Developer Guide

Abstract

Setup for development in a Container (VSCode)

Sources: We heavly rely on an official blog article and a tutorial, but make mor narrow choices for wich we apply the concepts laid out there.

Assumptions:

- **VSCode** as IDE
- **local** Docker agent (Windows or MacOS: Docker Desktop)[1]

Use of other IDEs

A similar setup would be possible in IntelliJ. We focus on VSCode for the moment to keep things simple.

For an IDE agnostic setup, we do not check in IDE config directly, however it will be checked in as part of the documentation to be used as a template.

It is a good idea to avoid committing the actual configuration to Git because:

- The configuration contains user specific data that need to be replaced when using the template.
- The individual developer might have the need to further individualize the template:
  - User specific path names/
  - Additional extentions need to be configured.

is also good practice because the individual configuration might vary (e.g. additional plugins, paths on the home system etc).

Why develop in a Container

The setup makes the project less accessible for inexperienced developers as they need to deal with Docker, but it comes with strong advantages:

- **Lack of features under Windows**
  The Python library Enchanted offers limited support for German under Windows.
- **local setup closer to CI/CD and production**
  For an effective dev-cycle, it is best practice for your local dev setup to represent the target environment closely. While esp. Java (Write once run anywhere) abstracts away the OS, the

implementation of compiled languages like RUST might differ between OS systems. Having access to Linux / Unix is therefore a good plattform outside the niche of desktop development.

- **Dependency Management**
  As we depend on a range of dependencies (Tesseract_ORC, Python, a multitude of python libs sometimes requiring further dependencies like dictionaries) the following problems we face are alleviated having them checked into the project:
  - Setup and maintenance of dev or production systems.
  - code becomes clustered with parameters, heuristics and conditions that deal with varying local dependencies (e.g. path names)

Setup of the IDE

Install and Configure Plugin from the Marketplace

Load the plugin "Dev Containers" published by Microsoft and put the following config into `/.devcontainer/devcontainer.json`.

```
{
  "name": "Dev Container",
  "dockerFile": "../docker/Dockerfile",
  "workspaceFolder": "/workspace",
  "context": "..",
  "mounts": [
    "source=${localWorkspaceFolder},target=/workspace,type=bind",
    "source=C:/Users/<YourUsername>/.ssh,target=/root/.ssh,type=bind"
  ],
  "settings": {
    "terminal.integrated.shell.linux": "/bin/bash"
  },
  "extensions": [
    "ms-python.python",
    "ms-azuretools.vscode-docker",
    "GitHub.vscode-pull-request-github",
    "asciidoctor.asciidoctor-vscode"
  ],
  "postCreateCommand": "chmod +x /workspace/scripts/setup_ssh_git.sh &&
/workspace/scripts/setup_ssh_git.sh && git config --global user.name '<Your Name>' &&
git config --global user.email '<your.email@example.com>' && python3 -m venv
/workspace/venv && /workspace/venv/bin/pip install --no-cache-dir -r
/workspace/requirements.txt",
  "remoteUser": "root"
}
```

IMPORTANT	change `<YourUsername>`, `<your.email@example.com>`, and `<Your Name>` into the to your Windows login, your email and the User that should appear on the Git commits!

Setup Dockerfile Dependencies

The fact, that our project requires a complex setup on the target environment was one motivation, to move to a container. Conclusively the Dockerfiles depend on secondary files

Python requirements

`requirements.txt` is a textfile that contains the python packages that need to be installed with 'pip'

```
pyenchant
tqdm
hunspell
```

Tesseract config

`tessdata.txt` is a textfile that contains additiona training files optimised for the relevant languages

```
https://github.com/tesseract-ocr/tessdata/raw/main/eng.traineddata
https://github.com/tesseract-ocr/tessdata/raw/main/spa.traineddata
```

Setup Dockerfile

The `Dockerfile` represents the development environment. It is common practice to use a different Docker file to generate the production setup because the two environments have drastically different requirements: While development environments need various tools and libraries for editing, debugging, and testing code, production environments need a lean and optimized setup for performance and security.

