fabric8io/docker-maven-plugin

Roland Huß

Version 0.43.1, 2023-07-28

docker-maven-plugin

Chapter 1. Introduction

This is a Maven plugin for managing Docker images and containers. It focuses on two major aspects for a Docker build integration:

1.1. Building Images

One purpose of this plugin is to create Docker images holding the actual application. This is done with the **docker:build** goal. It is easy to include build artefacts and their dependencies into an image.

Several ways for configuring the builds are supported:

- An own configuration syntax can be used to create a Dockerfile. For specifying artefacts and other files, the plugin uses the assembly descriptor format from the maven-assembly-plugin to copy over those file into the Docker image.

- An external Dockerfile can be specified in which Maven properties can be inserted. This is also the default mode, if only a single image should be built and a top-level `Dockerfile` exists. See Simple Dockerfile build for details of this zero XML configuration mode.

Images that are built with this plugin can be pushed to public or private Docker registries with **docker:push**.

1.2. Running Containers

With this plugin it is possible to run completely isolated integration tests so you don't need to take care of shared resources. Ports can be mapped dynamically and made available as Maven properties to your integration test code.

Multiple containers can be managed at once, which can be linked together or share data via volumes. Containers are created and started with the **docker:start** goal and stopped and destroyed with the **docker:stop** goal. For integration tests both goals are typically bound to the the pre-integration-test and post-integration-test phase, respectively. It is recommended to use the maven-failsafe-plugin for integration testing in order to stop the docker container even when the tests fail.

For proper isolation, container exposed ports can be dynamically and flexibly mapped to local host ports. It is easy to specify a Maven property which will be filled in with a dynamically assigned port after a container has been started. This can then be used as parameter for integration tests to connect to the application.

1.3. Configuration

The plugin configuration contains a global part and a list of image-specific configuration within a `<images>` list, where each image is defined within a `<image>` tag. See below for an example.

The global part contains configuration applicable to all images like the Docker URL or the path to the SSL certificates for communication with the Docker Host.

Then, each specific image configuration has three parts:

- A general image part containing the image name and alias.
- A `<build>` configuration specifying how images are built
- A `<run>` configuration describing how containers should be created and started.
- A `<copy>` configuration describing how files and directories from containers should be copied to the host.

The `<build>`, `<run>` and `<copy>` parts are optional and can be omitted.

1.4. Example

In the following examples, two images are specified. One is the official PostgreSQL 9 image from Docker Hub, which internally is referenced with an alias "database". It only has a `<run>` section which declares that the startup should wait until the given text pattern is matched in the log output. Next is a "service" image, which has a `<build>` section. It creates an image which has artifacts and dependencies in the `/maven` directory (and which are specified with an assembly descriptor). Additionally, it specifies the startup command for the container, which in this example fires up a microservice from a jar file copied over via the assembly descriptor. It also exposes port 8080. In the `<run>` section this port is mapped to a dynamically chosen port and then assigned to the Maven property `${tomcat.port}`. This property could be used, for example, by an integration test to access this microservice. An important part is the `<links>` section which indicates that the image with the alias of "database" is linked into the "service" container, which can access the internal ports in the usual Docker way (via environment variables prefixed with `DB_`).

Images can be specified in any order and the plugin will take care of the proper startup order (and will bail out if it detects circular dependencies).

Example plugin configuration

```xml
<configuration>
  <images>
    <image>
      <alias>service</alias> ①
      <name>fabric8/docker-demo:${project.version}</name>

      <build> ②
        <from>java:8</from> ③
        <assembly>
          <descriptor>docker-assembly.xml</descriptor> ④
        </assembly>
        <cmd> ⑤
          <shell>java -jar /maven/service.jar</shell>
        </cmd>
      </build>

      <run> ⑥
        <ports> ⑦
```

```xml
            <port>tomcat.port:8080</port>
          </ports>
          <wait>  ⑧
            <http>
              <url>http://localhost:${tomcat.port}/access</url>
            </http>
            <time>10000</time>
          </wait>
          <links>  ⑨
            <link>database:db</link>
          </links>
        </run>

        <copy>  ⑫
          <entries>
            <entry>
              <containerPath>/etc/hosts</containerPath>  ⑬
              <hostDirectory>${project.build.directory}</hostDirectory>  ⑭
            </entry>
          </entries>
        </copy>
      </image>

      <image>
        <alias>database</alias>  ⑩
        <name>postgres:9</name>
        <run>
          <wait>  ⑪
            <log>database system is ready to accept connections</log>
            <time>20000</time>
          </wait>
        </run>
      </image>
    </images>
  </configuration>
```

① Image configuration for a Java service with alias "service" and name `fabric8/docker-demo:${project.version}`

② build configuration defines how a Docker image should be created

③ Base image, in this case `java:8`

④ Content of the image can be specified with an assembly descriptor

⑤ Default command to run when a container is created.

⑥ Run configuration defines how a container should be created from this image

⑦ Port mapping defines how container ports should be mapped to host ports

⑧ Wait section which is a readiness check when starting the service

⑨ Network link describes how this service's container is linked to the database container

⑩ Second image is a plain database image which is only needed for running (hence there is no

`<build>` section). The alias is used in the network link section above

⑪ Wait until the corresponding output appears on stdout when starting the Docker container.

⑫ Copy configuration defines what files and directories of a container should be copied to the host by **docker:copy** goal

⑬ Defines what file of a container should be copied to the host

⑭ Defines target directory of the host to place the file copied from a container

1.5. Features

Some other highlights, in random order:

- Auto pulling of images with a progress indicator

- Waiting for a container to startup based on time, the reachability of an URL, or a pattern in the log output

- Support for SSL Authentication and OpenShift credentials

- Docker machine support

- Flexible registry handling (i.e. registries can be specified as metadata)

- Specification of encrypted registry passwords for push and pull in ~/.m2/settings.xml (i.e., outside the pom.xml)

- Color output

- Watching on project changes and automatic recreation of image

- Properties as alternative to the XML configuration

- Support for Docker daemons accepting http or https request via TCP and for Unix sockets

Chapter 2. Installation

This plugin is available from Maven central and can be connected to pre- and post-integration phase as seen below. The configuration and available goals are described below.

Example

```xml
<plugin>
  <groupId>io.fabric8</groupId>
  <artifactId>docker-maven-plugin</artifactId>
  <version>0.43.1</version>

  <configuration>
     ....
     <images>
        <!-- A single's image configuration -->
        <image>
           ....
        </image>
        ....
     </images>
  </configuration>

  <!-- Connect start/stop to pre- and
       post-integration-test phase, respectively if you want to start
       your docker containers during integration tests -->
  <executions>
    <execution>
       <id>start</id>
       <phase>pre-integration-test</phase>
       <goals>
         <!-- "build" should be used to create the images with the
              artifact -->
         <goal>build</goal>
         <goal>start</goal>
       </goals>
    </execution>
    <execution>
       <id>stop</id>
       <phase>post-integration-test</phase>
       <goals>
          <goal>stop</goal>
       </goals>
    </execution>
  </executions>
</plugin>
```

When working with this plugin you can use an own packaging with a specialized lifecycle in order to keep your pom files small. Three packaging variants are available:

- **docker** : This binds `docker:build` to the package phase and `docker:start` / `docker:stop` to the pre- and post-integration phase respectively. Also `docker:push` is bound to the deploy phase.

- **docker-build** : Much like the *docker* packaging, except that there are no integration tests configured by default.

- **docker-tar** : Create a so called *Docker tar* archive which is used as the artifact and which later can be used for building an image. It contains essentially a `Dockerfile` with supporting files. See [docker:source](#) for more details.

These packaging definitions include the *jar* lifecycle methods so they are well suited for simple Microservice style projects.

Example

```
<pom>
  <artifactId>demo</artifactId>
  <version>0.0.1</version>
  <packaging>docker</packaging>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>io.fabric8</groupId>
        <artifactId>docker-maven-plugin</artifactId>
        <extensions>true</extensions>
        <configuration>
          <images>
            <image>
            ...
            </image>
          </images>
        </configuration>
      </plugin>
    </plugins>
    ....
  </build>
</pom>
```

This will create the jar (if any), build the Docker images, start the configured Docker containers, runs the integration tests, stops the configured Docker container when you enter `mvn install`. With `mvn deploy` you can additionally push the images to a Docker configuration. Please note the `<extensions>true</extensions>` which is mandatory when you use a custom lifecycle.

The rest of this manual is now about how to configure the plugin for your images.

Chapter 3. Global configuration

Global configuration parameters specify overall behavior like the connection to the Docker host. The corresponding system properties which can be used to set it from the outside are given in parentheses.

The docker-maven-plugin uses the Docker remote API so the URL of your Docker Daemon must somehow be specified. The URL can be specified by the dockerHost or machine configuration, or by the `DOCKER_HOST` environment variable.

The Docker remote API supports communication via SSL and authentication with certificates. The path to the certificates can be specified by the certPath or machine configuration, or by the `DOCKER_CERT_PATH` environment variable.

