docker_interface Documentation

Till Hoffmann

Apr 29, 2020

CONTENTS:

Docker Interface (DI) is a declarative interface for building images and running commands in containers using Docker.

CHAPTER ONE

INTRODUCTION

Docker provides a containerised runtime that enables easy and reproducible deployment of applications to production. Unfortunately, these applications are often developed using the local environment of the developer such that it can be difficult to reproduce the results on another machine. Using Docker as a development environment is possible in principle but plagued by problems in practice. For example,

- mounting a folder from the host in the container can cause permission problems,
- ports need to be manually forwarded to run Jupyter notebook servers,
- or credentials are not available inside the container.

These issues can be addressed directly by modifying the arguments passed to the Docker command line interface (CLI), but the resulting commands can be formidable. Docker Interface allows users to define a Docker command declaratively in a configuration file rather than having to remember to type out all required arguments on the command line. In short, Docker Interface is a translator from a command declaration to a Docker command.

1.1 Installing Docker interface

You can install Docker Interface using the following `pip` command (you need a python3 interpreter).

```
pip install docker-interface
```

To check that Docker Interface was installed successfully, run

```
di --help
```

1.2 Using Docker Interface

Docker Interface will attempt to locate a configuration file `di.yml` in the current working directory. A basic configuration (as a YAML or JSON file) might look like so.

```
docker: docker   # The docker command to use, e.g. nvidia-docker
workspace: .     # The workspace path (relative to the directory containing the␣
↪configuration)
```

All paths in the configuration are relative to the `workspace`. The values shown above are default values and you can omit them unless you want to change them.

Docker Interface supports two commands:

- build to build a Docker image,

- and run to execute a command inside a Docker container.

Information that is relevant to a particular command is stored in a corresponding section of the configuration file. For example, you can run the `bash` shell in the latest `ubuntu` like so: First, create the following configuration file.

```
run:
  image: ubuntu
```

Second, run `di run bash` from the command line. In contrast to `docker run ubuntu bash`, the `di` command will open an interactive shell because it starts the container interactively if it detects that Docker Interface was launched interactively. By default, it will also create an ephemeral container which is deleted as soon as you log out of the shell.

Before delving into the plugin architecture that powers Docker Interface, let us consider a simple example for building your own Docker image. Create a `Dockerfile` with the following content

```
FROM python
RUN pip install ipython
```

and modify your `di.yml` configuration to read:

```
build:
  tag: my-ipython
```

Running `di build` from the command line will build your image, and `di run ipython` will run the `ipython` command inside the container. Unless otherwise specified, Docker Interface uses the image built in the `build` step to start a new container when you use the `run` command. Note: the `run` command also sets the environment variable `DOCKER_INTERFACE=true` which allows you to dynamically detect when running under the control of `di`.

A comprehensive list of variables that can be set in the `di.yml` configuration can be found in the *Plugin reference*.

CHAPTER TWO

PLUGINS

Docker Interface is a simple framework leveraging a suite of plugins to do most of its work. Each plugin is a `python` class that defines at least two static attributes:

- `COMANDS` is a sequence of commands such as `run` or `build` and defines for which command the plugin should be active. Setting `COMMANDS` to `all` will enable the plugin for all commands.
- `ORDER` is an integer that determines the order of execution with lower numbers being executed earlier.

Furthermore, each plugin can additionally define:

- `ENABLED` (defaults to `True`) which indicates whether the plugin is enabled. Set `ENABLED` to `False` if you want a plugin to be disabled by default.
- `SCHEMA` (defaults to `{}`) is a JSON schema definition that is specific to the plugin. The Docker Interface configuration is validated against the union of schemas defined by all enabled plugins.

