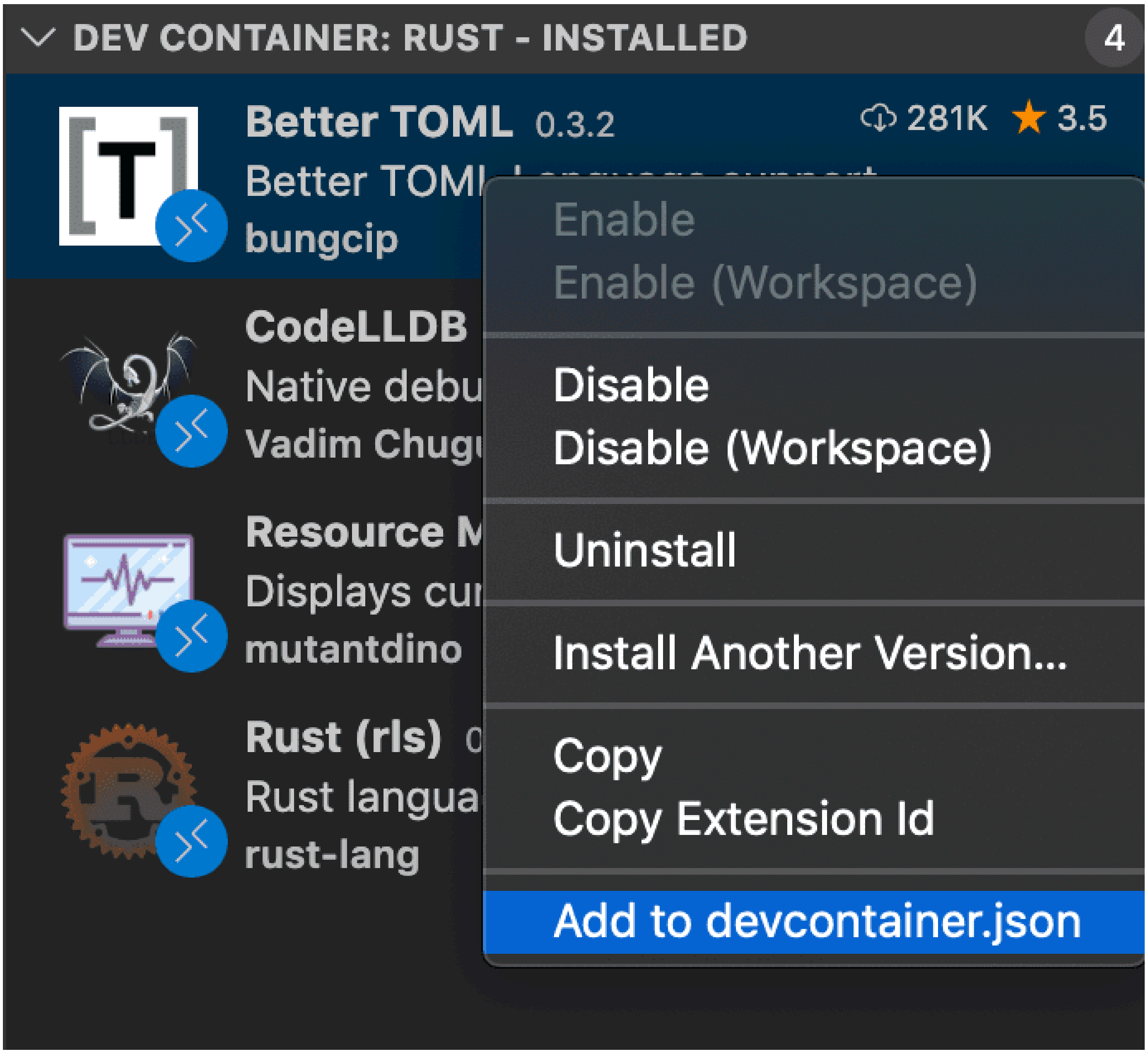
You can also install all locally installed extensions inside the Dev Container by going to the Extensions view and selecting **Install Local Extensions in Dev Container: {Name}** using the cloud button at the right of the **Local - Installed** title bar. This will display a dropdown where you can select which locally installed extensions to install in your container.


