Customizing Dev Containers

Jan Palášek

2023-11-10

Introduction

Docker is the industry standard in deploying the application. Besides the standard libraries, it allows us to specify the system libraries. What if we could use this capability and also avoid tedious installation of development environments?

Enter Dev Containers: Visual Studio Code's capability to work inside a Docker Container so seamlessly, that you won't even notice it. In this document, we will discuss how to customize them to shape our development environment just as we need it.

Since I (the author) am from the Python world, it is going to be demonstrated on a Python examples. However similar concepts can be applied for other languages.

Basics

To get us over the basics, this video provides a nice detail into what Dev Containers are and how to use them:

https://www.youtube.com/watch?v=b1RavPr_878&ab_channel=VisualStudioCode

In short, we need to create a folder in our repository: `.devcontainer`. Inside the directory, we need to place a new `devcontainer.json`.

```
.devcontainer/
    devcontainer.json
```

`devcontainer.json` specifies the configuration of the new devcontainer, such as:

- What image is going to be used as a base.
- Which vscode extensions are going to be installed.
- Additional vscode settings, such as path to the Python interpreter.

Listing 1 `.devcontainer/devcontainer.json`

```
{
    "name": "basic-dev-container",  // name of the devcontainer
    "image": "mcr.microsoft.com/devcontainers/python:0-3.11",  // image of the dev contain

    "customizations": {
        "vscode": {
            "settings": {},
            "extensions": [
                "ms-python.python",
                "ms-toolsai.jupyter"
            ]
        }
    },

    "postCreateCommand": "python -m pip install requirements.txt"  // command to run after
}
```

Custom Dockerfile

In most of the cases, we want to customize the prepared environment. We might need to install a few additional libraries. In this case, we need to use a custom dockerfile.

We create a new Dockerfile in `.devcontainer` directory. The structure of the Dev Container configuration will now be the following:

```
.devcontainer/
    Dockerfile
    devcontainer.json
```

In the Dockrefile, we first specify the base docker image. Then, we specify installation of all needed dependencies using `RUN` command.

Listing 2 `.devcontainer/Dockerfile`

```
FROM python:3.10.12

# Install custom libraries
RUN apt-get update && apt-get install -y librsvg2-bin
```

> **Tip**
>
> We can do all sorts of configuration in the Dockerfile. We can download files from the internet, modify `~/.bashrc` to alter environment paths. Sky is the limit.

We need to alter our `devcontainer.json` to reference this Dockerfile.

Listing 3 `.devcontainer/devcontainer.json`

```
{
    "name": "dev-container-w-dockerfile",
    "build": {
        "context": "..",
        "dockerfile": "Dockerfile",
    },

    // Use 'postCreateCommand' to run commands after the container is created.
    "postCreateCommand": "python -m pip install -r requirements.txt",

    // Configure tool-specific properties.
    "customizations": {
        "vscode": {
            "extensions": [
                "ms-python.python",
                "ms-toolsai.jupyter",
            ]
        }
    }
}
```

In our Dockerfile, we used `python:3.10.12` instead of the previous image that was prepared from Microsoft `mcr.microsoft.com/devcontainers/python:0-3.11`. This has some consequences:

- Some functionality stops working. For example `remoteUser` field in `devcontainer.json` could specify the user in the new container. However there are only two options asod: `vscode`, `root`.
- We gain possibility to create Dev Container from **any image available**.

The gained flexibility is surely worth it.

Avoiding Root in Dev Container

When we open our Dev Container, we are logged as a root user. We can install any system package as a root user. However as a superuser, we have too much power.

- If some directories were mapped into the Dev Container, we could accidentally delete them.
- If we create a new file, it will act as if it was created by a superuser, not our host user. This might interfere with the permission of our files on the host machine.

There are many more potential problems stemming from this setting. A sensible solution would be to use our host user inside the Dev Container.

We need to create a user inside Docker Image. This user needs to have the same user ID and its group ID as the one on the host.

Listing 4 `.devcontainer/Dockerfile`

```
FROM python:3.10.12

ARG USER_ID
ARG USER_NAME
ARG USER_GID
ARG USER_GNAME

RUN apt-get update && apt-get install -y sudo

# Create user and group
RUN groupadd -g ${USER_GID} ${USER_GNAME}
RUN useradd ${USER_NAME} -u ${USER_ID} -g ${USER_GNAME} -m -s /bin/bash

# Allow sudo (useful for developers to try new system packages)
RUN echo "${USER_NAME} ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers

# "Log on" as the user
USER ${USER_NAME}
```

The Dockerfile accepts arguments `USER_ID`, `USER_NAME`, `USER_GID` (group id of the user's group) and `USER_GNAME`. In the `devcontainer.json`, we need to pass those arguments to the Dockerfile. We will take them from the environment variables.

At last, we need to specify create the local environment variables. We can create and use the following script. It reads the values for `USER_ID` and others and stores them into `~/.bashrc` file.

After creating the init script, we will run the following commands.

```
bash .devcontainer/init.sh
source ~/.bashrc
```

When we create the Dev Container next time, the specified values will be available as the environment variables. And they will be available even after we log off.

Listing 5 `.devcontainer/devcontainer.json`

```
{
    "name": "dev-container-w-hostuser",

    "build": {
        "context": "..",

        "dockerfile": "Dockerfile",
        "args": {

            "USER_ID": "${localEnv:USER_ID}",
            "USER_NAME": "${localEnv:USER_NAME}",

            "USER_GID": "${localEnv:USER_GID}",
            "USER_GNAME": "${localEnv:USER_GNAME}"

        }
    },

    "postCreateCommand": "python -m pip install -r requirements.txt",


    // Configure tool-specific properties.

    "customizations": {
        "vscode": {

            "extensions": [
                "ms-python.python",

                "ms-toolsai.jupyter",
            ]

        }
    }

}
```

Listing 6 `.devcontainer/init.sh`

```
if ! grep -q "# DEVCONTAINER INIT" ~/.bashrc; then
    echo "Initializing..."
    echo "# DEVCONTAINER INIT #" >> ~/.bashrc;
    echo "export USER_NAME=$(id -un)" >> ~/.bashrc;
    echo "export USER_ID=$(id -u)" >> ~/.bashrc;
    echo "export USER_GID=$(id -g)" >> ~/.bashrc;
    echo "export USER_GNAME=$(id -gn)" >> ~/.bashrc;
fi
```

```
echo "Initialization complete..."
```