2.1 How plugins work

Each plugin has two methods used by Docker Interface:

- `add_arguments(parser)` is called for each enabled plugin before Docker Interface attempts to parse the command line arguments. Each plugin may add arbitrary arguments to the `parser` of the command line interface as long as they do not interfere with one another.
- `apply(configuration, schema, args)` is called for each plugin after `args` have been parsed. The `schema` passed to the plugins is the union of all plugins' schemas. Finally, `configuration` is the configuration returned by the `apply` method of a plugin with lower `ORDER`. The plugin may modify the configuration (as `UserPlugin` does), execute a Docker command (as `BuildExecutePlugin` does), or run any other python code.

2.2 Enable and disabling plugins

Unless otherwise specified, a plugin is enabled if and only if its class-level attribute `ENABLED` is `TRUE`. But you can specify which plugins to enable or disable in the configuration like so.

```
plugins:
  - user
  - homedir
```

The above configuration will enable only the `user` and the `homedir` plugins. Alternatively, you can specify which plugins to enable or disable explicitly.

```yaml
plugins:
  enable:
    - user
  disable:
    - homedir
```

Which will enable the `user` plugin, disable the `homedir` plugin, and leave all other plugins unchanged.

2.3 Schema validation

Docker Interface validates the configuration against the union of schemas defined by all enabled plugins. Different plugins may define the same schema as long as the definitions are consistent with one another. Schema definitions are also the preferred way to provide default values for the configuration. For example, the schema for the `BasePlugin` responsible for loading the configuration, handling the workspace, and other global variables looks like so.

```json
{
    "properties": {
        "workspace": {
            "type": "string",
            "description": "Path defining the DI workspace (absolute or relative to␣
↪the URI of the configuration document). All subsequent path definitions must be␣
↪absolute or relative to the `workspace`."
        },
        "docker": {
            "type": "string",
            "description": "Name of the docker CLI.",
            "default": "docker"
        }
    },
    "required": [
        "docker",
        "workspace"
    ]
}
```

The configuration can define the parameters `docker` and `workspace`, and we provide a default value for `docker`. We have omitted some properties for easier readability. If your plugin adds new configuration values, it should define a `SCHEMA`.

CHAPTER

THREE

PLUGIN REFERENCE

This document lists all plugins in order of execution.

3.1 BasePlugin

Load or create a default configuration and set up logging.

3.1.1 Properties

- workspace (`string`): Path defining the DI workspace (absolute or relative to the URI of this document). All subsequent path definitions must be absolute or relative to the *workspace*.
- docker (`string`): Name of the docker CLI (default: `docker`).
- log-level (`string`): (default: `info`).
- dry-run (`boolean`): Whether to just construct the docker command.
- status-code (`integer`): status code returned by docker.
- plugins: .

3.2 GoogleContainerRegistryPlugin

Configure docker authorization for Google services such as Google Container Registry.

3.3 WorkspaceMountPlugin

Mount the workspace inside the container.

3.3.1 Properties

- **run: .**
  - workspace-dir (`string`): Path at which to mount the workspace in the container (default: `/workspace`).
  - workdir (`string`): (default: `#{workspace-dir}`).

3.4 UserPlugin

Share the host user id and group id with the container.

The plugin provides the following additional variables for substitution:

- `user/name`: Name of the user on the host.
- `user/uid`: User id of the user on the host.
- `group/name`: Name of the user group on the host.
- `group/gid`: Group id of the user group on the host.

3.4.1 Properties

- **run: .**
  - user (`string`): Username or UID (format: <name|uid>[:<group|gid>]).

3.5 HomeDirPlugin

Mount a home directory placed in the current directory.

3.6 GoogleCloudCredentialsPlugin

Mount Google Cloud credentials in the Docker container.

3.7 BuildConfigurationPlugin

Configure how to build a docker image.

3.7.1 Properties

- **build: .**
  - path (`string`): Path of the build context (default: `#{/workspace}`).
  - tag (`string`): Name and optionally a tag in the 'name:tag' format (default: `docker-interface-image`).
  - file (`string`): Name of the Dockerfile (default: `#{path}/Dockerfile`).
  - build-arg (`object`): Set build-time variables.
  - no-cache (`boolean`): Do not use cache when building the image.
  - quiet (`boolean`): Suppress the build output and print image ID on success.
  - cpu-shares (`integer`): CPU shares (relative weight).
  - memory (`string`): Memory limit.