EXTENSIONS
Search Extensions in Marketplace
LOCAL - INSTALLED 7
An extension pack that lets you open any folder in
Microsoft
Install Local Extensions in 'Dev Container: C++ Sample'...
markdownlint 0.36.0 1.7M 4.5
Markdown linting and style checking for Visual Stu...
David Anson Install in Dev Container: C++ Sa
Prettier - Code formatter 4.7.0 6.7M 4
Code formatter using prettier
Esben Petersen Install in Dev Container: C++ Sa

However, some extensions may require you to install additional software (/docs/devcontainers/create-dev-container#_install-additional-software) in the container. Consult extension documentation for details if you encounter issues.

Adding an extension to devcontainer.json

While you can edit your devcontainer.json (/docs/devcontainers/create-dev-container#_create-a-devcontainerjson-file) file by hand to add a list of extension IDs, you can also right-click on any extension in the Extensions view and select **Add to devcontainer.json**.



Opt out of extensions

If a base image or Feature configures an extension that you do not want installed in your dev container, you can opt out by listing the extension with a minus sign. For example:

```
{
  "image": "mcr.microsoft.com/devcontainers/typescript-node:1-20-bookworm",
  "customizations": {
    "vscode": {
      "extensions": ["-dbaeumer.vscode-eslint"]
    }
  }
}
```

"Always installed" extensions

If there are extensions that you would like always installed in any container, you can update the ⚙ `dev.containers.defaultExtensions (vscode://settings/dev.containers.defaultExtensions)` User setting (/docs/configure/settings). For example, if you wanted to install the GitLens (https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens) and Resource Monitor (https://marketplace.visualstudio.com/items?itemName=mutantdino.resourcemonitor) extensions, you would specify their extension IDs as follows:

```
"dev.containers.defaultExtensions": [
    "eamodio.gitlens",
    "mutantdino.resourcemonitor"
]
```

Copy

Advanced: Forcing an extension to run locally or remotely

Extensions are typically designed and tested to either run locally or remotely, not both. However, if an extension supports it, you can force it to run in a particular location in your `settings.json` file.

For example, the setting below will force the Docker (https://marketplace.visualstudio.com/items?itemName=ms-azuretools.vscode-docker) extension to run locally and Remote - SSH: Editing Configuration Files (https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-ssh-edit) extension to run remotely instead of their defaults:

```
"remote.extensionKind": {
    "ms-azuretools.vscode-docker": [ "ui" ],
    "ms-vscode-remote.remote-ssh-edit": [ "workspace" ]
}
```

Copy

A value of `"ui"` instead of `"workspace"` will force the extension to run on the local UI/client side instead. Typically, this should only be used for testing unless otherwise noted in the extension's documentation since it **can break extensions**. See the section on preferred extension location (/api/advanced-topics/extension-host#preferred-extension-location) for details.

Forwarding or publishing a port

Containers are separate environments, so if you want to access a server, service, or other resource inside your container, you will need to either "forward" or "publish (https://stackoverflow.com/a/22150099)" the port to your host. You can either configure your container to always expose these ports or just forward them temporarily.

Always forwarding a port

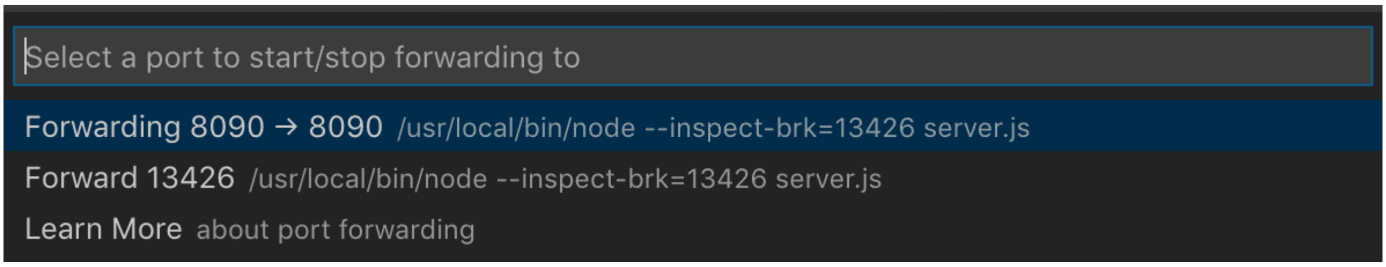
You can specify a list of ports you **always** want to forward when attaching or opening a folder in container by using the `forwardPorts` property in `devcontainer.json`.

```
"forwardPorts": [3000, 3001]
```

Simply reload / reopen the window and the setting will be applied when VS Code connects to the container.

