# DockMaker Command Line Tool

Version 9.3 - By Tim Kafalas, for Hitachi Vantara LLC 1/4/2022 rev. 3/23/2022

## Overview

The DockMaker command line tool is intended to create docker images and start docker containers that will support various Pentaho products, including Pentaho Server, Carte, pdi (kitchen/pan), and Spoon. The command line tool’s job is as follows:

1. Attempt to download all artifacts associated with the requested container into an “Artifact Cache”. Alternatively, the “Artifact Cache” folder can be pre-populated with artifacts by loading them into the artifact cache folder. The artifact downloads can be configured in the DockMaker.properties file explained later.
2. Create a “generatedFiles” folders where the docker files and all other dependent files will be located. The files generated here will contain all the necessary files to run a docker build and docker compose command automatically, or manually.
3. Create a fully functional Docker image of the Pentaho product using a “docker build” command generated.
4. Create all containers by running “docker compose up” on the generated docker-compose.yml file to bring up the product in a running container.

## Requirements

* An installed/stable docker instance. The instance should have docker-compose installed. If running on windows, you should have WSL2 installed and active.
* A user account with <https://hub.docker.com/> . The login should be configured using the “docker login” command so that docker can pull registered database containers.
* The curl command should be installed in the host operating system.
* If downloading oracle databases, you have to have a login to Oracle’s container repository.
* Pentaho licenses installed on the host. Environment variable PENTAHO\_INSTALLED\_LICENSE\_PATH should point to the installed. (eg: PENTAHO\_INSTALLED\_LICENSE\_PATH=C:\Pentaho\.installedLicenses.xml )
* Java 8 or 11 is installed on the host machine.

## Installation

Obtain a copy of the dock-maker-1.0-bin.zip file. Unzip this file to the folder of your choice. You will see a folder structure like the one below:

## Graphical user interface, text, application, email Description automatically generated

|  |  |
| --- | --- |
| Folder/File Name | Purpose |
| artifactCache | This folder serves as the default storage location for any artifacts that are downloaded or required to setup the image. The location of this folder can be changed in the DockMaker.properties file. |
| containers | This has various files and templates that will be tapped when running the command line tool. |
| generatedFiles | This folder is created when the command line tool is executed. It contains all the file necessary to create a docker image and use docker compose to bring up the containers. |
| lib | Dependent libraries. |
| DockMaker.bat or DockMaker.sh | The command line for Windows and Linux, respectively. |
| DockMakerDown.bat or DockMakerDown.sh | A command to bring down, and remove, the containers and volumes used by the docker compose command. |