3.8 RunConfigurationPlugin

Configure how to run a command inside a docker container.

3.8.1 Properties

- **run: .**
  - image (`string`): Image to derive the container from (default: `#{/build/tag}`).
  - env (`object`): Set environment variables (use *null* to forward environment variables).
  - env-file (`array`): Read in a file of environment variables.
  - mount (`array`): Attach a filesystem mount to the container.
  - publish (`array`): Publish a container's port(s), or range(s) of ports, to the host.
  - runtime (`string`): Runtime to use for this container.
  - tmpfs (`array`): Mount a tmpfs directory.
  - cmd (`array`): Command to execute inside the container.
  - tty (`boolean`): Allocate a pseudo-TTY.
  - cpu-shares (`integer`): CPU shares (relative weight).
  - name (`string`): Assign a name to the container.
  - network (`string`): Connect a container to a network (default "default").
  - label (`array`): Set meta data on a container.
  - rm (`boolean`): Automatically remove the container when it exits (default: `True`).
  - privileged (`boolean`): Give extended privileges to this container.
  - memory (`string`): Memory limit.
  - interactive (`boolean`): Keep STDIN open even if not attached.
  - entrypoint (`string`): Overwrite the default ENTRYPOINT of the image.
  - workdir (`string`): Working directory inside the container.
  - user (`string`): Username or UID (format: <name|uid>[:<group|gid>]).
  - group-add (`array`): Additional groups to run as.
  - gpus (`string`): GPU devices to add to the container ('all' to pass all GPUs).

3.9 JupyterPlugin

Forward the port required by Jupyter Notebook to the host machine and print a URL for easily accessing the notebook server.

3.10 SubstitutionPlugin

Substitute variables in strings.

String values in the configuration document may

- reference other parts of the configuration document using `#{path}`, where `path` may be an absolute or relative path in the document.
- reference a variable using `${path}`, where `path` is assumed to be an absolute path in the `VARIABLES` class attribute of the plugin.

By default, the plugin provides environment variables using the `env` prefix. For example, a value could reference the user name on the host using `${env/USER}`. Other plugins can provide variables for substitution by extending the `VARIABLES` class attribute and should do so using a unique prefix.

3.11 ValidationPlugin

Validate the configuration document.

3.12 BuildPlugin

Build a docker image.

3.13 RunPlugin

Run a command inside a docker container.

CHAPTER

FOUR

EXAMPLES

This document lists example use cases for Docker Interface that are available on GitHub. Additional, comprehensive examples can be found in the tests.

4.1 cython

This simple example addresses some of the difficulties associated with using cython and `di` together: the cython code is compiled when the Docker image is built. But when the workspace is mounted in the container, the binaries are hidden and the code can no longer be executed. We thus use pyximport to compile the cython code on the fly. See `cython_example/__init__.py` for details.

4.2 notebook

This example demonstrates automatic port forwarding for the Jupyter notebook using `di`. Port forwarding is implemented using the `JupyterPlugin` in `docker_interface/plugins/python.py`.

4.3 ports

This example demonstrates how to forward ports using `di`'s declarative syntax.

4.4 env

This example demonstrates that a default environment variable is set whe using `di run`.