Table 1. Global Configuration

Element	Description	Property
apiVersion	Use this variable if you are using an older version of docker not compatible with the current default use to communicate with the server.	`docker.apiVersion`
authConfig	Authentication information when pulling from or pushing to Docker registry. There is a dedicated section [Authentication](#) for how doing security.	
autoCreateCustomNetworks	Create automatically Docker networks during `docker:start` and remove it during `docker:stop` if you provide a custom network in the run configuration of an image. The default is `false`.	`docker.autoCreateCustomNetworks`
autoPull	Decide how to pull missing base images or images to start. This option is **deprecated**, please use [imagePullPolicy](#) instead. The following values are supported: • `on` or `once`: Automatic download any missing images (default) • `off` : Automatic pulling is switched off always • `always` : Pull images always even when they already exist locally.	`docker.autoPull`
buildArchiveOnly	Skip the actual Docker image build and only create the archive holding the Dockerfile and build context. The following values are supported: • `/path/to/archive` : Create the build tar archive as file with name `/path/to/archive` and then stop without doing the actual image build • `true` (or an empty value) : Skip building the image, but don't copy the generated build archive. • `false` : Build the image. This is the default behaviour.	`docker.buildArchiveOnly`

Element	Description	Property
certPath	Path to SSL certificate when SSL is used for communicating with the Docker daemon. These certificates are normally stored in `~/.docker/`. With this configuration the path can be set explicitly. If not set, the fallback is first taken from the environment variable `DOCKER_CERT_PATH` and then as last resort `~/.docker/`. The keys in this are expected with it standard names `ca.pem`, `cert.pem` and `key.pem`. Please refer to the Docker documentation for more information about SSL security with Docker.	`docker.certPath`
dockerHost	The URL of the Docker Daemon. If this configuration option is not given, then the optional `<machine>` configuration section is consulted. The scheme of the URL can be either given directly as `http` or `https` depending on whether plain HTTP communication is enabled or SSL should be used. Alternatively the scheme could be `tcp` in which case the protocol is determined via the IANA assigned port: 2375 for `http` and 2376 for `https`. Finally, Unix sockets are supported by using the scheme `unix` together with the filesystem path to the unix socket. The discovery sequence used by the docker-maven-plugin to determine the URL is: 1. value of **dockerHost** (`docker.host`) 2. the Docker host associated with the docker-machine named in `<machine>`, i.e. the `DOCKER_HOST` from `docker-machine env`. See below for more information about Docker machine support. If `<machine>` is not set, then no docker-machine detection is used. 3. the value of the environment variable `DOCKER_HOST`. 4. `/var/run/docker.sock` if it is a readable socket (Unix & OS X). 5. `//./pipe/docker_engine` if it is a readable named pipe (Windows)	`docker.host`
filter	In order to temporarily restrict the operation of plugin goals this configuration option can be used. Typically this will be set via the system property `docker.filter` when Maven is called. The value can be a single image name (either its alias or full name) or it can be a comma separated list with multiple image names. Any name which doesn't refer an image in the configuration will be ignored.	`docker.filter`

Element	Description	Property
imagePullPolicy	Specify whether images should be pull when looking for base images while building or images for starting. This property can take the following values (case insensitive): • `IfNotPresent`: Automatic download any missing images (default) • `Never` : Automatic pulling is switched off always • `Always` : Pull images always even when they already exist locally. By default a progress meter is printed out on the console, which is omitted when using Maven in batch mode (option `-B`). A very simplified progress meter is provided when using no color output (i.e. with `-Ddocker.useColor=false`).	`docker.imagePullPolicy`
logDate	Date format which is used for printing out container logs. This configuration can be overwritten by individual run configurations and described below. The format is described in Logging.	`docker.logDate`
logStdout	For all container logging to standard output if set to `true`, regardless whether a `file` for log output is specified. See also Logging	`docker.logStdout`
machine	Docker machine configuration. See Docker Machine for possible values	
maxConnections	Number of parallel connections are allowed to be opened to the Docker Host. For parsing log output, a connection needs to be kept open (as well for the wait features), so don't put that number to low. Default is 100 which should be suitable for most of the cases.	`docker.maxConnections`
jib	Delegate Image Build process to JIB, `false` by default. Note that this option is applicable only for build and push goals, other goals won't work if this is enabled (since they dependend on Docker specific features)	`docker.build.jib`
jibImageFormat	Format of the image to be built. Values can be `oci` and `docker` with `docker` as default value	`docker.build.jib.imageFormat`
outputDirectory	Default output directory to be used by this plugin. The default value is `target/docker` and is only used for the goal `docker:build`.	`docker.target.dir`
portPropertyFile	Global property file into which the mapped properties should be written to. The format of this file and its purpose are also described in Port Mapping.	
registry	Specify globally a registry to use for pulling and pushing images. See Registry handling for details.	`docker.registry`
skip	With this parameter the execution of this plugin can be skipped completely.	`docker.skip`
skipBuild	If set no images will be build (which implies also *skip.tag*) with `docker:build`	`docker.skip.build`
skipPush	If set dont push any images even when `docker:push` is called.	`docker.skip.push`

Element	Description	Property
skipPom	If set to `true` this plugin will skip every projects, where `project.packaging` is set to `pom`.	`docker.skip.pom`
skipRun	If set dont create and start any containers with `docker:start` or `docker:run`	`docker.skip.run`
skipTag	If set to `true` this plugin won't add any tags to images that have been built with `docker:build`. If set to `true` this plugin won't push any tags with `docker:push`. If set to `true` this plugin won't remove any tags with `docker:remove`.	`docker.skip.tag`
skipMachine	Skip using docker machine in any case	`docker.skip.machine`
sourceDirectory	Default directory that contains the assembly descriptor(s) used by the plugin. The default value is `src/main/docker`. This option is only relevant for the `docker:build` goal.	`docker.source.dir`
useColor	Whether to use colored log output. By default this is switched on when running on a console, off otherwise.	`docker.useColor`
outputFile	If specified, this parameter will cause the logs to be written to the path specified, instead of writing to the console.	`outputFile`
verbose	String attribute for switching on verbose output on standard output (stdout). It takes a comma separated list of string values to switch on various verbosity groups. The currently known groups are: build:: Print out Docker build instructions api:: API calls to the Docker daemons are logged all:: All levels are enabled If you set an empty string (or only e.g. `-Ddocker.verbose`) then the "build" group is enabled. You can also use "true" / "false" to switch on / off verbose logging. Default is that verbose logging is disabled.	`docker.verbose`

Example

```
<configuration>
    <dockerHost>https://localhost:2376</dockerHost>
    <certPath>src/main/dockerCerts</certPath>
    <useColor>true</useColor>
    .....
</configuration>
```

Docker Machine

This plugin supports also Docker machine (which must be installed locally, of course). A Docker machine configuration can be provided with a top-level `<machine>` configuration section. This configuration section knows the following options:

Table 2. Docker Machine Options

Element	Description
name	Docker machine's name. Default is `default`
autoCreate	if set to `true` then a Docker machine will automatically created. Default is `false`.
regenerateCertsAfterStart	if set to `true` then certificates will be regenerated after starting the Docker Machine. This is useful if using the AWS EC2 driver, which will assign machines new IP addresses after each start. Default is `false`.
createOptions	Map with options for Docker machine when auto-creating a machine. See the docker machine documentation for possible options.

When no Docker host is configured or available as an environment variable, then the configured Docker machine is used. If the machine exists but is not running, it is started automatically. If it does not exists but `autoCreate` is true, then the machine is created and started. Otherwise, an error is printed. Please note, that a machine which has been created because of `autoCreate` gets never deleted by docker-maven-plugin. This needs to be done manually if required.

In absence of a `<machine>` configuration section the Maven property `docker.machine.name` can be used to provide the name of a Docker machine. Similarly, the property `docker.machine.autoCreate` can be set to true for creating a Docker machine, too.

You can use the property `docker.skip.machine` if you want to override the internal detection mechanism to always disable docker machine support.

Example

```
<!-- Work with a docker-machine -->
<configuration>
  <machine>
    <name>maven</name>
    <autoCreate>true</autoCreate>
    <createOptions>
      <driver>virtualbox</driver>
      <virtualbox-cpu-count>2</virtualbox-cpu-count>
    </createOptions>
  </machine>
    .....
</configuration>
```

Chapter 4. Image configuration

The plugin's configuration is centered around *images*. These are specified for each image within the `<images>` element of the configuration with one `<image>` element per image to use.

The `<image>` element can contain the following sub elements:

Table 3. Image Configuration

Element	Description
name	Each `<image>` configuration has a mandatory, unique docker repository *name*. This can include registry and tag parts, but also placeholder parameters. See below for a detailed explanation.
alias	Shortcut name for an image which can be used for identifying the image within this configuration. This is used when linking images together or for specifying it with the global **image** configuration element.
registry	Registry to use for this image. If the `name` already contains a registry this takes precedence. See Registry handling for more details.
build	Element which contains all the configuration aspects when doing a **docker:build**. This element can be omitted if the image is only pulled from a registry e.g. as support for integration tests like database images.
run	Element which describe how containers should be created and run when **docker:start** is called. If this image is only used a *data container* (i.e. is supposed only to be mounted as a volume) for exporting artifacts via volumes this section can be missing.
copy	Describes how files and directories of containers should be copied when **docker:copy** is called. This element is optional.
external	Specification of external configuration as an alternative to this XML based configuration with `<run>` and `<build>`. It contains a `<type>` element specifying the handler for getting the configuration. See External configuration for details.
removeName Pattern	When this image is to be removed by **docker:remove**, use this pattern list to find images to remove rather than just using the name.
stopNamePatt ern	When containers associated with this image will be stopped by **docker:stop**, use this pattern list to find containers to remove rather than just using the associated container name.
copyNamePatt ern	When copying files and directories defined in copy element of the image configuration with **docker:copy** goal, use this pattern to find containers to copy from. This element is optional.

Either a `<build>` or `<run>` section must be present (except when you are using the simple Dockerfile build mode). These are explained in details in the corresponding goal sections.

Example

```
<configuration>
  ....
  <images>
    <image>
      <name>%g/docker-demo:0.1</name>
      <alias>service</alias>
      <run>....</run>
      <build>....</build>
    </image>
  </images>
</configuration>
```

When using Maven profiles, it can be useful to override settings of a particular image. To facilitate this, the element `<imagesMap>` can be used alongside the `<images>` element. Each entry in `<imagesMap>` translates to an image configuration where the **alias** of the image is set to the map entry's **key**. The examples above and below produce identical image configurations.