Dockerfile for development (`Dockerfile.dev`)

```
# Use an official Ubuntu as a parent image
FROM ubuntu:latest

# Install required tools
RUN apt-get update && apt-get install -y \
    build-essential \
    curl \
    wget \
    git \
    vim \
    python3 \
    python3-pip \
    python3-venv \
    tesseract-ocr \
    asciidoctor \
    bash
```

```
# Set the working directory for the project
WORKDIR /workspace

# Copy the tessdata list
COPY ../tessdata.txt /workspace/tessdata.txt

# Download Tesseract language data files
RUN xargs -n 1 wget -P /usr/share/tesseract-ocr/4.00/tessdata/ <
/workspace/tessdata.txt \
    && rm -rf /var/lib/apt/lists/*

# Copy the source code
COPY ../src /workspace/src

# Copy the requirements file
COPY ../requirements.txt /workspace

# Create and activate a virtual environment
RUN python3 -m venv /workspace/venv

# Install any Python dependencies in the virtual environment
RUN /workspace/venv/bin/pip install --no-cache-dir -r /workspace/requirements.txt

# Set environment variables for the virtual environment
ENV PATH="/workspace/venv/bin:$PATH"

# Run bash by default
CMD ["bash"]
```

Dockerfile for production (`Dockerfile.prod`)

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# Set the working directory
WORKDIR /app

# Copy the source code
COPY ./src /app/src

# Copy the requirements file
COPY requirements.txt /app

# Copy the tessdata list
COPY tessdata.txt /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r /app/requirements.txt

# Install Tesseract OCR and necessary language data
```

```
RUN apt-get update && apt-get install -y tesseract-ocr \
    && apt-get install -y wget \
    && xargs -n 1 wget -P /usr/share/tesseract-ocr/4.00/tessdata/ < /app/tessdata.txt
\
    && rm -rf /var/lib/apt/lists/*

# Set environment variables for the virtual environment
ENV PATH="/app/venv/bin:$PATH"

# Set the entrypoint to ensure additional arguments are passed to the Python script
ENTRYPOINT ["python", "/app/src/main_script.py"]
```

The base image is a lightweight version of Python

```
FROM python:3.9-slim
```

Installation of the Python dependencies

```
RUN pip install --no-cache-dir -r /app/requirements.txt
```

Installation of the Tessdata dependencies

- Install `wget`

```
RUN apt-get update && apt-get install -y tesseract-ocr \
    && apt-get install -y wget \
```

- Download Language Data Files

These lines download the English (eng.traineddata) and Spanish (spa.traineddata) language data files from the Tesseract GitHub repository and place them in the appropriate directory (`/usr/share/tesseract-ocr/4.00/tessdata/`)

```
&& wget -P /usr/share/tesseract-ocr/4.00/tessdata/ https://github.com/tesseract-
ocr/tessdata/raw/main/eng.traineddata \
&& wget -P /usr/share/tesseract-ocr/4.00/tessdata/ https://github.com/tesseract-
ocr/tessdata/raw/main/spa.traineddata \
...
```

- Streamline the download

Use `xargs` to read each URL from `tessdata.txt` and download the corresponding file(s). This makes it easy to manage and update the list of languages without modifying the Dockerfile directly.

```
xargs -n 1 wget -P /usr/share/tesseract-ocr/4.00/tessdata/ < /app/tessdata.txt \
```

- This removes the package lists to reduce the image size.

```
rm -rf /var/lib/apt/lists/*
```

Start the Container

In the Dialog `Ctrl+Shift+P` Choose the option Dev Container: Open Folder in Container

Prepare VS Code for Dev Containers