CHAPTER FIVE

SCHEMA

The document below provides a comprehensive schema definition for Docker Interface.

```
{
    "additionalProperties": false,
    "$schema": "http://json-schema.org/draft-04/schema",
    "properties": {
        "workspace": {
            "type": "string",
            "description": "Path defining the DI workspace (absolute or relative to␣
↪the URI of this document). All subsequent path definitions must be absolute or␣
↪relative to the `workspace`."
        },
        "docker": {
            "type": "string",
            "description": "Name of the docker CLI.",
            "default": "docker"
        },
        "log-level": {
            "type": "string",
            "enum": [
                "debug",
                "info",
                "warning",
                "error",
                "critical",
                "fatal"
            ],
            "default": "info"
        },
        "dry-run": {
            "type": "boolean",
            "description": "Whether to just construct the docker command.",
            "default": false
        },
        "status-code": {
            "type": "integer",
            "description": "status code returned by docker"
        },
        "plugins": {
            "oneOf": [
                {
                    "type": "array",
                    "description": "Enable the listed plugins and disable all plugins␣
↪not listed.",
```

(continues on next page)

(continued from previous page)

```
                    "items": {
                        "type": "string"
                    }
                },
                {
                    "type": "object",
                    "properties": {
                        "enable": {
                            "type": "array",
                            "description": "Enable the listed plugins.",
                            "items": {
                                "type": "string"
                            }
                        },
                        "disable": {
                            "type": "array",
                            "description": "Disable the listed plugins.",
                            "items": {
                                "type": "string"
                            }
                        }
                    },
                    "additionalProperties": false
                }
            ]
        },
        "run": {
            "properties": {
                "workspace-dir": {
                    "type": "string",
                    "description": "Path at which to mount the workspace in the␣
↪container.",
                    "default": "/workspace"
                },
                "workdir": {
                    "type": "string",
                    "default": "#{workspace-dir}",
                    "description": "Working directory inside the container"
                },
                "user": {
                    "type": "string",
                    "description": "Username or UID (format: <name|uid>[:<group|gid>])
↪"
                },
                "entrypoint": {
                    "type": "string",
                    "description": "Overwrite the default ENTRYPOINT of the image"
                },
                "interactive": {
                    "type": "boolean",
                    "description": "Keep STDIN open even if not attached"
                },
                "publish": {
                    "type": "array",
                    "description": "Publish a container's port(s), or range(s) of␣
↪ports, to the host.",
                    "items": {
```

(continues on next page)

(continued from previous page)

```
                        "type": "object",
                        "properties": {
                            "ip": {
                                "type": "string",
                                "description": ""
                            },
                            "host": {
                                "anyOf": [
                                    {
                                        "type": "number"
                                    },
                                    {
                                        "type": "string",
                                        "pattern": "\\d+-\\d+"
                                    }
                                ],
                                "description": "Port (e.g. `8000`) or range of ports␣
↪(e.g. `8000-8100`) on the host."
                            },
                            "container": {
                                "anyOf": [
                                    {
                                        "type": "number"
                                    },
                                    {
                                        "type": "string",
                                        "pattern": "\\d+-\\d+"
                                    }
                                ],
                                "description": "Port (e.g. `8000`) or range of ports␣
↪(e.g. `8000-8100`) on the container."
                            }
                        },
                        "required": [
                            "container"
                        ],
                        "additionalProperties": false
                    }
                },
                "privileged": {
                    "type": "boolean",
                    "description": "Give extended privileges to this container",
                    "default": false
                },
                "rm": {
                    "type": "boolean",
                    "description": "Automatically remove the container when it exits",
                    "default": true
                },
                "image": {
                    "type": "string",
                    "description": "Image to derive the container from.",
                    "default": "#{/build/tag}"
                },
                "env": {
                    "type": "object",
                    "description": "Set environment variables (use `null` to forward␣
↪environment variables).",
```

(continues on next page)