Example

```
<configuration>
  ....
  <imagesMap>
    <service>
      <name>%g/docker-demo:0.1</name>
      <run>....</run>
      <build>....</build>
    </service>
  </images>
</configuration>
```

4.1. Image Names

When specifying the image name in the configuration with the `<name>` field you can use several placeholders which are replaced during runtime by this plugin. In addition you can use regular Maven properties which are resolved by Maven itself.

Replacements can also be used in `<tag>` fields within the the tags of any build configuration.

Placeholder	Description
%g	The last part of the Maven group name, sanitized so that it can be used as username on GitHub. Only the part after the last dot is used. E.g. for a group id `io.fabric8` this placeholder would insert `fabric8`
%a	A sanitized version of the artefact id so that it can be used as part of an Docker image name. I.e. it is converted to all lower case (as required by Docker)
%v	The project version. Synonym to `${project.version}`

Placeholder	Description
%l	If the project version ends with `-SNAPSHOT` then this placeholder is `latest`, otherwise its the full version (same as `%v`)
%t	If the project version ends with `-SNAPSHOT` this placeholder resolves to `snapshot-<timestamp>` where timestamp has the date format `yyMMdd-HHmmss-SSSS` (eg `snapshot-`). This feature is especially useful during development in oder to avoid conflicts when images are to be updated which are still in use. You need to take care yourself of cleaning up old images afterwards, though.
%T	Timestamp with the format `yyMMdd-HHmmss-SSSS`.

4.2. Container Names

Similar to image name placeholders, for starting and stopping containers and alternate set of placeholders can be configured in order to the name the containers to create.

These placeholders can be used in the top-level configuration value `containerNamePattern` which is used globally for every container that is created. This global pattern can be overwritten individually by each image's **run** configuration. If neither is given, then by default the pattern `%n-%i` is used.

When specifying the container name pattern the following placeholders can be used:

Placeholder	Description
%a	The `<alias>` of an image which must be set. The alias is set in the top-level image configuration
%e	Choose an empty container name, which will let the docker engine chose a random container name automatically. This placeholder must be given as single value to `containerNamePattern` when used.
%n	A sanitized version of the image's short name from which this container is created. "Sanitized" means that any non letter, digit, dot or dash is replaced by an underscore.
%t	The build time stamp. This is the timestamp which created during the building of an image and locally cached. A rebuild of the image will update the timestamp.
%i	An index which is incremented if a container has already been created. With this parameter it is easily possible to have multiple, similar containers. See the example below for more details.

You can combine the placeholders in any combination and will be resolved during `docker:start`, `docker:stop` and `docker:watch`.

The following example is using a container name pattern of `%n-%i` which is also the default. Given an image `fabric8io/dmp-sample-jolokia:latest`, then during `mvn docker:start` a container with the name `dmp-sample-jolokia-1` is first tried. If there is already a container with this name, then `dmp-sample-jolokia-2` is the second attempt. This goes on until a "free" name is found.

Similar, when stopping containers with `mvn docker:stop` then only the container with the highest index is stopped. However, if you don't use an index via `%i` then *all* containers started with

`docker:start` are stopped. Use `mvn docker:stop -Ddocker.allContainers` to also stop every container named via a `%i` pattern.

4.3. Name Patterns

Goals that need to refer to images or containers where the name of the image or container is not fixed may support name patterns for matching. Patterns can use an Ant-like syntax or Java regular expressions.

4.3.1. Ant-like Name Patterns

Ant path matching patterns that operate on path names use the convention that a * matches within a single path component, while ** can match multiple components.

Adapting this style to image names requires some tweaks since image names may include registry information, a path-like repository name and a tag. Consider the following image names:

- `alpine:latest`
- `fluent/fluentd:edge`
- `quay.io/operator-framework/helm-operator:v0.9.0`
- `company.local:5000/division/project/artifact:version`

Unlike in Ant matching of file system paths, the `:` is an important marker, but only at the end where it separates the version from the repository. Also, patterns that match repository names need to anticipate that there may be a registry name at the beginning if the image has been tagged for pushing to a registry.

Taking this into account, the name pattern wildcards are:

- `?` matches a single character
- `*` matches zero or more characters, up to the next slash or the tag separator
- `**` matches zero or more characters, up to the tag separator
- `**/` matches zero or more characters, up to the tag separator, and ensures that if any characters are matched, the final character matched is a slash

Examples of Ant-like Name Patterns

Pattern	Matches	Does Not Match
`**tomcat:jdk-11*`	- megacorp/tomcat:jdk-11-alpine - megacorp.com:5000/megacorp/project-x-tomcat:jdk-11	- megacorp/tomcat-*operator*:jdk-11 - megacorp/project-x-tomcat:*jdk-9-alpine*
`**/megacorp/tomcat:*alpine`	- megacorp/tomcat:alpine - megacorp.com:5000/megacorp/tomcat:jdk-11-alpine	- megacorp/tomcat:*jdk-11* - megacorp.com:5000/*ultra*megacorp/tomcat:jdk-11-alpine

Pattern	Matches	Does Not Match
`megacorp/*-operator:*`	• megacorp/tomcat-operator:alpine • megacorp/mysql-operator:latest	• megacorp/*tomcat*:jdk-11 • *megacorp.com:5000/*megacorp/tomcat-operator:alpine

4.3.2. Java Regular Expression Patterns

To indicate that a name pattern is a Java regular expression, prefix the regular expression with `%regex[` and suffix with `]`.

Examples of Java Regular Expression Patterns

Pattern	Matches	Does Not Match
`%regex[j(dk|re)-11]`	• megacorp/tomcat:jdk-11-alpine • openjdk-11:latest	• openjdk:11-alpine
`%regex[tomcat]`	• megacorp/tomcat:alpine • megacorp.com:5000/tomcat-projects/project-x:latest	• megacorp/topcat:edge

4.3.3. Name Pattern Lists

In goals such as **docker:stop** and **docker:remove** where multiple patterns are supported, separate patterns with commas.

Chapter 5. Maven Goals

This plugin supports the following goals which are explained in detail in the next sections.

Table 4. Plugin Goals

Goal	Description	Default Lifecycle Phase
docker:build	Build images	install
docker:start or **docker:run**	Create and start containers	pre-integration-test
docker:stop	Stop and destroy containers	post-integration-test
docker:push	Push images to a registry	deploy
docker:watch	Watch for doing rebuilds and restarts	
docker:remove	Remove images from local docker host	post-integration-test
docker:logs	Show container logs	
docker:copy	Copy container files and directories to the host	post-integration-test
docker:source	Attach docker build archive to Maven project	package
docker:save	Save images to a file	
docker:volume-create	Create a volume for containers to share data	pre-integration-test
docker:volume-remove	Remove a volume	post-integration-test

Note that all goals are orthogonal to each other. For example in order to start a container for your application you typically have to build its image before. `docker:start` does **not** imply building the image so you should use it then in combination with `docker:build`.

5.1. docker:build

This goal will build all images which have a `<build>` configuration section, or, if the global configuration variable `filter` (property: `docker.filter`) is set, only the images contained in this variable (comma separated) will be built.

There are two different modes how images can be built:

Inline plugin configuration

With an inline plugin configuration all information required to build the image is contained in the plugin configuration. By default its the standard XML based configuration for the plugin but can be switched to a property based configuration syntax as described in the section External configuration. The XML configuration syntax is recommended because of its more structured and typed nature.

When using this mode, the Dockerfile is created on the fly with all instructions extracted from the configuration given.

External Dockerfile or Docker archive

Alternatively an external Dockerfile template or Docker archive can be used. This mode is switched on by using one of these three configuration options within

- **contextDir** specifies docker build context if an external dockerfile is located outside of Docker build context. If not specified, Dockerfile's parent directory is used as build context.

- **dockerFile** specifies a specific Dockerfile path. The Docker build context directory is set to `contextDir` if given. If not the directory by default is the directory in which the Dockerfile is stored.

- **dockerArchive** specifies a previously saved image archive to load directly. Such a tar archive can be created with `docker save` or the **docker:save** goal. If a `dockerArchive` is provided, no `dockerFile` or `dockerFileDir` must be given.

- **dockerFileDir** (*deprecated*, use **contextDir**) specifies a directory containing a Dockerfile that will be used to create the image. The name of the Dockerfile is `Dockerfile` by default but can be also set with the option `dockerFile` (see below).

All paths can be either absolute or relative paths (except when both `dockerFileDir` and `dockerFile` are provided in which case `dockerFile` must not be absolute). A relative path is looked up in `${project.basedir}/src/main/docker` by default. You can make it easily an absolute path by using `${project.basedir}` in your configuration.

Adding assemblies in Dockerfile mode

Any additional files located in the `dockerFileDir` directory will also be added to the build context as well. You can also use an assembly if specified in an assembly configuration. However, you need to add the files on your own in the Dockerfile with an `ADD` or `COPY` command. The files of the assembly are stored in a build context relative directory `maven/` but can be changed by changing the assembly name with the option `<name>` in the assembly configuration.

E.g. the files can be added with

Example

```
COPY maven/ /my/target/directory
```

so that the assembly files will end up in `/my/target/directory` within the container.

If this directory contains a `.maven-dockerignore` (or alternatively, a `.maven-dockerexclude` file), then it is used for excluding files for the build. Each line in this file is treated as a FileSet exclude pattern as used by the maven-assembly-plugin. It is similar to `.dockerignore` when using Docker but has a slightly different syntax (hence the different name). Example `.maven-dockerexclude` or `.maven-dockerignore` is an example which excludes all compiled Java classes.

Example 1. Example `.maven-dockerexclude` or `.maven-dockerignore`

```
target/classes/**    ①
```

① Exclude all compiled classes

If this directory contains a `.maven-dockerinclude` file, then it is used for including only those files for the build. Each line in this file is also treated as a FileSet exclude pattern as used by the maven-assembly-plugin. Example `.maven-dockerinclude` shows how to include only jar file that have build to the Docker build context.

Example 2. Example `.maven-dockerinclude`

```
target/*.jar    ①
```

① Only add jar file to you Docker build context.