## DockMaker Command Line Syntax

|  |  |  |
| --- | --- | --- |
| Command Parameter | Required | Description |
| -V,--pentaho-version <arg> | Yes | The version of the pentaho server to download as a base installation followed by a "/", followed by the distribution build number, followed by another “/”, followed by "ce" or "ee". For example "9.1.0.0/324/ee" installs pentaho server-ee version 9.1.0.0 build 324. |
| -T,--product-type | No | The type of product this image represents. Valid values are:   |  |  | | --- | --- | | Value | Description | | server | The docker image will contain a fully functional pentaho-server. It is the default value if omitted. | | pdi | The docker image will contain a pdi kernel sufficient to run pan and kitchen commands. | | carte | The docker image will contain a fully functional carte server. | | spoon | The docker image will contain a runnable spoon instance (See more below) | |
| -A,--additional-plugins <arg> | No | Contains acronyms for the plugin products to be installed with the server. Known plugin types are "std", "paz", "pdd" and "pir". Enter all plugins to install, separated with commas, or leave blank to install no additional plugins. "std" is a special entry that will add all the plugins without the need to specify them individually. |
| -D,--database <arg> | No | Sets the underlaying database for the repository. Supported values are: "postgres/9.6", “postgres/13.5”, "mysql/5.7", "mysql/8.0", "oracle/latest/ent" and "oracle/latest/ex". Defaults to "postgres/9.6" if omitted. |
| -I,--install-path <arg> | No | Optional path from root to install pentaho server in the image. If omitted, defaults to "/opt/pentaho" |
| -K,--kar-ids <arg> | No | Sets the kar files to include in the PentahoServer image. Supported values are: "cdh61", "cdpdc71", "dataproc1421", "emr521", "hdi40", "hdp30" and "mapr61". Multiple kar files can be loaded by providing a comma separated list. No kar files are loaded if omitted. |
| -M,--metastore | No | Contains the path to a local folder whose contents will be mounted under the /home/pentaho folder on the container. It is intended to hold the pentaho metastore folders, but can also be used to sync other folders, as needed. This folder will be kept in sync with the one on the container, so any metastore changes made in the container will persist back to the local filesystem. The folder given should contain a populated “.pentaho” and “.kettle” folder. This folder will be mounted on /home/pentaho in the container. |
| -N,--no-cache | No | Set this flag to force docker build to use the --no-catch option. |
| -P,--patch-version <arg> | No | The version of the pentaho server patch to download as a base installation followed by a "/" followed by the distribution build number, followed by another “/”, followed by “ce" or "ee". For example “9.1.0.8/627/ee" patches to pentaho server-ee version 9.1.0.8 build 627. |
| -p,--port <arg> | No | Sets the tomcat port number to use for server communication. If omitted, it uses port 8081 for the server, and 8082 for carte. |
| -U,--use-existing-downloads | No | Set this flag to re-use any existing downloaded artifacts. If omitted, artifacts will always be downloaded. Artifacts are kept in the folder defined by the “docker.server.artifactCache” property in the DockMaker.properties file. If Downloading fails, the user can manually put the artifacts needed in this folder and set -U to use them. |
| --user <arg> | No | Only used in carte configurations. Set’s the username associated with the carte login. |

## Sample command examples

Create server for version 9.2.0.0 build 290, ee edition. Include the paz, pdd, and pir plugins. Don’t download files if they already exist in the cache, and lastly accept the license agreement. The docker commands will be displayed but not executed.

DockMaker -V 9.2.0.0/290/ee -A paz,pdd,pir -U --EULA\_ACCEPT=true

Like above but execute the docker commands to bring up the container.

DockMaker -V 9.2.0.0/290/ee -A paz,pdd,pir -U --EULA\_ACCEPT=true -X

Create server version 9.2.0.1 build 364 ee edition by applying the 9.2.0.1 patch to a 9.2.0.0 image.

DockMaker -V 9.2.0.0/290/ee -A std -U -P 9.2.0.1/364/ee --EULA\_ACCEPT=true

Create server using current snapshot of build.

DockMaker -V 9.3.0.0-SNAPSHOT/latest/ee -A std -U --EULA\_ACCEPT=true

Create server using mysql 5.7 database.

DockMaker -V 9.2.0.1/364/ee -A std -U -D mysql/5.7 --EULA\_ACCEPT=true

Create server using latest oracle enterprise database (login to Oracle repo needed for download)

DockMaker -V 9.2.0.1/364/ee -A std -U -D oracle/latest/ent --EULA\_ACCEPT=true

Create server including the cdh61 and hdp30 kar files.

DockMaker -V 9.2.0.1/364/ee -A std -U -K cdh61,hdp30 --EULA\_ACCEPT=true

Create pdi (kitchen/pan).

DockMaker -T pdi -V 9.2.0.1/364/ee -U --EULA\_ACCEPT=true

Create carte server

DockMaker -T carte -V 9.2.0.1/364/ee -U --user cluster --password cluster --EULA\_ACCEPT=true

Create Spoon using d:/metastore as the metastore folder and bring up the container.

DockMaker -T spoon -V 9.2.0.1/364/ee -U --EULA\_ACCEPT=true -M d:/metastore -X

## Creating/Destroying a Pentaho Server Container Set

Two containers are used to run Pentaho Server in docker; one container is for the server itself, and the other is for the database. After issuing a command to build a server, the “generatedFiles” folder will contain a “dockerfile” and a “docker-compose.yml” file. The dockerfile will be a standard dockerfile from which a Pentaho Server image can be obtained. The docker-compose.yml file is a standard docker compose file and brings up both the database and PentahoServer containers from their associated images.