(continued from previous page)

```
                    "additionalProperties": {
                        "type": [
                            "string",
                            "null"
                        ]
                    }
                },
                "runtime": {
                    "type": "string",
                    "description": "Runtime to use for this container."
                },
                "tmpfs": {
                    "type": "array",
                    "description": "Mount a tmpfs directory",
                    "items": {
                        "type": "object",
                        "properties": {
                            "destination": {
                                "type": "string",
                                "description": "Absolute mount path in the container."
                            },
                            "options": {
                                "type": "array",
                                "description": "Mount options for the temporary file␣
↪system.",
                                "items": {
                                    "type": "string"
                                }
                            },
                            "size": {
                                "type": "integer",
                                "description": "Size of the tmpfs mount in bytes."
                            },
                            "mode": {
                                "type": "integer",
                                "description": "File mode of the tmpfs in octal."
                            }
                        },
                        "required": [
                            "destination"
                        ],
                        "additionalProperties": false
                    }
                },
                "network": {
                    "type": "string",
                    "description": "Connect a container to a network (default \
↪"default\")"
                },
                "memory": {
                    "type": "string",
                    "description": "Memory limit"
                },
                "name": {
                    "type": "string",
                    "description": "Assign a name to the container"
                },
```

(continues on next page)

(continued from previous page)

```
                "env-file": {
                    "type": "array",
                    "description": "Read in a file of environment variables.",
                    "items": {
                        "type": "string"
                    }
                },
                "mount": {
                    "type": "array",
                    "description": "Attach a filesystem mount to the container.",
                    "items": {
                        "type": "object",
                        "properties": {
                            "type": {
                                "type": "string",
                                "enum": [
                                    "bind",
                                    "tmpfs",
                                    "volume"
                                ]
                            },
                            "source": {
                                "type": "string",
                                "description": "Volume name or path on the host."
                            },
                            "destination": {
                                "type": "string",
                                "description": "Absolute mount path in the container."
                            },
                            "readonly": {
                                "type": "boolean",
                                "description": "Whether to mount the volume read-only.
↪"
                            }
                        },
                        "required": [
                            "type",
                            "destination"
                        ],
                        "additionalProperties": false
                    }
                },
                "label": {
                    "type": "array",
                    "description": "Set meta data on a container"
                },
                "group-add": {
                    "type": "array",
                    "description": "Additional groups to run as.",
                    "items": {
                        "type": "string"
                    }
                },
                "gpus": {
                    "type": "string",
                    "description": "GPU devices to add to the container␣
↪(\u2018all\u2019 to pass all GPUs)"
```

(continues on next page)

(continued from previous page)

```
                },
                "cpu-shares": {
                    "type": "integer",
                    "description": "CPU shares (relative weight)",
                    "minimum": 0,
                    "maximum": 1024
                },
                "cmd": {
                    "type": "array",
                    "description": "Command to execute inside the container.",
                    "items": {
                        "type": "string"
                    }
                },
                "tty": {
                    "type": "boolean",
                    "description": "Allocate a pseudo-TTY"
                }
            },
            "additionalProperties": false
        },
        "build": {
            "properties": {
                "path": {
                    "type": "string",
                    "description": "Path of the build context.",
                    "default": "#{/workspace}"
                },
                "tag": {
                    "type": "string",
                    "description": "Name and optionally a tag in the 'name:tag'␣
↪format.",
                    "default": "docker-interface-image"
                },
                "file": {
                    "type": "string",
                    "description": "Name of the Dockerfile.",
                    "default": "#{path}/Dockerfile"
                },
                "build-arg": {
                    "type": "object",
                    "description": "Set build-time variables.",
                    "additionalProperties": {
                        "type": "string"
                    }
                },
                "no-cache": {
                    "type": "boolean",
                    "description": "Do not use cache when building the image"
                },
                "quiet": {
                    "type": "boolean",
                    "description": "Suppress the build output and print image ID on␣
↪success"
                },
                "cpu-shares": {
                    "type": "integer",
```

(continues on next page)

(continued from previous page)

```
                    "description": "CPU shares (relative weight)",
                    "minimum": 0,
                    "maximum": 1024
                },
                "memory": {
                    "type": "string",
                    "description": "Memory limit"
                }
            },
            "required": [
                "tag",
                "path",
                "file"
            ],
            "additionalProperties": false
        }
    },
    "required": [
        "workspace",
        "docker"
    ],
    "title": "Declarative Docker Interface (DI) definition."
}
```

CHAPTER SIX

INDICES AND TABLES

- genindex
- modindex
- search