Except for the assembly configuration all other configuration options are ignored for now.

Simple Dockerfile build

When only a single image should be built with a Dockerfile no XML configuration is needed at all. All what need to be done is to place a `Dockerfile` into the top-level module directory, alongside to `pom.xml`. You can still configure global aspects in the plugin configuration, but as soon as you add an `<image>` in the XML configuration, you need to configure also the build explicitly.

The image name is by default set from the Maven coordinates (`%g/%a:%l`, see Image Name for an explanation of the params which are essentially the Maven GAV) This name can be set with the property `docker.name`.

If you want to add some `<run>` configuration to this image for starting it with `docker:run` then you can add an image configuration but without a `<build>` section in which case the Dockerfile will be picked up, too. This works only for a single image, though.

Filtering

fabric8-maven-plugin filters given Dockerfile with Maven properties, much like the `maven-resource-plugin` does. Filtering is enabled by default and can be switched off with a build config `<filter>false</filter>`. Properties which we want to replace are specified with the `${..}` syntax. Replacement includes Maven project properties such as `${project.artifactId}`, properties set in the build, command-line properties, and system properties. Unresolved properties remain untouched.

This partial replacement means that you can easily mix it with Docker build arguments and environment variable reference, but you need to be careful. If you want to be more explicit about the property delimiter to clearly separate Docker properties and Maven properties you can redefine the delimiter. In general, the `filter` option can be specified the same way as delimiters in the resource plugin. In particular, if this configuration contains a * then the parts left, and right of the asterisks are used as delimiters.

For example, the default `<filter>${*}</filter>` parse Maven properties in the format that we know. If you specify a single character for `<filter>` then this delimiter is taken for both, the start and the end. E.g a `<filter>@</filter>` triggers on parameters in the format @···@, much like in the `maven-invoker-plugin`. Use something like this if you want to clearly separate from Docker builds args. This form of property replacement works for Dockerfile only. For replacing other data in other files targeted for the Docker image, please use the `maven-resource-plugin` or an [assembly configuration](#) with filtering to make them available in the docker build context.

Example

The following example uses a Dockerfile in the directory `src/main/docker/demo` and replaces all properties in the format `@property@` within the Dockerfile.

```
<plugin>
 <configuration>
   <images>
     <image>
       <name>user/demo</name>
       <build>
         <dockerFileDir>demo</dockerFileDir>
         <filter>@</filter>
       </build>
     </image>
   </images>
 </configuration>
 ...
</plugin>
```

Build Plugins

This plugin supports so call **dmp-plugins** which are used during the build phase. dmp-plugins are enabled by just declaring a dependency in the plugin declaration:

```
<plugin>
  <groupId>io.fabric8</groupId>
  <artifactId>docker-maven-plugin</artifactId>

  <dependencies>
    <dependency>
      <groupId>io.fabric8</groupId>
      <artifactId>run-java-sh</artifactId>
      <version>1.2.2</version>
    </dependency>
  </dependencies>
</plugin>
```

These plugins contain a descriptor `META-INF/maven/io.fabric8/dmp-plugin` with class names, line-by-line:

```
io.fabric8.runsh.RunShLoader
```

During a build with `docker:build`, those classes are loaded and certain fixed method are called.

The following methods are supported:

Method	Description
addExtraFiles	A *static* method called by dmp with a single `File` argument. This will point to a directory `docker-extra` which can be referenced easily by a Dockerfile or an assembly. A dmp plugin typically will create an own subdirectory to avoid a clash with other dmp-plugins.

If a configured plugin does not provide method of this name and signature, then it will be simply ignored. Also, no interface needs to be implemented to keep the coupling low.

The following official dmp-plugins are known and supported:

Name	G,A	Description
run-java.sh	`fabric8.io`, `run-java`	General purpose startup script fo running Java applications. The dmp plugin creates a `target/docker-extra/run-java/run-java.sh` which can be included in a Dockerfile (see the example above). See the run-java.sh Documentation for more details.

Check out `samples/run-java` for a fully working example.

5.1.1. Configuration

All build relevant configuration is contained in the `<build>` section of an image configuration. The following configuration options are supported:

Table 5. Build configuration (<image>)

Element	Description
assemblies	Specifies multiple assembly configurations as described in Build Assembly
assembly	Specifies the assembly configuration as described in Build Assembly
args	Map specifying the value of Docker build args which should be used when building the image with an external Dockerfile which uses build arguments. The key-value syntax is the same as when defining Maven properties (or `labels` or `env`). This argument is ignored when no external Dockerfile is used. Build args can also be specified as properties as described in Build Args

Element	Description
buildOptions	Map specifying the build options to provide to the docker daemon when building the image. These options map to the ones listed as query parameters in the Docker Remote API and are restricted to simple options (e.g.: memory, shmsize). If you use the respective configuration options for build options natively supported by the build configuration (i.e. `squash`, `noCache`, `cleanup=remove` for buildoption `forcerm=1` and `args` for build args) then these will override any corresponding options given here. The key-value syntax is the same as when defining environment variables or labels as described in Setting Environment Variables and Labels.
buildx	Specifies the buildx configuration for multi-architecture images. See Buildx Options
createImageOptions	Map specifying the create image options to provide to the docker daemon when pulling or importing an image. These options map to the ones listed as query parameters in the Docker Remote API and are restricted to simple options (e.g.: fromImage, fromSrc, platform).
cleanup	Cleanup dangling (untagged) images after each build, including any stopped containers created from them. Also cleanup dangling images as a result of image tagging, auto-pulling a base image, or auto-pulling a cacheFrom image. Default is `try`, which tries to remove the old image, but doesn't fail the build if this is not possible (e.g. because the image is still used by a running container). Other possible values are `remove`, if you want to fail the build, or `none`, to skip cleanup altogether.
contextDir	Path to a directory used for the build's context. You can specify the `Dockerfile` to use with **dockerFile**, which by default is the Dockerfile found in the `contextDir`. The Dockerfile can be also located outside of the `contextDir`, if provided with an absolute file path. See External Dockerfile for details.
cmd	A command to execute by default (i.e. if no command is provided when a container for this image is started). See Startup Arguments for details.
compression	The compression mode how the build archive is transmitted to the docker daemon (`docker:build`) and how docker build archives are attached to this build as sources (`docker:source`). The value can be `none` (default), `gzip` or `bzip2`.
dockerFile	Path to a `Dockerfile` which also triggers *Dockerfile mode*. See External Dockerfile for details.
dockerFileDir (*deprecated* in favor of **contextDir**)	Path to a directory holding a `Dockerfile` and switch on *Dockerfile mode*. See External Dockerfile for details. *This option is deprecated in favor of _contextDir and will be removed for the next major release_.*
dockerArchive	Path to a saved image archive which is then imported. See Docker archive for details.
entryPoint	An entrypoint allows you to configure a container that will run as an executable. See Startup Arguments for details.
env	The environments as described in Setting Environment Variables and Labels.
filter	Enable and set the delimiters for property replacements. By default properties in the format `${..}` are replaced with Maven properties. You can switch off property replacement by setting this property to `false`. When using a single char like `@` then this is used as a delimiter (e.g `@···@`). See Filtering for more details.

Element	Description
from	The base image which should be used for this image. If not given this default to `busybox:latest` and is suitable for a pure data image.
fromExt	Extended definition for a base image. This field holds a map of defined in `<key>value</key>` format. The known keys are: • `<name>` : Name of the base image A provided `<from>` takes precedence over the name given here. This tag is useful for extensions of this plugin like the fabric8-maven-plugin which can evaluate the additional information given here.
healthCheck	Definition of a health check as described in Healthcheck
imagePullPolicy	Specific pull policy for the base image. This overwrites any global pull policy. See the globale configuration option imagePullPolicy for the possible values and the default.
loadNamePattern	Scan the archive specified in `dockerArchive` and find the actual repository and tag in the archive that matches this name pattern. After loading the archive, link the image name configured in the POM to the repository and tag matched in the archive.
labels	Labels as described in Setting Environment Variables and Labels.
maintainer	The author (`MAINTAINER`) field for the generated image
network	Set the networking mode for the RUN instructions during build
noCache	Don't use Docker's build cache. This can be overwritten by setting a system property `docker.noCache` when running Maven.
squash	Squash newly built layers into a single new layer. This can be overwritten by setting a system property `docker.squash` when running Maven.
cacheFrom	A list of `<image>` elements specifying image names to use as cache sources. During image build, it will attempt to pull these images, but not fail the build. Follows `imagePullPolicy` semantics.
optimise	if set to true then it will compress all the `runCmds` into a single `RUN` directive so that only one image layer is created.
ports	The exposed ports which is a list of `<port>` elements, one for each port to expose. Whitespace is trimmed from each element and empty elements are ignored. The format can be either pure numerical ("8080") or with the protocol attached ("8080/tcp").
shell	Shell to be used for the **runCmds**. It contains **arg** elements which are defining the executable and its params.
runCmds	Commands to be run during the build process. It contains **run** elements which are passed to the shell. Whitespace is trimmed from each element and empty elements are ignored. The run commands are inserted right after the assembly and after **workdir** into the Dockerfile. This tag is not to be confused with the `<run>` section for this image which specifies the runtime behaviour when starting containers.

Element	Description
skip	if set to true disables building of the image. This config option is best used together with a maven property
skipPush	if set to true disables pushing of the image. This config option is best used together with a maven property
skipTag	If set to `true` this plugin won't add any tags to images. Property: `docker.skip.tag`
tags	List of additional `tag` elements with which an image is to be tagged after the build. Whitespace is trimmed from each element and empty elements are ignored.
user	User to which the Dockerfile should switch to the end (corresponds to the `USER` Dockerfile directive).
volumes	List of `volume` elements to create a container volume. Whitespace is trimmed from each element and empty elements are ignored.
workdir	Directory to change to when starting the container.

