Using DPF Through Docker¶

You can run DPF within a container on any OS using Docker.

Advantages of running DPF in a containerized environment, such as Docker or Singularity, include:

- Running in a consistent environment regardless of the host operating system
- Offering portability and ease of install
- Supporting large-scale cluster deployment using Kubernetes
- Providing genuine application isolation through containerization

Installing the DPF Image¶

Using your GitHub credentials, you can download the Docker image in the DPF-Core GitHub repository.

If you have Docker installed, you can get started by authorizing Docker to access this repository using a GitHub personal access token with `packages read` permissions. For more information, see <https://help.github.com/en/github/authenticating-to-github/creating-a-personal-access-token>`_.

Save the token to a file:

```
echo XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX > GH_TOKEN.txt
```

This lets you send the token to Docker without leaving the token value in your history.

Next, authorize Docker to access the repository:

```
GH_USERNAME=<my-github-username>
cat GH_TOKEN.txt | docker login docker.pkg.github.com -u $GH_USERNAME --password-stdin
```

You can now launch DPF directly from Docker with a short script or directly from the command line.

```
docker run -it --rm -v `pwd`:/dpf -p 50054:50054 docker.pkg.github.com/pyansys/dpf-
core/dpf:v2021.1
```

Note that this command shares the current directory to the `/dpf` directory contained within the image. This is necessary as the DPF binary within the image must access the files within the image itself. Any files that you want to have DPF read must be placed in the `pwd`. You can map other directories as needed, but these directories must be mapped to the `/dpf` directory for the server to see the files you want it to read.

Using the DPF Container from Python¶

Normally `ansys.dpf.core` attempts to start the DPF server at the first usage of a DPF class. If you do not have Ansys installed and simply want to use the Docker image, you can override this behavior by connecting to the DPF server on the port you mapped:

```
from ansys.dpf import core as dpf_core

# uses 127.0.0.1 and port 50054 by default
dpf_core.connect_to_server()
```

If you want to avoid having to run `connect_to_server` at the start of every script, you can tell `ansys.dpf.core` to always attempt to connect to DPF running within the Docker image by setting environment variables:

On Linux:

```
export DPF_START_SERVER=False
export DPF_PORT=50054
```

On Windows:

```
set DPF_START_SERVER=False
set DPF_PORT=50054
```

The environment variable `DPF_PORT` is the port exposed from the DPF container. It should match the first value within the `-p 50054:50054` pair.

The environment variable `DPF_START_SERVER` tells `ansys.dpf.core` not to start an instance but rather look for the service running at `DPF_IP` and `DPF_PORT`. If these environment variables are undefined, they default to 127.0.0.1 and 50054 for `DPF_IP` and `DPF_PORT` respectively.

Created using Sphinx 4.3.1.