Temporarily forwarding a port

If you need to access a port that you didn't add to `devcontainer.json` or publish in your Docker Compose file, you can **temporarily forward** a new port for the duration of the session by running the **Forward a Port** command from the Command Palette (`F1`).



Select a port to start/stop forwarding to

Forwarding 8090 → 8090 /usr/local/bin/node --inspect-brk=13426 server.js

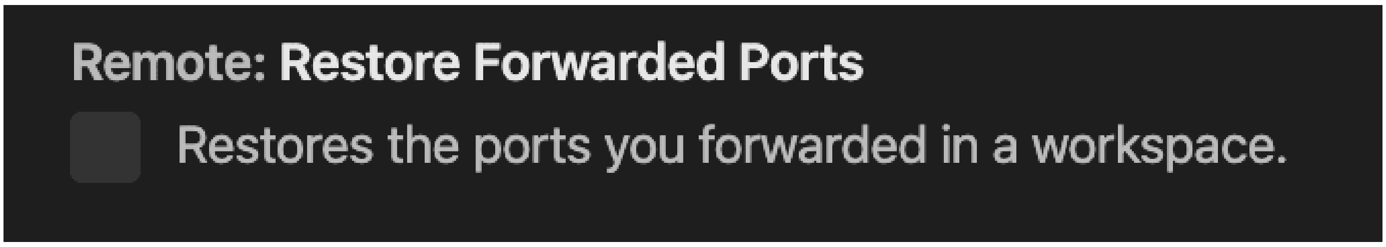
Forward 13426 /usr/local/bin/node --inspect-brk=13426 server.js

Learn More about port forwarding

After selecting a port, a notification will tell you the localhost port you should use to access the port in the container. For example, if you forwarded an HTTP server listening on port 3000, the notification may tell you that it was mapped to port 4123 on localhost. You can then connect to this remote HTTP server using `http://localhost:4123` .

This same information is available in the **Forwarded Ports** section of the Remote Explorer if you need to access it later.

If you would like VS Code to remember any ports you have forwarded, check **Remote: Restore Forwarded Ports** in the Settings editor (`Ctrl+,`) or set `"remote.restoreForwardedPorts": true` in `settings.json` .



Remote: Restore Forwarded Ports

Restores the ports you forwarded in a workspace.

Publishing a port

Docker has the concept of "publishing" ports when the container is created. Published ports behave very much like ports you make available to your local network. If your application only accepts calls from `localhost` , it will reject connections from published ports just as your local machine would for network calls. Forwarded ports, on the other hand, actually look like `localhost` to the application. Each can be useful in different situations.

To publish a port, you can:

1. **Use the appPort property:** If you reference an image or Dockerfile in `devcontainer.json` , you can use the `appPort` property to publish ports to the host.

   ```
   "appPort": [ 3000, "8921:5000" ]
   ```

2. **Use the Docker Compose ports mapping:** The ports mapping (https://docs.docker.com/compose/compose-file#ports) can easily be added your `docker-compose.yml` file to publish additional ports.

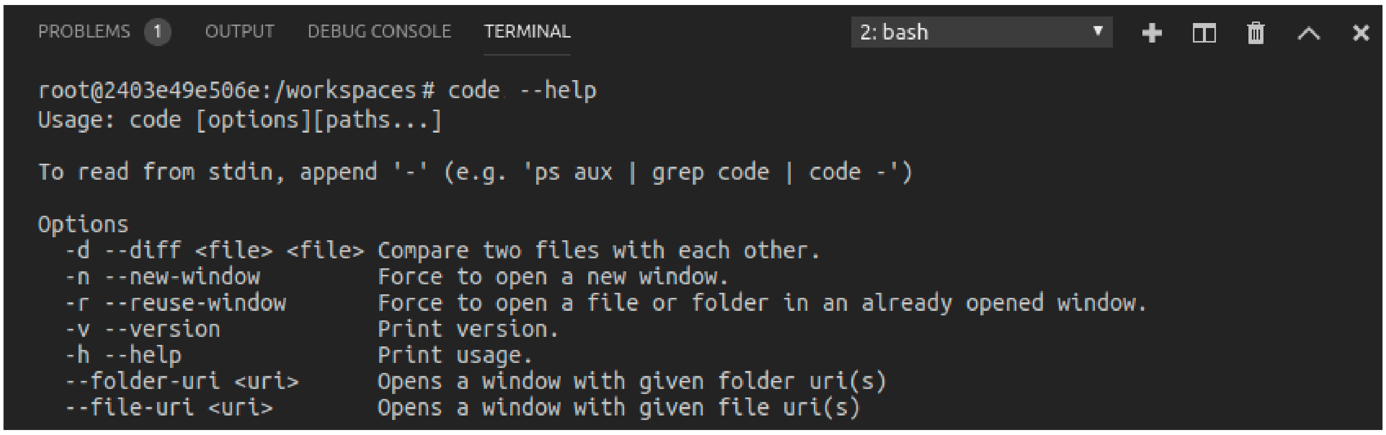
   ```
   ports:
   - "3000"
   - "8921:5000"
   ```