From this configuration this Plugin creates an in-memory Dockerfile, copies over the assembled files and calls the Docker daemon via its remote API.

Example

```xml
<build>
  <from>java:8u40</from>
  <maintainer>john.doe@example.com</maintainer>
  <tags>
    <tag>latest</tag>
    <tag>${project.version}</tag>
  </tags>
  <ports>
    <port>8080</port>
  </ports>
  <volumes>
    <volume>/path/to/expose</volume>
  </volumes>
  <buildOptions>
    <shmsize>2147483648</shmsize>
  </buildOptions>

  <shell>
    <exec>
      <arg>/bin/sh</arg>
      <arg>-c</arg>
    </exec>
  </shell>
  <runCmds>
    <run>groupadd -r appUser</run>
    <run>useradd -r -g appUser appUser</run>
  </runCmds>

  <entryPoint>
    <!-- exec form for ENTRYPOINT -->
    <exec>
      <arg>java</arg>
      <arg>-jar</arg>
      <arg>/opt/demo/server.jar</arg>
    </exec>
  </entryPoint>

  <assembly>
    <mode>dir</mode>
    <targetDir>/opt/demo</targetDir>
    <descriptor>assembly.xml</descriptor>
  </assembly>
</build>
```

In order to see the individual build steps you can switch on `verbose` mode either by setting the property `docker.verbose` or by using `<verbose>true</verbose>` in the Global configuration

5.1.2. Assembly

The `<assembly>` element within `<build>` has an XML structure and defines how build artifacts and other files can enter the Docker image. Multiple `<assembly>` elements may be specified by adding them to an `<assemblies>` element. If both `<assembly>` and `<assemblies>` are present in `<build>`, the `<assembly>` element is treated as if it were the last child of `<assemblies>`.

When multiple assemblies are provided, each will be added as a separate layer in the image.

Table 6. Assembly Configuration (<image> : <build>)

Element	Description
name	Assembly name, which is `maven` by default. This name is used for the archives and directories created during the build. This directory holds the files specified by the assembly. If an external Dockerfile is used than this name is also the relative directory which contains the assembly files. If multiple assemblies are provided, they must each have a unique name.
targetDir	Directory under which the files and artifacts contained in the assembly will be copied within the container. The default value for this is `/<assembly name>`, so `/maven` if **name** is not set to a different value. This option has no meaning when an external Dockerfile is used.
inline	Inlined assembly descriptor as described in Assembly Descriptor below.
descriptor	Path to an assembly descriptor file, whose format is described Assembly Descriptor below.
descriptorRef	Alias to a predefined assembly descriptor. The available aliases are also described in Assembly Descriptor below.
dockerFileDir	Directory containing an external Dockerfile. *This option is deprecated, please use <dockerFileDir> directly in the <build> section.*
exportTargetDir	Specification whether the `targetDir` should be exported as a volume. This value is `true` by default except in the case the `targetDir` is set to the container root (`/`). It is also `false` by default when a base image is used with `from` since exporting makes no sense in this case and will waste disk space unnecessarily.
ignorePermissions	Specification if existing file permissions should be ignored when creating the assembly archive with a mode `dir`. This value is `false` by default. *This property is deprecated, use a `permissions` of `ignore` instead.*
mode	Mode how the how the assembled files should be collected: • `dir` : Files are simply copied (default), • `tar` : Transfer via tar archive • `tgz` : Transfer via compressed tar archive • `zip` : Transfer via ZIP archive The archive formats have the advantage that file permission can be preserved better (since the copying is independent from the underlying files systems), but might triggers internal bugs from the Maven assembler (as it has been reported in #171)

Element	Description
permissions	Permission of the files to add: • `ignore` to use the permission as found on files regardless on any assembly configuration • `keep` to respect the assembly provided permissions, `exec` for setting the executable bit on all files (required for Windows when using an assembly mode `dir`) • `auto` to let the plugin select `exec` on Windows and `keep` on others. `keep` is the default value.
tarLongFileMode	Sets the TarArchiver behaviour on file paths with more than 100 characters length. Valid values are: "warn"(default), "fail", "truncate", "gnu", "posix", "posix_warn" or "omit"
user	User and/or group under which the files should be added. The user must already exist in the base image. It has the general format `user[:group[:run-user]]`. The user and group can be given either as numeric user- and group-id or as names. The group id is optional. If a third part is given, then the build changes to user `root` before changing the ownerships, changes the ownerships and then change to user `run-user` which is then used for the final command to execute. This feature might be needed, if the base image already changed the user (e.g. to 'jboss') so that a `chown` from root to this user would fail. (***This third user part has been marked as deprecated and will not be supported in future versions of this plugin.***) For example, the image `jboss/wildfly` use a "jboss" user under which all commands are executed. Adding files in Docker always happens under the UID root. These files can only be changed to "jboss" is the `chown` command is executed as root. For the following commands to be run again as "jboss" (like the final `standalone.sh`), the plugin switches back to user `jboss` (this is this "run-user") after changing the file ownership. For this example a specification of `jboss:jboss:jboss` would be required.

In the event you do not need to include any artifacts with the image, you may safely omit this element from the configuration.

Assembly Descriptor

With using the `inline`, `descriptor` or `descriptorRef` option it is possible to bring local files, artifacts and dependencies into the running Docker container. A `descriptor` points to a file describing the data to put into an image to build. It has the same [format](#) as for creating assemblies with the [maven-assembly-plugin](#) with following exceptions:

• `<formats>` are ignored, the assembly will allways use a directory when preparing the data container (i.e. the format is fixed to `dir`)

• The `<id>` is ignored since only a single assembly descriptor is used (no need to distinguish

multiple descriptors)

Also you can inline the assembly description with a `inline` description directly into the pom file. Adding the proper namespace even allows for IDE autocompletion. As an example, refer to the profile `inline` in the `data-jolokia-demo` 's pom.xml.

Alternatively `descriptorRef` can be used with the name of a predefined assembly descriptor. The following symbolic names can be used for `descriptorRef`:

Table 7. Predefined Assembly Descriptors

Assembly Reference	Description
artifact-with-dependencies	Attaches project's artifact and all its dependencies. Also, when a `classpath` file exists in the target directory, this will be added to.
artifact	Attaches only the project's artifact but no dependencies.
dependencies	Attaches only the project's dependencies. Also, when a `classpath` file exists in the target directory, this will be added too.
release-dependencies	Attaches only the project's released (non-snapshot) dependencies.
snapshot-dependencies	Attaches only the project's snapshot dependencies.
project	Attaches the whole Maven project but without the `target/` directory.
rootWar	Copies the artifact as `ROOT.war` to the exposed directory. I.e. Tomcat will then deploy the war under the root context.

Examples

```
<images>
  <image>
    <build>
      <assembly>
        <descriptorRef>artifact-with-dependencies</descriptorRef>
        .....
```

will add the created artifact with the name `${project.build.finalName}.${artifact.extension}` and all jar dependencies in the `targetDir` (which is `/maven` by default).

All declared files end up in the configured `targetDir` (or `/maven` by default) in the created image.

```xml
<images>
  <image>
    <build>
      <assemblies>
        <assembly>
            <name>deps-release</name>
            <descriptorRef>release-dependencies</descriptorRef>
            <targetDir>/work/lib</targetDir>
        </assembly>
        <assembly>
            <name>deps-snapshot</name>
            <descriptorRef>snapshot-dependencies</descriptorRef>
            <targetDir>/work/lib</targetDir>
        </assembly>
        <assembly>
            <descriptorRef>artifact</descriptorRef>
            <targetDir>/work</targetDir>
        </assembly>
      </assemblies>
      .....
    </build>
  </image>
</images>
```

will create three layers:

1. Release dependencies (in jar format) added to /work/lib

2. Snapshot dependencies (in jar format) added to /work/lib

3. The created artifact with the name `${project.build.finalName}.${artifact.extension}` added to /work

Maven peculiarities when including the artifact

If the assembly references the artifact to build with this pom, it is required that the `package` phase is included in the run. Otherwise the artifact file, can't be found by `docker:build`. This is an old outstanding issue of the assembly plugin which probably can't be fixed because of the way how Maven works. We tried hard to workaround this issue and in 90% of all cases, you won't experience any problem. However, when the following warning happens which might lead to the given error:

```
[WARNING] Cannot include project artifact: io.fabric8:helloworld:jar:0.20.0; it
doesn't have an associated file or directory.
[WARNING] The following patterns were never triggered in this artifact inclusion
filter:
o  'io.fabric8:helloworld'

[ERROR] DOCKER> Failed to create assembly for docker image  (with mode 'dir'): Error
creating assembly archive docker: You must set at least one file.
```

then you have two options to fix this:

- Call `mvn package docker:build` to explicitly run "package" and "docker:build" in a chain.

- Bind `build` to an to an execution phase in the plugin's definition. By default `docker:build` will bind to the `install` phase is set in an execution. Then you can use a plain `mvn install` for building the artifact and creating the image.

```
<executions>
  <execution>
    <id>docker-build</id>
    <goals>
        <goal>build</goal>
    </goals>
  </execution>
</executions>
```

Example

In the following example a dependency from the pom.xml is included and mapped to the name `jolokia.war`. With this configuration you will end up with an image, based on `busybox` which has a directory `/maven` containing a single file `jolokia.war`. This volume is also exported automatically.

```
<assembly>
  <inline>
    <dependencySets>
      <dependencySet>
        <includes>
          <include>org.jolokia:jolokia-war</include>
        </includes>
        <outputDirectory>.</outputDirectory>
        <outputFileNameMapping>jolokia.war</outputFileNameMapping>
      </dependencySet>
    </dependencySets>
  </inline>
</assembly>
```

Another container can now connect to the volume an 'mount' the `/maven` directory. A container from `consol/tomcat-7.0` will look into `/maven` and copy over everything to `/opt/tomcat/webapps` before starting Tomcat.