When run with the -X parameter to bring up the containers, the images look like this:

D:\tmp\cli\dock-maker-1.0-SNAPSHOT>docker image ls

REPOSITORY TAG IMAGE ID CREATED SIZE

pentaho/pentaho-server 9.2.0.0 a1db3120f694 5 minutes ago 3.22GB

postgres 9.6 9d61bc309f2f 2 months ago 384MB

Note that if you are on a Linux machine, you might need to prefix the command with sudo. The containers look like this:

D:\tmp\cli\dock-maker-1.0-SNAPSHOT>docker container ls

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

aad32eac73cf postgres:9.6 "docker-entrypoint.s…" 4 minutes ago Up 4 minutes 5432/tcp generatedfiles-repository-1

c1fc31af8f8f pentaho/pentaho-server:9.2.0.0 "/docker-entrypoint.…" 4 minutes ago Up 4 minutes 0.0.0.0:8081->8080/tcp generatedfiles-pentahoServer-1

The database data is stored on a docker volume and looks like this:

D:\tmp\cli\dock-maker-1.0-SNAPSHOT>docker volume ls

DRIVER VOLUME NAME

local generatedfiles\_repository-data

You can now log into the server by pasting <http://localhost:8081/pentaho/Login> into your browser. Note that 8081 is the default port number used. It can be changed with the -p parameter.

To stop the containers without deleting them or the database data, enter this command:

docker compose -f generatedFiles/docker-compose.yml stop

To restart the containers, enter this:

docker compose -f generatedFiles/docker-compose.yml start

To stop and delete the containers but leave the database volume intact enter this:

docker compose -f generatedFiles/docker-compose.yml down

To stop and delete both the containers and all associated volumes enter this:

docker compose -f generatedFiles/docker-compose.yml down -v

Alternatively, we provide a script file, DockMakerDown.bat or .sh that will issue this command for you.

## Creating a PDI container

The pdi container is used to run kitchen and pan commands. To create a pdi container, use the “-T pdi” parameter in the command line. When executed with -X, the container will be built and a test.ktr transformation will be run on the container. After completion the container will be stopped.

D:\tmp\cli\dock-maker-1.0-SNAPSHOT>docker container ls -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

b2bc29644312 pentaho/pdi:9.2.0.1 "/docker-entrypoint.…" 49 seconds ago Exited (0) 8 seconds ago generatedfiles-pdi-1

Docker keeps a log of any container that has not been deleted. To see a log of the container, use the following command:

docker logs {container id}

where {container id} is the id reported by the “docker container ls -a” command.

To run a command yourself, you use the “docker run” command on the stopped container like so:

docker-compose -f generatedFiles/docker-compose.yml run pdi ./pan.sh /file:/opt/pentaho/data-integration/simpleTrans.ktr

In this example we will manually run our test.ktr but any ktr can be used. Kitchen can be used to run jobs in the same manor.

## Creating a Carte Container

The carte container offers a carte server. To create a pdi container, use the “-T carte” parameter in the command line. It uses the same image as the pdi container, but it starts this container in carte mode. The default login user, password and port can be used for the carte server, or it can be specified on the command line. If default values are used, the user will be “carte”, the password will be “carte”, and the port number will be 8082.

Using the -X param to bring up the carte container will result in a container that will remain running until it is stopped.

To bring down the Carte Server and delete the container, enter:

docker compose -f generatedFiles/docker-compose.yml down

or just run:

DockMakerDown.bat/sh

## Creating a Spoon Container

The spoon container will run a fully functional GUI out of the container. It starts with an Ubuntu 18 Linux image to support the GUI interface, and because of this, it creates a separate spoon image. There are special requirements for running spoon in windows, (see the following section). If we just start the spoon container, we will have a fully functional spoon but with no other files or even metadata such as databases or big data clusters. Everything you want to run must be built from scratch, since the container has no access to the host’s files. We get around this by sharing a folder with the host that is intended to hold the Pentaho Metadata, and anything else the user needs to pass to the container. See the “Shared Volume” section for how this is done.