In each case, you'll need to rebuild your container for the setting to take effect. You can do this by running the **Dev Containers: Rebuild Container** command in the Command Palette (`F1`) when you are connected to the container.

Opening a terminal

Opening a terminal in a container from VS Code is simple. Once you've opened a folder in a container, **any terminal window** you open in VS Code (**Terminal** > **New Terminal**) will automatically run in the container rather than locally.

You can also use the `code` command line from this same terminal window to perform a number of operations such as opening a new file or folder in the container. Type `code --help` to learn what options are available from the command line.



```
root@2403e49e506e:/workspaces# code --help
Usage: code [options][paths...]

To read from stdin, append '-' (e.g. 'ps aux | grep code | code -')

Options
  -d --diff <file> <file> Compare two files with each other.
  -n --new-window         Force to open a new window.
  -r --reuse-window       Force to open a file or folder in an already opened window.
  -v --version            Print version.
  -h --help               Print usage.
  --folder-uri <uri>      Opens a window with given folder uri(s)
  --file-uri <uri>        Opens a window with given file uri(s)
```

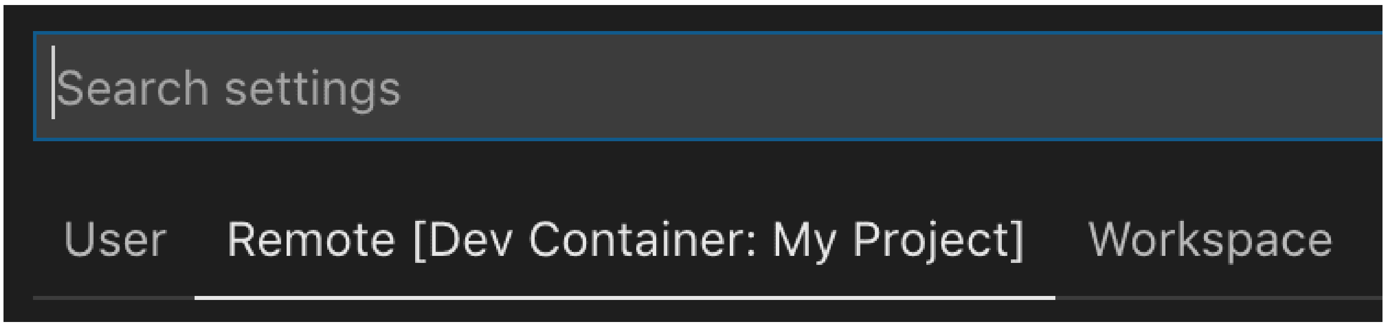
Debugging in a container

Once you've opened a folder in a container, you can use VS Code's debugger in the same way you would when running the application locally. For example, if you select a launch configuration in `launch.json` and start debugging (`F5`), the application will start on the remote host and attach the debugger to it.

See the debugging (/docs/debugtest/debugging) documentation for details on configuring VS Code's debugging features in `.vscode/launch.json` .

Container specific settings

VS Code's local user settings are also reused when you are connected to a dev container. While this keeps your user experience consistent, you may want to vary some of these settings between your local machine and each container. Fortunately, once you have connected to a container, you can also set container-specific settings by running the **Preferences: Open Remote Settings** command from the Command Palette (`F1`) or by selecting the **Remote** tab in the Settings editor. These will override any local settings you have in place whenever you connect to the container.



Default container specific settings

You can include defaults for container specific settings in `devcontainer.json` using the `settings` property. These values will be automatically placed in the container specific settings file inside the container once it is created.