If you are using the `artifact` or `artifact-with-dependencies` descriptor, it is possible to change the name of the final build artifact with the following:

Example

```
<build>
  <finalName>your-desired-final-name</finalName>
  ...
</build>
```

Please note, based upon the following documentation listed here, there is no guarantee the plugin creating your artifact will honor it in which case you will need to use a custom descriptor like above to achieve the desired naming.

Currently the `jar` and `war` plugins properly honor the usage of `finalName`.

5.1.3. Startup Arguments

Using `entrypoint` and `cmd` it is possible to specify the entry point or cmd for a container.

The difference is, that an `entrypoint` is the command that always be executed, with the `cmd` as argument. If no `entrypoint` is provided, it defaults to `/bin/sh -c` so any `cmd` given is executed with a shell. The arguments given to `docker run` are always given as arguments to the `entrypoint`, overriding any given `cmd` option. On the other hand if no extra arguments are given to `docker run` the default `cmd` is used as argument to `entrypoint`.

> See this stackoverflow question for a detailed explanation.

An entry point or command can be specified in two alternative formats:

Table 8. Entrypoint and Command Configuration

Mode	Description
shell	Shell form in which the whole line is given to `shell -c` for interpretation.
exec	List of arguments (with inner `<args>`) arguments which will be given to the `exec` call directly without any shell interpretation.

Either shell or params should be specified.

Example

```
<entryPoint>
   <!-- shell form  -->
   <shell>java -jar $HOME/server.jar</shell>
</entryPoint>
```

or

Example

```
<entryPoint>
    <!-- exec form  -->
    <exec>
      <arg>java</arg>
      <arg>-jar</arg>
      <arg>/opt/demo/server.jar</arg>
    </exec>
</entryPoint>
```

This can be formulated also more dense with:

Example

```
<!-- shell form  -->
<entryPoint>java -jar $HOME/server.jar</entryPoint>
```

or

Example

```
<entryPoint>
  <!-- exec form  -->
  <arg>java</arg>
  <arg>-jar</arg>
  <arg>/opt/demo/server.jar</arg>
</entryPoint>
```

5.1.4. Build Args

As described in section Configuration for external Dockerfiles Docker build arg can be used. In addition to the configuration within the plugin configuration you can also use properties to specify them:

- Set a system property when running Maven, eg.: `-Ddocker.buildArg.http_proxy=http://proxy:8001`. This is especially useful when using predefined Docker arguments for setting proxies transparently.

- Set a project property within the `pom.xml`, eg.:

Example

```
    <docker.buildArg.myBuildArg>myValue</docker.buildArg.myBuildArg>
```

Please note that the system property setting will always override the project property. Also note that for all properties which are not Docker predefined properties, the external Dockerfile must contain an `ARGS` instruction.

5.1.5. Healthcheck

Healthchecks has been introduced since Docker 1.12 and are a way to tell Docker how to test a container to check that it's still working. With a health check you specify a command which is periodically executed and checked for its return value. If the healtcheck return with an exit 0 the container is considered to be healthy, if it returns with 1 then the container is not working correctly.

The healtcheck configuration can have the following options

Table 9. Healthcheck Configuration

Element	Description
cmd	Command to execute, which can be given in an shell or exec format as described in Startup Arguments.
interval	Interval for how often to run the healthcheck. The time is specified in seconds, but a time unit can be appended to change this.
mode	Mode of the healthcheck. This can be `cmd` which is the default and specifies that the health check should be executed. Or `none` to disable a health check from the base image. Only use this option with `none` for disabling some healthcheck from the base image.
retries	How many retries should be performed before the container is to be considered unhealthy.
startPeriod	Initialization time for containers that need time to bootstrap. Probe failure during that period will not be counted towards the maximum number of retries. However, if a health check succeeds during the start period, the container is considered started and all consecutive failures will be counted towards the maximum number of retries. Given in seconds, but another time unit can be appended.
timeout	Timeout after which healthckeck should be stopped and considered to have failed. Given in seconds, but another time unit can be appended.

The following example queries an URL every 10s as an healthcheck:

Example

```
<healthCheck>
  <!-- Check every 5 minutes -->
  <interval>5m</interval>
  <!-- Fail if no response after 3 seconds -->
  <timeout>3s</timeout>
  <!-- Allow 30 minutes for the container to start before being flagged as unhealthy
-->
  <startPeriod>30m</startPeriod>
  <!-- Fail 3 times until the container is considerd unhealthy -->
  <retries>3</retries>
  <!-- Command to execute in shell form -->
  <cmd>curl -f http://localhost/ || exit 1</cmd>
</healthCheck>
```

5.1.6. Multi-Architecture Build

Buildx is enabled when there is a non-empty `<platform>` element inside the `<buildx>` configuration.

The local image cache cannot hold multi-architecture images nor can it have two platform specific images of the same name. Thus the `build` goal will build and save a single-architecture image to the local image cache if possible:

- If the `<platform>` element contains a single platform, that image will be built.

- If the `<platform>` element contains more than one platform including the native platform, the native platform be used.

- If the `<platform>` element contains more than one platform not including the native platform, no image will be built.

These rules only apply to the image built and loaded into the local image cache with the `build` goal. They do not apply to the `push` goal which will always build and push either a single-architecture or multi-architecture image with whatever platforms are specified in the `<platform>` element.

The recommended `<buildx>` configuration is to specify all supported platforms, including the native platform, in the `<platform>` element. This allows local integration testing of the build image from the local cache. During install or deploy phase, the build machine will build and push a multi-architecture image containing all specified platforms to the registry. Any downstream consumers, regardless of native architecture, will be able to use the multi-architecture image.

The `<buildx>` element within `<build>` defines how to build multi-architecture images.

Table 10. BuildX Options

Element	Description
builderName	Name of builder to use with buildx. If not supplied, the builder is named `maven`. The builder is created as necessary. The builder manages the build cache.
configFile	Configuration file for builder. Non-absolute files are relative to the maven project directory. If configFile starts with `~/`, the configuration file is relative to the user's home directory.
dockerStateDir	State directory for docker builder. This directory holds docker builder configurations and context state. Sharing a state directory across builds will share the cache and will decrease pull times. Non-absolute files are relative to the maven project directory. If dockerConfigDir starts with `~/`, the configuration directory is relative to the user's home directory.
platforms	A list of `<platform>` elements specifying platform to build. A platform has syntax of `OS/architecture` (e.g. linux/amd64, linux/arm64, darwin/amd64). Each `<platform>` element may have a comma separated list of platforms. Empty `<platform>` elements are ignored. If no platform architecture is specified, buildx is **not** used. You can use
attestations	The configuration of attestation modes. The `<provenance>` element may be set to `min`, `max`, or `false`. The `<sbom>` element may be set to `true` or `false`. The `<provenance>` element defaults to `min` and the `<sbom>` element defaults to `false`.

Examples

The recommended configuration is setting a top level property with the list of platforms to build.

```
<properties>
  <docker.platforms></docker.platforms>
</properties>
```

Then in the image configuration, use the following;

```
<configuration>
  <images>
    <image>
      <name>${project.groupId}.${project.artifactId}</name>
      <build>
        <buildx>
          <platforms>
            <platform>${docker.platforms}</platform>
          </platforms>
        </buildx>
        <!-- add other configuration ... -->
      </build>
    </image>
  </images>
</configuration>
```

You can now override the built platforms using a command line define:

```
mvn clean deploy -Ddocker.platforms=linux/amd64,linux/arm64
```

5.2. docker:start

This goal creates and starts docker containers. This goal evaluates the configuration's `<run>` section of all given (and enabled images).

Also you can specify `docker.follow` as system property so that the `docker:start` will never return but block until CTRL-C is pressed. That is similar to the option `-i` for `docker run`. This will automatically switch on `showLogs` so that you can see what is happening within the container. Also, after stopping with CTRL-C, the container is stopped (but not removed so that you can make postmortem analysis). `docker:run` is an alias for `docker:start` with `docker.follow` enabled.

By default container specific properties are exposed as Maven properties. These properties have the format `docker.container.<alias>.<prop>` where `<alias>` is the name of the container (see below) and `<prop>` is one of the following container properties:

Table 11. Properties provided

Property	Description
ip	Internal IP address of the container.
id	Container id
net.*<network>*. **ip**	Internal IP address of the container in the specified custom network. This works only for custom networks.

Instead of the `<alias>` a fixed property key can be configured in the image's `<run>` configuration with the option `exposedPropertyKey`.

For example the Maven property `docker.container.tomcat.ip` would hold the Docker internal IP for a container with an alias "tomcat". You can set the global configuration **exposeContainerInfo** to an empty string to not expose container information that way or to a string for an other prefix than `docker.container`.

5.2.1. Configuration

In addition to the Global configuration, this goal supports the following global configuration options.

Table 12. Start options

Element	Description	Property
containerNamePattern	Default pattern for naming all containers when they are created. See Container Names for details.	`docker.containerNamePattern`
showLogs	In order to switch on globally the logs **showLogs** can be used as global configuration (i.e. outside of `<images>`). If set it will print out all standard output and standard error messages for all containers started. As value the images for which logs should be shown can be given as a comma separated list. This is probably most useful when used from the command line as system property `docker.showLogs`.	`docker.showLogs`
startParallel	Starts docker images in parallel while dependencies expressed as Link or dependsOn are respected. This option can significantly reduce the startup time because independent containers do not need to wait for each other.	`docker.startParallel`

The `<run>` configuration element knows the following sub elements:

Table 13. Run configuration (<image>)

Element	Description
autoRemove	If `true` automatically remove the container when it exits. This has no effect if Restart Policy has been set.
capAdd	List of `add` elements to specify kernel parameters to add to the container.
capDrop	List of `drop` elements to specify kernel parameters to remove from the container.
sysctls	Map of namespaced kernel parameters (sysctls) to set in the container.
cmd	Command which should be executed at the end of the container's startup. If not given, the image's default command is used. See Startup Arguments for details.