Using the -X parameter will start a Spoon session in the container’s display. To stop the container simply close the spoon window, or from another terminal, enter:

docker compose -f generatedFiles/docker-compose.yml stop

To start up spoon again, enter:

docker compose -f generatedFiles/docker-compose.yml start spoon

To delete the container all together, enter:

docker compose -f generatedFiles/docker-compose.yml down

or just run:

DockMakerDown.bat/sh

## Running Spoon in a container on Windows.

To run any GUI application in a container on windows 10 or better, it must support one of the GUI protocols in Linux. For best results with Spoon, we recommend installing VcXsrv Windows X Server which is a free tool that provides an X server on Windows which can be download at <https://sourceforge.net/projects/vcxsrv/> .

Install the application and start XLaunch. Hit next until you get to “Extra Settings”. Check the “Disable access control” checkbox, at least for the first try. Once you have Xlaunch running, you can bring up the Spoon docker image. If there is no display to communicate with (due to Xlaunch not present or running in a non-compatible mode), you will get an exception similar to:

generatedfiles-spoon-1 | org.eclipse.swt.SWTError: No more handles [gtk\_init\_check() failed]

generatedfiles-spoon-1 | at org.eclipse.swt.SWT.error(SWT.java:4621)

## Shared Volumes

Containers, by definition, are isolated from the host machine. But suppose you have files on your local file system that needs to be accessible on the container? Docker has the concept of Volumes to address just this type of problem. The DockMaker command tool supports three different volumes to be used with our containers. We will explain the three volumes and how to configure them in the following sub-sections.

### The Override Files Volume

The Override Volume is generated by the DockMaker tool. It contains all the changes that must be made to the basic artifact files to create the configuration desired. This volume binds the generatedFiles\fileOverride folder on the host with the /docker-entrypoint-init folder on the container. When the container is started, the files present in this folder will overwrite the files/folders in the /opt/pentaho/data-integration or /opt/pentaho/pentaho-server folder, whichever one is in play.

If you need any additional files, drivers, etc to be placed in the application’s folder just add it with the proper path, in the fileOverride folder.

Be aware, however, any changes made here will be lost if the command tool is ever executed again. That is why the command line normally gives you the docker commands without executing them. In many cases the user will need to make more changes to the templates provided. If you want to make sure you do not lose the files that were generated, (or manually changed), rename the generatedFiles folder to something else, or copy the folder entirely to another place. It will still work though some paths may have to be adjusted in the commands listed in this document.

### The Metastore Volume

This volume is provided by the user of tool by including the -M or –metastore parameter. It is intended to allow the user to create a bi-directional bind on a metastore folder. It binds the folder defined in the -M parameter with the /home/pentaho folder on the container. You probably don’t want to link it directly to your own metastore because that breaks the container contract of not changing host environment, but you can make a copy of your metastore and bind to that. To do this, perform the following steps:

1. Create an arbitrary folder on your host file system to serve as the shared folder. We’ll use d:\metastore which demonstrates Windows drive usage as well.
2. Copy the “.kettle” and “.pentaho” folders from your home folder to the “d:/metastore” folder, or whatever names you used. This is enough to share the metastore.
3. Copy any other files you need to process your use case. The d:/metastore folder will be bound to the /home/pentaho folder on the container.
4. When you generate the files with DockMaker command, it should contain “-M d:/metastore”.

### The Database Volume

The database volume is only created when a Pentaho Server container is run. This volume contains all the database tables associated with the server repository, quartz scheduler, and log tables. When the database container is started for the first time, it will run DDL provided in generatedFiles that will define and possibly populate the tables needed. The container’s form is essentially dictated by the database provider and is included here for completeness.