For example, adding this to `.devcontainer/devcontainer.json` will set the Java home path:

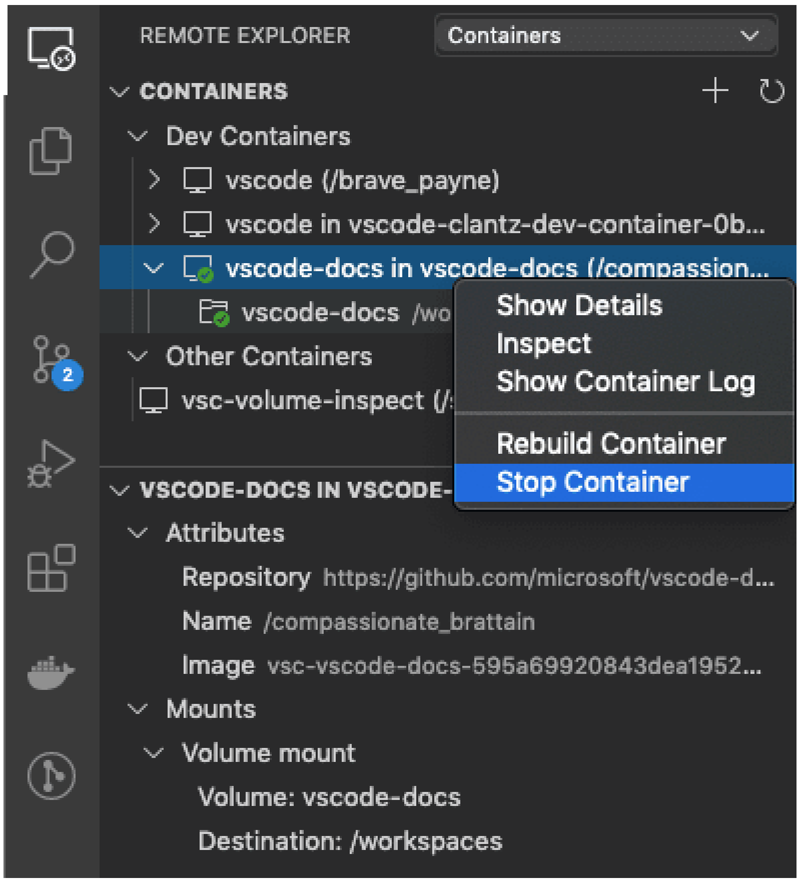
```
// Configure tool-specific properties.
"customizations": {
    // Configure properties specific to VS Code.
    "vscode": {
        "settings": {
            "java.home": "/docker-java-home"
        }
    }
}
```

Since this just establishes the default, you are still able to change the settings as needed once the container is created.

Managing containers

By default, the Dev Containers extension automatically starts the containers mentioned in the `devcontainer.json` when you open the folder. When you close VS Code, the extension automatically shuts down the containers you've connected to. You can change this behavior by adding `"shutdownAction": "none"` to `devcontainer.json` .

While you can use the command line to manage your containers, you can also use the **Remote Explorer**. To stop a container, select Containers from the dropdown (if present), right-click on a running container, and select **Stop Container**. You can also start exited containers, remove containers, and remove recent folders. From the Details view, you can forward ports and open already forwarded ports in the browser.