Element	Description
containerNamePattern	Pattern for naming the container when it is created. See Container Naming Strategy for details.
domainname	Domain name for the container
dns	List of `host` elements specifying dns servers for the container to use
dnsSearch	List of `host` elements specifying dns search domains
entrypoint	Entry point for the container. See Startup Arguments for details.
env	Environment variables as subelements which are set during startup of the container. They are specified in the typical maven property format as described Environment and Labels.
envPropertyFile	Path to a property file holding environment variables. If given, the variables specified in this property file overrides the environment variables specified in the configuration.
extraHosts	List of `host` elements in the form `host:ip` to add to the container's `/etc/hosts` file. Additionally, you may specify a `host` element in the form `host:host` to have the right side host ip address resolved at container startup.
exposedPropertyKey	Set the property part for the exposed container properties as described above. This will take precedence of the image's `alias` which is the default value. For example, when this property is set to `jboss`, then for this container its IP address is exposed in Maven property `docker.container.jboss.ip` regardless how the image is named.
hostname	Hostname of the container
imagePullPolicy	Specific pull policy for downloading the image. This overwrites any global pull policy. See the global imagePullPolicy configuration option for the possible values and the default.
labels	Labels which should be attached to the container. They are specified in the typical maven property format as described in Environment and Labels.
links	Network links for connecting containers together as described in Network Links.
log	Log configuration for whether and how log messages from the running containers should be printed. This also can configure the log driver to use. See Logging for a detailed description.
isolation	This option sets container's isolation technology. See Isolation for a detailed description.
memory	Memory limit in bytes.
memorySwap	Total memory limit (memory + swap) in bytes. Set `memorySwap` equal to `memory` to disable swap. Set to `-1` to allow unlimited swap.
namingStrategy	**This option is deprecated, please use a `containerNamePattern` instead** Naming strategy for how the container name is created: • **none** : uses randomly assigned names from docker (default) • **alias** : uses the `alias` specified in the `image` configuration. An error is thrown, if a container already exists with this name.
network	Network configuration for your container.

Element	Description
portPropertyFile	File path into which the mapped port properties are written. The format of this file and its purpose are also described in Port mapping
ports	Port mappings for exposing container ports to host ports.
platform	Specify an explicit platform to use when starting a docker container. May be set with property `docker.platform`. Defaults to native platform.
privileged	If `true` give container full access to host
readOnly	If `true` mount the container's root filesystem as read only
restartPolicy	Restart Policy
securityOpts	List of `<opt>` elements to specify kernel security options to add to the container. See below for an example.
shmSize	Size of `/dev/shm` in bytes.
skip	If `true` disable creating and starting of the container. This option is best used together with a Maven property which can be set from the outside.
stopMode	Specifies how to stop a running container. It supports the modes `graceful` and `kill` as values, with `graceful` being the default.
tmpfs	List countaintin `<mount>` elements for directories to mount with a temporary filesystem. Optionally, mount options can be appended after a ':'. See below for an example.
ulimits	ulimits for the container. This list contains `<ulimit>` elements which three sub elements: • `<name>` : The ulimit to set (e.g. `memlock`). Please refer to the Docker documentation for the possible values to set • `<hard>` : The hard limit • `<soft>` : The soft limit See below for an example.
user	User used inside the container
volumes	Volume configuration for binding to host directories and from other containers. See Volumes for details.
wait	Condition which must be fulfilled for the startup to complete. See Wait for all possible ways to wait for a startup condition.
workingDir	Working directory for commands to run in

Example

```
<run>
  <env>
    <CATALINA_OPTS>-Xmx32m</CATALINA_OPTS>
    <JOLOKIA_OFF/>
  </env>
  <labels>
    <environment>development</environment>
    <version>${project.version}</version>
  </labels>
  <ports>
    <port>jolokia.port:8080</port>
  </ports>
  <ulimits>
    <ulimit>
        <name>memlock</name>
        <hard>-1</hard>
        <soft>-1</soft>
    </ulimit>
  </ulimits>
  <tmpfs>
    <mount>/var/lib/mysql:size=10m</mount>
    <mount>/opt/mydata</mount>
  </tmpfs>
  <securityOpts>
    <opt>seccomp=unconfined</opt>
  </securityOpts>
  <links>
    <link>db</db>
  </links>
  <wait>
    <http>
      <url>http://localhost:${jolokia.port}/jolokia</url>
    </http>
    <time>10000</time>
  </wait>
  <log>
    <prefix>DEMO</prefix>
    <date>ISO8601</date>
    <color>blue</color>
  </log>
  <cmd>java -jar /maven/docker-demo.jar</cmd>
</run>
```

5.2.2. Environment and Labels

When creating a container one or more environment variables can be set via configuration with the `env` parameter

```
<env>
  <JAVA_HOME>/opt/jdk8</JAVA_HOME>
  <CATALINA_OPTS>-Djava.security.egd=file:/dev/./urandom</CATALINA_OPTS>
</env>
```

If you put this configuration into profiles you can easily create various test variants with a single image (e.g. by switching the JDK or whatever).

It is also possible to set the environment variables from the outside of the plugin's configuration with the parameter `envPropertyFile`. If given, this property file is used to set the environment variables where the keys and values specify the environment variable. Environment variables specified in this file override any environment variables specified in the configuration.

Labels can be set inline the same way as environment variables:

Example

```
<labels>
   <com.example.label-with-value>foo</com.example.label-with-value>
   <version>${project.version}</version>
   <artifactId>${project.artifactId}</artifactId>
</labels>
```

5.2.3. Port Mapping

The `<ports>` configuration contains a list of port mappings. Whitespace is trimmed from each element and empty elements are ignored. Each mapping has multiple parts, each separate by a colon. This is equivalent to the port mapping when using the Docker CLI with option `-p`.

A `port` stanza may take one of the following forms:

Table 14. Port mapping format

Format	Description
18080:8080	Tuple consisting of two numeric values separated by a `:`. This form will result in an explicit mapping between the docker host and the corresponding port inside the container. In the above example, port 18080 would be exposed on the docker host and mapped to port 8080 in the running container.

Format	Description
host.port:80	Tuple consisting of a string and a numeric value separated by a `:`. In this form, the string portion of the tuple will correspond to a Maven property. If the property is undefined when the `start` task executes, a port will be dynamically selected by Docker in the ephemeral port range and assigned to the property which may then be used later in the same POM file. The ephemeral port range is configured by the `/proc/sys/net/ipv4/ip_local_port_range` kernel parameter, which typically ranges from 32678 to 61000. If the property exists and has numeric value, that value will be used as the exposed port on the docker host as in the previous form. In the above example, the docker service will elect a new port and assign the value to the property `host.port` which may then later be used in a property expression similar to `<value>${host.port}</value>`. This can be used to pin a port from the outside when doing some initial testing similar to `mvn -Dhost.port=10080 docker:start`
bindTo:host.port:80	Tuple consisting of two strings and a numeric value separated by a `:`. In this form, `bindTo` is an ip address on the host the container should bind to. As a convenience, a hostname pointing to the docker host may also be specified. The container will fail to start if the hostname can not be resolved.
+**host.ip:host.port**:80	Tuple consisting of two strings and a numeric value separated by a `:`. In this form, the host ip of the container will be placed into a Maven property name `host.ip`. If docker reports that value to be `0.0.0.0`, the value of `docker.host.address` will be substituted instead. In the event you want to use this form and have the container bind to a specific hostname/ip address, you can declare a Maven property of the same name (`host.ip` in this example) containing the value to use. `host:port` works in the same way as described above.

By default TCP is used as protocol but you can also use UDP by appending '/udp' to the port number.

The following are examples of valid configuration entries:

Example

```xml
<properties>
  <bind.host.ip>1.2.3.4</bind.host.ip>
  <bind.host.name>some.host.pvt</bind.host.name>
</properties>

...

<ports>
  <port>18080:8080</port>
  <port>15060:5060/udp</port>
  <port>host.port:80</port>
  <port>127.0.0.1:80:80</port>
  <port>localhost:host.port:80</port>
  <port>+container.ip.property:host.port:5678</port>
  <port>+bind.host.ip:host.port:5678</port>
  <port>+bind.host.name:5678:5678</port>
</ports>
```

Another useful configuration option is `portPropertyFile` which can be used to write out the container's host ip and any dynamic ports that have been resolved. The keys of this property file are the property names defined in the port mapping configuration and their values those of the corresponding docker attributes.

This property file might be useful with tests or with other maven plugins that will be unable to use the resolved properties because they can only be updated after the container has started and plugins resolve their properties in an earlier lifecycle phase.

If you don't need to write out such a property file and thus don't need to preserve the property names, you can use normal maven properties as well. E.g. `${host.var}:${port.var}:8080` instead of `+host.var:port.var:8080`.

5.2.4. Links

The `<links>` configuration contains a list of containers that should be linked to this container according to Docker Links. Each link can have two parts where the optional right side is separated by a `:` and will be used as the name in the environment variables and the left side refers to the name of the container linking to. This is equivalent to the linking when using the Docker CLI `--link` option.

Example for linking to a container with name or alias *postgres* :

Example

```xml
<links>
  <link>postgres:db</link>
</links>
```

This will create the following environment variables, given that the postgres image exposes TCP port 5432:

Example

```
DB_NAME=/web2/db
DB_PORT=tcp://172.17.0.5:5432
DB_PORT_5432_TCP=tcp://172.17.0.5:5432
DB_PORT_5432_TCP_PROTO=tcp
DB_PORT_5432_TCP_PORT=5432
DB_PORT_5432_TCP_ADDR=172.17.0.5
```

Additionally, each `<link>` element can specify a comma separated set of links. Comma (and whitespace) can be used to separate links since valid docker link names/aliases contain only characters, digits, underscores, periods and dashes.

