ADAMS

Advanced **D**ata mining **A**nd **M**achine learning **S**ystem

Module: adams-docker



Peter Reutemann

November 21, 2026

©2023



THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato



Except where otherwise noted, this work is licensed under
http://creativecommons.org/licenses/by-sa/4.0/

Contents

1 Introduction **7**

2 Flow **9**
2.1 Simple docker commands 9

Bibliography **11**

List of Figures

Chapter 1

Introduction

Docker[2] offers OS-level virtualization for software using containers. This allows the encapsulation of software that would otherwise interfer with each other.

Chapter 2

Flow

The following standalones are available:

- *SimpleDockerConnection* – determines the context in which to run the docker commands, e.g., what docker registry to use.

The following sources are available:

- *SimpleDockerCommand* – executes the selected docker command plugin, requires presence of a *SimpleDockerConnection* (see **??**).

The following transformers are available:

- *SimpleDockerCommand* – passes the incoming data as additional arguments to the selected docker command plugin, requires presence of a *SimpleDockerConnection* (see **??**).

The following conversions are available:

- *LocalPathToContainerPath* – using the directory mappings defined by a *SimpleDockerConnection*, the local path passing through gets converted into one to be used within the container.

2.1 Simple docker commands

simpledockercommands) The following source commands are supported:

- *BuildImage* – for building an image (`docker image build`)
- *ExecContainer* – for executing a command in a running container (`docker container exec`)
- *Generic* – for running an arbitrary docker sub-command
- *Info* – outputs information about the docker environment (`docker info`)
- *ListContainers* – lists container IDs (`docker container ls`)
- *ListImages* – lists image IDs (`docker image ls`)
- *PruneContainers* – prunes containers (`docker container prune`)
- *PruneImages* – prunes images (`docker image prune`)
- *Pull* – pulls a specific image (`docker pull`)
- *Push* – pushes the specified image (`docker push`)

- *Run* – executes the specified image (`docker run`)

The following transformer commands are supported:

- *GenericWithArgs* – generic command that takes additional arguments as input (appended to command)
- *KillContainers* – kills the specified containers (`docker container kill`)
- *PauseContainers* – pauses the specified containers (`docker container pause`)
- *RemoveContainers* – for removing conatiners (`docker container rm`)
- *RemoveImages* – for removing images (`docker image rm`)
- *RunWithArgs* – like *Run*, but appends the incoming arguments to the command
- *StartContainers* – for starting containers (`docker container start`)
- *StopContainers* – stops the specified containers (`docker container stop`)

Some of the commands can be run in either *blocking* or *asynchronous* mode. The former waits till the command finishes and then forwards the collected output from stdout (recommended for quick commands). The latter forwards the output being generated by the command as it happens (recommended for long-running commands like pulling or building).
The following plugins can be used for handling output received from stderr while running a docker command:

- *CallableActorSink* – sends the output to the specified callable sink.
- *Enqueue* – adds the output to the specified queue in internal storage.
- *Log* – uses its logger instance to print the output.
- *Null* – ignores any output.

NB: When needing to start multiple Docker containers in a single flow, but still monitoring the output that they are generating (i.e., in blocking fashion), then the *LaunchTee* or *LaunchTrigger* control actor can be used to wrap the Docker command.

Bibliography

[1] *ADAMS* – Advanced Data mining and Machine learning System
`https://adams.cms.waikato.ac.nz/`

[2] *Docker* – offers OS-level virtualization of software using containers
`https://www.docker.com/`