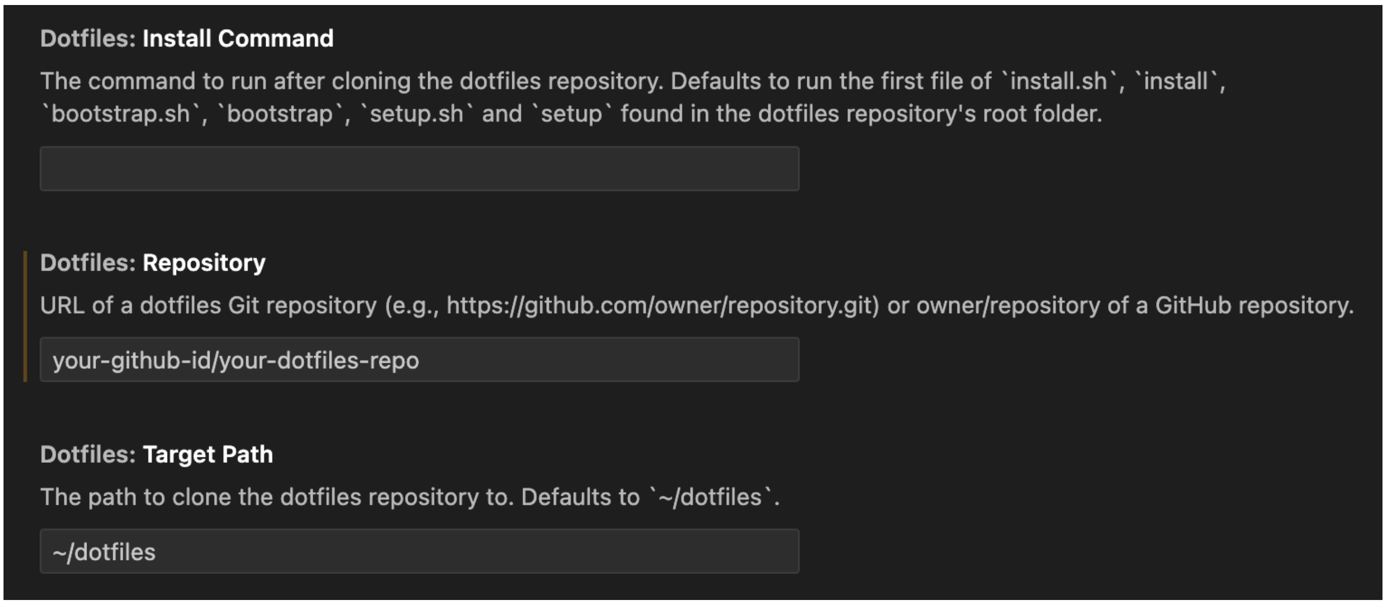
If you want to clean out images or mass-delete containers, see Cleaning out unused containers and images (/docs/devcontainers/tips-and-tricks#_cleaning-out-unused-containers-and-images) for different options.

Personalizing with dotfile repositories

Dotfiles are files whose filename begins with a dot (`.`) and typically contain configuration information for various applications. Since development containers can cover a wide range of application types, it can be useful to store these files somewhere so that you can easily copy them into a container once it is up and running.

A common way to do this is to store these dotfiles in a GitHub repository and then use a utility to clone and apply them. The Dev Containers extension has built-in support for using these with your own containers. If you are new to the idea, take a look at the different dotfiles bootstrap repositories (https://dotfiles.github.io/) that exist.

To use it, add your dotfiles GitHub repository to VS Code's User Settings (`Ctrl+,`) as follows:


Dotfiles: Install Command

The command to run after cloning the dotfiles repository. Defaults to run the first file of `install.sh`, `install`, `bootstrap.sh`, `bootstrap`, `setup.sh` and `setup` found in the dotfiles repository's root folder.

Dotfiles: Repository

URL of a dotfiles Git repository (e.g., https://github.com/owner/repository.git) or owner/repository of a GitHub repository.

your-github-id/your-dotfiles-repo

Dotfiles: Target Path

The path to clone the dotfiles repository to. Defaults to `~/dotfiles`.

~/dotfiles

Or in `settings.json`:

```
{
  "dotfiles.repository": "your-github-id/your-dotfiles-repo",
  "dotfiles.targetPath": "~/dotfiles",
  "dotfiles.installCommand": "install.sh"
}
```

From this point forward, the dotfiles repository will be used whenever a container is created.

Known limitations

Dev Containers limitations

- Windows container images are **not** supported.
- All roots/folders in a multi-root workspace will be opened in the same container, regardless of whether there are configuration files at lower levels.
- The unofficial Ubuntu Docker **snap** package for Linux is **not** supported. Follow the official Docker install instructions for your distribution (https://docs.docker.com/install/#supported-platforms).
- Docker Toolbox on Windows is not supported.
- If you clone a Git repository using SSH and your SSH key has a passphrase, VS Code's pull and sync features may hang when running remotely. Either use an SSH key without a passphrase, clone using HTTPS, or run `git push` from the command line to work around the issue.
- Local proxy settings are not reused inside the container, which can prevent extensions from working unless the appropriate proxy information is configured (for example global `HTTP_PROXY` or `HTTPS_PROXY` environment variables with the appropriate proxy information).
- There is an incompatibility between OpenSSH versions on Windows when the ssh-agent runs with version <= 8.8 and the SSH client (on any platform) runs version >= 8.9. The workaround is to upgrade OpenSSH on Windows to 8.9 or later, either using winget or an installer from Win32-OpenSSH/releases (https://github.com/PowerShell/Win32-OpenSSH/releases). (Note that `ssh-add -l` will work correctly, but `ssh <ssh-server>` will fail with `<ssh-server>: Permission denied (publickey)`. This also affects Git when using SSH to connect to the repository.)