## Getting a command prompt on a container

If you need to get a Linux command prompt on a running container like Pentaho Server or Carte, you can issue the following command:

docker exec -it {containerId} bash

…where {containerId} is the id listed with the “docker container ls” command.

To get into a stopped PDI container after a command has been executed:

docker compose run pdi bash

The docker compose yaml file will define pdi as the reference to the container.

## Working with Big Data

To work with big data you will probably want to include a metastore volume, (See Shared Volumes section), that contains all your cluster definitions, and possibly a kar file, (added with the -K parameter). If you are communicating with an unsecure cluster you should be able to bring up a spoon, pdi, or carte container and it should work out of the box.

There are many ways to configure for connections to a secure server. We will illustrate one example here, using Kerberos. Adding a secure cluster connection will require additional changes to the “generatedFiles” folder. We will be changing the dockerfile to bring in the additional dependencies and install SSL keys. In this case our cluster is defined to use user/password rather than keytab, but to use a keytab just requires an additional ADD command to add the keytab file.

1. Run DockMaker without using -X so it doesn’t run the docker commands.
2. Copy the NOT EXECUTED commands from the output and put them somewhere for later use.
3. Copy your krb5.conf file and cacerts.pem file to the “generatedFiles” folder. Any files we copy to the container must be in the generatedFiles context to be available. This is a restriction imposed by docker.
4. Edit the generatedFiles/dockerfile file. Add the following lines close to the bottom of the file but make sure they appear above the “USER ${PENTAHO\_USER}” line as you must be root to execute them.

RUN apt-get install -y krb5-user  
ADD krb5.conf /etc/krb5.conf  
ADD cacerts.pem /tmp/cacerts.pem  
RUN /usr/bin/keytool -import -noprompt -alias CDP71Secure -keystore /etc/ssl/certs/java/cacerts -file /tmp/cacerts.pem -storepass changeit;

1. Run the docker build command you copied in step 2
2. Run the docker compose command you copied in step 2
3. You should now have a running instance with Kerberos support.

You may not need the apt-get command, depending on what image you are working with. Once you get everything working, you might want to make it so that these lines are always added to the Dockerfile when generatedFiles is first created. To do this, you can locate the Dockerfile template being used and make the changes there. The template Dockerfiles’s can be found at:

|  |  |
| --- | --- |
| Product being built | Template File |
| Server | containers\pentaho-server\pentaho-server-auto\Dockerfile |
| Pdi | containers\pentaho-data-integration\pdi-client-auto\Dockerfile |
| Carte | containers\pentaho-data-integration\pdi-client-auto\Dockerfile |
| Spoon | containers\pentaho-data-integration\pdi-client-auto\Dockerfile-spoon |

## DockMaker.properties File

This file contains configuration settings for the DockMaker command line tool, particularly the download and storage of artifacts. The artifact cache folder is defined with:

docker.server.artifactCache=./artifactCache

The default is to use the “artifactCache” folder in command line directory but feel free to set this to someplace more secure on your filesystem.

The rest of the file contains information on how to build a url for the artifact one wants to download. For instance, the docker.download.host.prefix property defines the host URI that will be used for all pentaho artifact downloads. The docker.server.curlCommand property defines the syntax for the curl command, etc. In some installations you may want to change this to a wget command, for instance.

### Preset Variables

A set of variables are automatically available and can be used within the properties file to create other variables. The preset variables include:

|  |  |
| --- | --- |
| Variable Name | Description |
| version | The version of the base server zip (ie: 9.2.0.0) |
| distNumber | The build number of base server zip (ie: 657) |
| patchVersion | The version of the patch file (ie: 9.2.0.3), if supplied. |
| patchDistNumber | The build number of the patch file (ie: 828) |
| edition | Contains CE or EE |
| pluginId | 3 character plugin ID (ie: paz, pdd, or pir) being processed. |
| karId | The unique ID for a kar. (ie: The kar file used on the command line such as cdh61) |
| PORT | The port number from the command line |
| USER | The user from the command line |
| PASSWORD | The password from the command line |

