Dev Containers: Getting Started

If you are a developer and have experience with Visual Studio Code (VS Code) or Docker, then it's probably time you look at development containers (dev containers). This readme is intended to assist developers in the decision-making process needed to build dev containers. The guidance provided should be especially helpful if you are experiencing VS Code dev containers for the first time.

> **Note:** This guide is not about setting up a Docker file for deploying a running Python program for CI/CD.

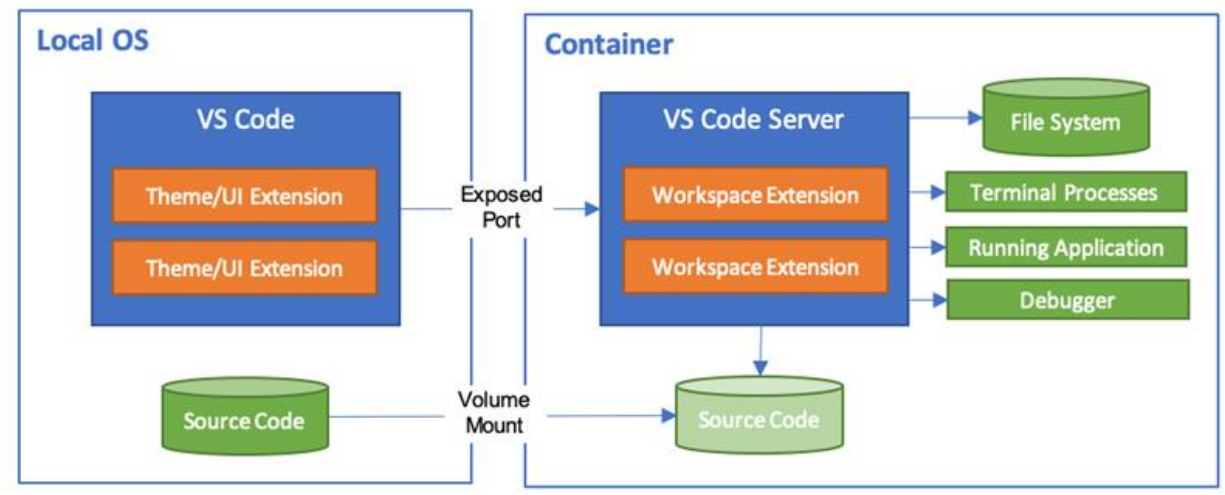
Prerequisites

- Experience with VS Code
- Experience with Docker

What are Dev Containers?

Development containers are a VS Code feature that allows developers to package a local development tool stack into the internals of a Docker container while also bringing the VS Code UI experience with them. Have you ever set a breakpoint inside a Docker container? Maybe not. Dev containers make that possible. This is all made possible through a VS Code extension called the Remote Development Extension Pack that works together with Docker to spin-up a VS Code Server within a Docker container. The VS Code

UI component remains local, but your working files are volume mounted into the container. The diagram below, taken directly from the official VS Code docs, illustrates this:



If the above diagram is not clear, a basic analogy that might help you intuitively understand dev containers is to think of them as a union between Docker's interactive mode (`docker exec -it 987654e0ff32`), and the VS Code UI experience that you are used to.

To set yourself up for the dev container experience described above, use your VS Code's Extension Marketplace to install the Remote Development Extension Pack.

How can Dev Containers Improve Project Collaboration?

VS Code dev containers have improved project collaboration between developers on recent team projects by addressing two very specific problems:

- Inconsistent local developer experiences within a team.

- Slow onboarding of developers joining a project.

The problems listed above were addressed by configuring and then sharing a dev container definition. Dev containers are defined by their base image, and the artifacts that support that base image. The base image and the artifacts that come with it live in the .devcontainer directory. This directory is where configuration begins. A central artifact to the dev container definition is a configuration file called `devcontainer.json`. This file orchestrates the artifacts needed to support the base image and the dev container lifecycle. Installation of the Remote Development Extension Pack is required to enable this orchestration within a project repo.

All developers on the team are expected to share and use the dev container definition (.devcontainer directory) in order to spin-up a container. This definition provides consistent tooling for locally developing an application across a team.

The code snippets below demonstrate the common location of a .devcontainer directory and devcontainer.json file within a project repository. They also highlight the correct way to reference a Docker file.

```
$ tree vs-code-remote-try-python  # main repo directory
└──.devcontainers
        ├──Dockerfile
        ├──devcontainer.json
```

```
# devcontainer.json
{
    "name": "Python 3",
    "build": {
        "dockerfile": "Dockerfile",
        "context": "..",
        // Update 'VARIANT' to pick a Python version: 3, 3.6,
3.7, 3.8
        "args": {"VARIANT": "3.8"}
```

```
    },
}
```

For a list of devcontainer.json configuration properties, visit VS Code documentation on dev container properties.

How do I Decide Which Dev Container is Right for my Use Case?

Fortunately, VS Code has a repo gallery of platform specific folders that host dev container definitions (.devcontainer directories) to make getting started with dev containers easier. The code snippet below shows a list of gallery folders that come directly from the VS Code dev container gallery repo:

```
$ tree vs-code-dev-containers  # main repo directory
└──containers
        ├──dotnetcore
        |   └──.devcontainers # dev container
        ├──python-3
        |   └──.devcontainers # dev container
        ├──ubuntu
        |   └──.devcontainers # dev container
        └──....
```

Here are the final high-level steps it takes to build a dev container:

1. Decide which platform you'd like to build a local development tool stack around.
2. Browse the VS Code provided dev container gallery of project folders that target your platform and choose the most appropriate one.
3. Inspect the dev container definitions (.devcontainer directory) of a project for the base image, and the artifacts that support that base

image.

4. Use what you've discovered to begin setting up the dev container as it is, extending it or building your own from scratch.

Going further

There are use cases where you would want to go further in configuring your Dev Container. More details here

---

Last update: August 26, 2024