See here for a list of active issues (https://aka.ms/vscode-remote/containers/issues) related to Containers.

Docker limitations

See the Docker troubleshooting guide for Windows (https://docs.docker.com/docker-for-windows/troubleshoot) or Mac (https://docs.docker.com/docker-for-mac/troubleshoot), consult Docker Support Resources (https://success.docker.com/article/best-support-resources) for more information.

Docker Extension limitations

If you are using the Docker or Kubernetes extension from a WSL, Remote - Tunnels or Remote - SSH window, using the **Attach Visual Studio Code** context menu action in the Docker or Kubernetes views will ask to pick from the available containers a second time.

Extension limitations

At this point, most extensions will work inside Dev Containers without modification. However, in some cases, certain features may require changes. If you run into an extension issue, see here for a summary of common problems and solutions (/docs/remote/troubleshooting#_extension-tips) that you can mention to the extension author when reporting the issue.

In addition, while Alpine support is available, some extensions installed in the container may not work due to `glibc` dependencies in native code inside the extension. See the Remote Development with Linux (/docs/remote/linux) article for details.

Advanced container configuration

See the Advanced container configuration (/remote/advancedcontainers/overview) articles for information on the following topics:

- Adding environment variables (/remote/advancedcontainers/environment-variables)
- Adding another local file mount (/remote/advancedcontainers/add-local-file-mount)
- Changing or removing the default source code mount (/remote/advancedcontainers/change-default-source-mount)
- Improving container disk performance (/remote/advancedcontainers/improve-performance)
- Adding a non-root user to your dev container (/remote/advancedcontainers/add-nonroot-user)
- Setting the project name for Docker Compose (/remote/advancedcontainers/set-docker-compose-project-name)
- Using Docker or Kubernetes from inside a container (/remote/advancedcontainers/use-docker-kubernetes)
- Connecting to multiple containers at once (/remote/advancedcontainers/connect-multiple-containers)
- Developing inside a container on a remote Docker Machine or SSH host (/remote/advancedcontainers/develop-remote-host)
- Reducing Dockerfile build warnings (/remote/advancedcontainers/reduce-docker-warnings)
- Sharing git credentials with your container (/remote/advancedcontainers/sharing-git-credentials)

devcontainer.json reference

There is a full devcontainer.json reference (https://containers.dev/implementors/json_reference), where you can review the file schema to help you customize your development containers and control how you attach to running containers.

Questions or feedback

- See Tips and Tricks (/docs/devcontainers/tips-and-tricks) or the FAQ (/docs/devcontainers/faq).
- Search on Stack Overflow (https://stackoverflow.com/questions/tagged/vscode-remote).
- Add a feature request (https://aka.ms/vscode-remote/feature-requests) or report a problem (https://aka.ms/vscode-remote/issues/new).
- Create a Dev Container Template (https://containers.dev/templates) or Feature (https://containers.dev/features) for others to use.
- Review and provide feedback on the Development Containers Specification (https://containers.dev/).
- Contribute to our documentation (https://github.com/microsoft/vscode-docs) or VS Code itself (https://github.com/microsoft/vscode).
- See our CONTRIBUTING (https://aka.ms/vscode-remote/contributing) guide for details.

Next steps

- Attach to a Running Container (/docs/devcontainers/attach-container) - Attach to an already running Docker container.
- Create a Development Container (/docs/devcontainers/create-dev-container) - Create a custom container for your work environment.
- Advanced Containers (/remote/advancedcontainers/overview) - Find solutions to advanced container scenarios.
- devcontainer.json reference (https://containers.dev/implementors/json_reference) - Review the `devcontainer.json` schema.

RSS Feed(/feed.xml) Ask questions(https://stackoverflow.com/questions/tagged/vscode)

Follow @code(https://go.microsoft.com/fwlink/?LinkID=533687)

Request features(https://go.microsoft.com/fwlink/?LinkID=533482)

Report issues(https://www.github.com/Microsoft/vscode/issues)

Watch videos(https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)

(https://www.microsoft.com)

(https://go.microsoft.com/fwlink/?LinkID=533687)

(https://github.com/microsoft/vscode)

(https://www.youtube.com/@code)