```
are found in the official documentation for VSCode.
```

The following description will point out specific points relevant to the project but does not aspire to provide an exhaustive explanation.

Dev Container Configuration

The `.devcontainer` directory contains configuration files for the VS Code Dev Container. The `devcontainer.json` file specifies how the container should be built and what directories should be mounted.

Key settings in `devcontainer.json`: - **dockerFile**: Points to the Dockerfile in the `docker` directory. - **workspaceFolder**: Sets the working directory inside the container. - **mounts**: Binds the local directories to the container directories. - **settings**: Configures the terminal shell to use bash. - **extensions**: Installs the necessary VS Code extensions inside the container. - **postCreateCommand**: Installs the Python dependencies listed in `requirements.txt`. - **remoteUser**: Sets the user as `root` to ensure necessary permissions.

Dockerfile

The Dockerfiles in the `/docker` directory sets up...

- ... a container to be used by VSCode as a dev environment (`Dockerfile`)
- ... a container to be used for the production build (`Dockerfile.prod`)

They have obviously things in common as they share many dependencies but the dev container starts up with bash while the productio version imediately calls our python script on startup.

Project Structure from inside and outside the development Container

- **data**

store sample input files for development.

- **Documentation (`/docs`)**
  project documentation in asciidoc format.
- **scripts (`/scripts/`)**
  scripts for development (windows command shell and unix bash).
- **Soruce Code (`/src/`)**
  project source code.
- **Target directory (`/target/`)**
  build artifacts (incl. generated PDFs).
- **Docker Setup (`/docker/`)**
  dockerfiles for dev and production containers.
- **VC Code Setup for Container development (/.devcontainer/)**
  configuration of the develompent container for VS-Code.
- **VC Code Setup for Container development (/.github/)**
  configuration github pipelines.

These folders are inaccessible for the container - you have to edit them locally

- **project root (/)** contains GITIGNORE and README.adoc
- "Readme resources (`/README/`)" contains resources associated with the README file (e.g. images)

If you want to make a new folder accessible to the container, it has to be lsited among the mount points in `/.devcontainer/devcontainer.json`. as long as the container side of the mout point is below `/workspace/` a change to the `Dockerfile` is not necessary.

Using the scripts during development

For typical development activities we developed shell-scripts, see `/scripts/`

Building and starting the development container is normally done by the IDE - you need these scripts, if you want to perform something directly on the comand line, not within the IDE.

you will need the bash (**.sh) or the windows shell (**.cmd) version of the script, depending on wether you are within the container or outside when you perform the action. Take care not to use windows powershell, since the scripts are optimised for the normal comand shell.

Managing the connection between the IDE and the development container

The connection between the IDE and the development container is managed by an extention

- Start Docker Daemon (usually by starting Docker Desktop)
- Make sure the service is running: `docker info ` (optional)
- Make sure there is no container lingering from the last session: `docker ps -a ` (optional)
- Stop and remove a lingering container: `docker stop <container_id>` and `docker rm <container_id>`

- Open VS Code will start in local mode
- Switch to Container mode: `Str + Shift + P` and type "Dev Containers: Open Folder in Container"
- Switch back to Local mode `Str + Shift + P` and type "Dev Containers: Open Folder Locally"
- Perform a container restart, e.g. after a config change `Str + Shift + P` and type "Dev Containers: Reopen Folder Locally"
- Trouble-Shooting: If you get tangled. Stop VS Code and make sure there is no process running (Code), stop all containers and start afresh.

Adding Extentions to VSCode

If you install a new extension in VS Code while using a Dev Container, it will not automatically persist across container restarts or rebuilds unless specified in the devcontainer.json configuration. To ensure that the extension is always available in your Dev Container, you need to add it to the extensions list in your devcontainer.json. This can be done directly from the Marketplace view via the context menu.

You might also have to restart VS Code to complete the process.

Using Git and Dev Containers

Manage Git from Local Machine

Pros and Cons

- Pro
  - simpler
  - avoids the need to duplicate SSH key management in the container

Setup

1. Generate or Use Existing SSH Keys:

   ```
   If you don't already have SSH keys, generate them on your local machine:
   ```

   ```
   Copy code
   ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
   ```

2. Add the SSH key to your GitHub account

   ```
   Copying the public key content (~/.ssh/id_rsa.pub)
   ```

3. Configure VS Code to Use Local Git

   ```
   When using VS Code on your local machine, it will use the SSH keys from your local
   ```

```
.ssh directory.
```

Manage Git from Within the Container

Pros and Cons

- Pro
  - more comfort and consistency
- Con
  - SSH key management and git configuration in the container necessary

Implications

If we want to manage git from within the container it does not make sense anymore to rely on binding seleced folders in our project to the container because we need access to the project as a whole and `.gitignore` in particular, which resides in the (otherwise inaccessible) root dir. Instead we should mount the whole project directory in one go.

We have to store the ssh keys and they have to be in a location separate from the project.

Setup

update Dockerfile