### Derived Variables

Derived variables are variables whose values are based on a true/false condition. All properties names for these derived variables start with “docker.server.var” followed by a number to make each line unique, followed by the name of variable being set, followed by the value to set it too. The value of this property contains a javascript representation of the condition for setting this variable. All derived variables are usually supplied as the last section in the file but can appear anywhere. Let’s examine some derived properties to see how they work. Consider the lines below:

# set fileDistNumber to the distNumber, or blank if the distNumber="latest"

docker.server.var.0.fileDistNumber.="${distNumber}" === "latest"

docker.server.var.1.fileDistNumber.-${distNumber}="${distNumber}" != "latest"

The first line is simply a comment describing the lines that follow. Line 2 will set a derived variable named “fileDistNumber” to an empty string, (since there is nothing to the right of the final period), but only if the distNumber equals “latest”. Line 3 will set the same “fileDistNumber” variable to ${distNumber}, but only if the distNumber does not equal “lastest”.

### Mandatory Variables

All other properties in the file are simply variables with a value. Some of these are required by the application to properly download artifacts. They are:

|  |  |
| --- | --- |
| Mandatory Variable Name | Description |
| docker.pdi.pdiArtifactFileName | Defines the file name associated with the PDI artifact. |
| docker.pdi.pdiArtifactUrl | Defines the complete URL used in the curl command to download the PDI artifact. |
| docker.server.serverArtifactFileName | Defines the file name associated with the Pentaho Server artifact |
| docker.server.serverArtifactUrl | Defines the complete URL used in the curl command to download the Pentaho Server artifact |
| docker.server.serverPluginFileName | Defines the file name associated with a ${pluginId} (paz, pad, pir) |
| docker.server.serverPluginUrl | Defines the complete URL used in the curl command to download a ${pluginId} |
| docker.server.serverPatchFileName | Defines the file name associated with the patch artifact. |
| docker.server.serverPatchUrl | Defines the complete URL used in the curl command to download the patch. |
| docker.server.karFileName | Defines the file name associated with the a ${karId} |
| docker.server.karUrl | Defines the complete URL used in the curl command to download a kar file. |
| docker.server.curlCommand | Defines the syntax of the curl command where ${url} contains one of the URLs listed above, ${outputFolder} contains the destination folder for the artifact, and ${filename} contains the file name associated with the url. |

### Variable Resolution Order

All variables in the file are resolved in a specific order as follows:

1. Resolve all Preset Variables based on command line entered.
2. Resolve all Derived Variables in the order they appear in the DockMaker.properties file. Derived Variables therefore only have access to Preset Variables and any Derived Variables that appear earlier in the file.
3. Resolve all Mandatory variables and their dependencies just before applying that variable.

Other variables can be added as necessary to simplify the definition process. For instance, the “docker.download.host.prefix” was added to facilitate the hostname in the urls.

## The registry.yml file

The containers/registry.yml file contains information about what versions and combinations are supported by the command tool. The “databaseMap” section defines what databases can be used to host the Pentaho Server repository database. The section for postgres is show below:

**postgres**:  
 **databaseInstances**:  
 - **versions**: **"9.6,13.5"  
 edition**:  
 **composeYml**: docker-compose-postgres.yml  
 **dbInitFolder**: db\_init\_postgres  
 **image**: postgres:${DATABASE\_VERSION}

Two versions of postgres are allowed, 9.6 or 13.5. The fully qualified database names available on the command line will be “postgres/9.6” and “postgres/13.5”. The compose yml file that will serve as a template to this configuration is “dock-compose-postgres.yml” and the folder containing the DDL to define the tables can be found in the “db\_init\_postgres” folder. The docker image that will be pulled down is defined in “image” property and will be either “postgres:9.6” or “postgres:13.5”.

The “karFileRegistry” section defines the various kar files that can be installed.

The “buildRegistry” section defines various fields that are conditional to the Pentaho version being chosen. Pentaho Server Version 8.X support openjdk:8, but Pentaho Server Version 9.X supports openjdk:8 or openjdk:11.

.