```
# Use an official Ubuntu as a parent image
FROM ubuntu:latest

# Install required tools
RUN apt-get update && apt-get install -y \
    build-essential \
    curl \
    wget \
    git \
    vim \
    python3 \
    python3-pip \
    python3-venv \
    tesseract-ocr \
    asciidoctor \
    bash

# Set the working directory for the project
WORKDIR /workspace

# Create and activate a virtual environment
RUN python3 -m venv /workspace/venv

# Set environment variables for the virtual environment
ENV PATH="/workspace/venv/bin:$PATH"
```

```
# Create a separate directory for SSH keys
RUN mkdir /root/.ssh

# Run bash by default
CMD ["bash"]
```

Update `devcontainer.json` to mount the ssh and the project as a whole and mount the ssh directory thus making the keys accessible in the container.

```
 {
  "name": "Dev Container",
  "dockerFile": "../docker/Dockerfile",
  "workspaceFolder": "/workspace",
  "context": "..",
  "mounts": [
    "source=${localWorkspaceFolder},target=/workspace,type=bind",
    "source=${env:HOME}/.ssh,target=/root/.ssh,type=bind"
  ],
  "settings": {
    "terminal.integrated.shell.linux": "/bin/bash"
  },
  "extensions": [
    "ms-python.python",
    "ms-azuretools.vscode-docker",
    "GitHub.vscode-pull-request-github",
    "asciidoctor.asciidoctor-vscode"  // Add this line for AsciiDoc extension
  ],
  "postCreateCommand": "/workspace/venv/bin/pip install --no-cache-dir -r
/workspace/requirements.txt",
  "remoteUser": "root"
}
```

Configure ssh in the container

Ensure the SSH configuration inside the container recognizes your keys:

```
eval "$(ssh-agent -s)"
ssh-add /root/.ssh/id_rsa
```

Test the SSH connection to GitHub:

```
ssh -T git@github.com
```

Using Git and SSH in VS Code VS Code SSH Configuration:

If you are using VS Code locally with the Remote - Containers extension, it will use the SSH keys

from your local .ssh directory. If you are working inside a container, ensure the container has access to the SSH keys as described above. VS Code Git Integration:

You can use the Source Control view in VS Code to perform Git operations. Ensure the SSH key used for authentication is accessible (either from the local machine or within the container, depending on your setup).

{ "name": "Dev Container", "dockerFile": "../docker/Dockerfile", "workspaceFolder": "/workspace", "context": "..", "mounts": ["source=${localWorkspaceFolder}/src,target=/workspace/src,type=bind", "source=${localWorkspaceFolder}/data,target=/workspace/data,type=bind", "source=${localWorkspaceFolder}/docs,target=/workspace/docs,type=bind", "source=${localWorkspaceFolder}/scripts,target=/workspace/scripts,type=bind", "source=${localWorkspaceFolder}/docker,target=/workspace/docker,type=bind", "source=${localWorkspaceFolder}/target,target=/workspace/target,type=bind", "source=${localWorkspaceFolder}/requirements.txt,target=/workspace/requirements.txt,type=bind" , "source=${localWorkspaceFolder}/.github,target=/workspace/.github,type=bind", "source=${localWorkspaceFolder}/.github,target=/workspace/.git,type=bind", "source=${localWorkspaceFolder}/.devcontainer,target=/workspace/.devcontainer,type=bind"], "settings": { "terminal.integrated.shell.linux": "/bin/bash" }, "extensions": ["ms-python.python", "ms-azuretools.vscode-docker", "asciidoctor.asciidoctor-vscode", "mhutchie.git-graph"], "postCreateCommand": "/workspace/venv/bin/pip install --no-cache-dir -r /workspace/requirements.txt", "remoteUser": "root" }

RUN pip3 install --no-cache-dir -r requirements.txt || true

ssh -T git@github.com

[1] As detailed in the article use of a remote Docker host or even a Kubernetes Cluster would be other viable options