Example

```
<links>
  <link>postgres:db, search, saml:identity</link>
</links>
```

If you wish to link to existing containers not managed by the plugin, you may do so by specifying the container name obtained via `docker ps` in the configuration.

Please note that the link behaviour also depends on the network mode selected. Links as described are referred to by Docker as *legacy links* and might vanish in the future. For custom networks no environments variables are set and links create merely network aliases for the linked container. To express start order dependencies using custom networks refer to the dependsOn configuration.

For a more detailed documentation for the new link handling please refer to the Docker network documentation

5.2.5. Network

The `<network>` element in the `<run>` configuration section can be used to configure the network mode of the container. This is now the preferred way for linking containers together. It knows the following sub elements:

Table 15. Network configuration

Element	Description
mode	The network mode, which can be one of the following values: • **bridge** : Bridged mode with the default Docker bridge (default) • **host** : Share the Docker host network interfaces • **container** : Connect to the network of the specified container. The name of the container is taken from the `<name>` element. • **custom** : Use a custom network, which must be created before by using `docker network create`. Alternatively you can set the `docker.autoCreateCustomNetworks` global configuration parameter to `true` to automatically create custom networks. Custom networks are available for Docker 1.9 and newer. For more about the networking options please refer to the Docker documentation. • **none** : No network will be setup.
name	For mode `container` this is the container name, which is this image alias. For Mode `custom` this is the name of the custom network.
alias	One or more alias element can be provided which gives a way for a container to be discovered by alternate names by any other container within the scope of a particular network. This configuration only has effect for when the network mode is `custom`. More than one alias can be given by providing multiple entries.

If no `mode` is given but a `name`, then a `custom` network mode is assumed. For the simple modes which does not take an argument (`none`, `bridge` or `host`) a single `<net>` *mode* `</net>` can be used as alternative to using `<network>` with a `<mode>` subelement.

Example <network>

```
<network>
    <mode>custom</mode>
    <name>my-network</name>
    <alias>box1</alias>
    <alias>box2</alias>
</network>
```

or for a simple `host` network:

Example <net>

```
<net>host</net>
```

5.2.6. Depends-On

Custom networks do not provide a mechanism like `<links>` to express strong links between containers. They are normally not required because docker ensures that all containers within the same custom network can eventually resolve each other via DNS.

Your containers should preferably be able to deal with temporarily unresolvable dependencies but in some cases it is helpful to be able to rely the availability of other infrastructure containers.

The `<dependsOn>` configuration can be used to expresses custom network dependencies between your containers. `docker:start` will ensure that all dependencies a container depends on are completely started (fulfilling all `<wait>` conditions) before the depending container is started.

Additionally, each `<container>` element can specify a comma separated set of containers. Comma (and whitespace) can be used to separate containers since valid docker container names contain only characters, digits, underscores, periods and dashes.

Example

```xml
<configuration>
  <!-- .... -->
  <run>
    <dependsOn>
      <container>postgres</container>
      <container>logstash</container>
    </dependsOn>
  </run>
</configuration>
```

5.2.7. Restart Policy

Specify the behavior to apply when the container exits. These values can be specified withing a `<restartPolicy>` section with the following sub-elements:

Table 16. Restart Policy configuration

Element	Description
name	Restart policy name, choose from: • `always` (*v1.15*) always restart • `on-failure` (*v1.15*) restart on container non-exit code of zero
retry	If `on-failure` is used, controls max number of attempts to restart before giving up.

The behavior to apply when the container exits. The value is an object with a name property of either "always" to always restart or "on-failure" to restart only when the container exit code is non-zero. If on-failure is used, MaximumRetryCount controls the number of times to retry before giving up. The default is not to restart. (optional)

5.2.8. Volumes

A container can bind (or "mount") volumes from various source when starting up: Either from a directory of the host system or from another container which exports one or more directories. The mount configuration is specified within a `<volumes>` section of the run configuration. It can contain the following sub elements:

Table 17. Volume configuration

Element	Description
from	List of `<image>` elements which specify image names or aliases of containers whose volumes should be imported.
bind	List of `<volume>` specifications (or *host mounts*). Use `/path` to create and expose a new volume in the container, `/host_path:/container_path` to mount a host path into the container and `/host_path:/container_path:ro` to bind it read-only.

Volumes example

```
<volumes>
  <bind>
    <volume>/logs</volume>
    <volume>/opt/host_export:/opt/container_import</volume>
  </bind>
  <from>
    <image>jolokia/docker-demo</image>
  </from>
</volumes>
```

In this example the container creates a new volume named `/logs` on the container and mounts `/opt/host_export` from the host as `/opt/container_import` on the container. In addition all exported volumes from the container which has been created from the image `jolokia/docker-demo` are mounted directly into the container (with the same directory names under which the exporting container exposes these directories). This image must be also configured for this plugin. Instead of the full image name, an alias name can be used, too.

If a volume name instead of a path is referenced to in `<bind>` and a volume configuration exists with this name, then this this volume is created upfront with the provided options instead of using default options.

You can use Maven variables in the path specifications. This should even work for boot2docker and docker-machine:

Example with absolute paths

```
<volumes>
  <bind>
    <volume>${project.build.directory}/${project.artifactId}-
${project.version}:/usr/local/tomcat/webapps/${project.name}</volume>
    <volume>${project.basedir}/data:/data</volume>
  </bind>
</volumes>
```

You can also use relative paths. Relative paths are interpreted relative to the Maven project base directory. Paths that begin with `~` are interpreted relative to the JVM's `HOME` or `user.home` directory.

Example with relative paths

```
<volumes>
  <bind>
    <volume>src/main/webapps/foo:/usr/local/tomcat/webapps/foo</volume>
    <volume>./target:/data</volume>
    <volume>~:/home/user</volume>
    <volume>~/.m2/repository:/home/user/.m2/repository</volume>
  </bind>
</volumes>
```

If you wish to mount volumes from an existing container not managed by the plugin, you may do by specifying the container name obtained via `docker ps` in the configuration.

5.2.9. Wait

While starting a container is it possible to block the execution until some condition is met. These conditions can be specified within a `<wait>` section which the following sub-elements:

Table 18. Wait configuration

Element	Description
http	HTTP ping check which periodically polls an URL. It knows the following sub-elements: • **url** holds an URL and is mandatory • **method** Optional HTTP method to use. • **status** Status code which if returned is considered to be a successful ping. This code can be given either as a single number (200) or as a range (200..399). The default is `200..399` • **allowAllHosts** If `url` is an HTTPS url and this option is set, then server certificates are not validated. By default they are checked for a proper CA signature.
log	Regular expression which is applied against the log output of an container and blocks until the pattern is matched. You can use `(?s)` in the pattern to switch on multi line matching.
time	Time in milliseconds to block.
kill	Time in milliseconds between sending `SIGTERM` and `SIGKILL` when stopping a container. Since docker itself uses second granularity, you should use at least 1000 milliseconds.
shutdown	Time to wait in milliseconds between stopping a container and removing it. This might be helpful in situation where a Docker croaks with an error when trying to remove a container to fast after it has been stopped.

Element	Description
exec	Commands to execute during specified lifecycle of the container. It knows the following sub-elements: • **postStart** Command to run after the above wait criteria has been met • **preStop** Command to run before the container is stopped. • **breakOnError** If set to `true` then break the build if a `postStart` or `preStop` command exits with an return code other than 0, otherwise only print an error message.
tcp	TCP port check which periodically polls given tcp ports. It knows the following sub-elements: • **mode** can be either `mapped` which uses the mapped ports or `direct` in which case the container ports are addressed directly. In the later case the host field should be left empty in order to select the container ip (which must be routed which is only the case when running on the Docker daemon's host directly). Default is `direct` when host is *localhost*, `mapped` otherwise. The direct mode might help when a so called *user-proxy* is enabled on the Docker daemon which makes the mapped ports directly available even when the container is not ready yet. • **host** is the hostname or the IP address. It defaults to `${docker.host.address}` for a mapped mode and the container ip address for the direct mode. • **ports** is a list of TCP ports to check. These are supposed to be the container internal ports.
healthy	Check that waits until the container health state becomes `healthy`. A container is considered healthy when its configured healtcheck succeeds. This behaviour mimics the docker compose dependsOn `condition: service_healthy`.
exit	Check that waits until a container finishes with the given exit code.

As soon as one condition is met the build continues. If you add a `<time>` constraint this works more or less as a timeout for other conditions. The build will abort if you wait on an url or log output and reach the timeout. If only a `<time>` is specified, the build will wait that amount of milliseconds and then continues.

Example

```
<wait>
  <http>
    <url>http://localhost:${host.port}</url>
    <method>GET</method>
    <status>200..399</status>
  </http>
  <time>10000</time>
  <kill>1000</kill>
  <shutdown>500</shutdown>
  <exec>
      <postStart>/opt/init_db.sh</postStart>
      <preStop>/opt/notify_end.sh</preStop>
  </exec>
  <tcp>
      <host>192.168.99.100</host>
      <ports>
          <port>3306</port>
          <port>9999</port>
      </ports>
  </tcp>
  <healthy>true</healthy>
</wait>
```

This setup will wait for the given URL to be reachable but ten seconds at most. Additionally, it will wait for the TCP ports 3306 and 9999. Also, when stopping the container after integration tests, the build wait for 500 ms before it tries to remove the container (if not `keepContainer` or `keepRunning` is used). You can use maven properties in each condition, too. In the example, the `${host.port}` property is probably set before within a port mapping section.

The property `${docker.host.address}` is set implicitly to the address of the Docker host. This host will be taken from the `docker.host` configuration if HTTP or HTTPS is used. If a Unix socket is used for communication with the docker daemon, then `localhost` is assumed. You can override this property always by setting this Maven property explicitly.

5.2.10. Logging

When running containers the standard output and standard error of the container can be printed out. Several options are available for configuring the log output:

Table 19. Logging configuration

Element	Description
enabled	If set to `false` log output is disabled. This is useful if you want to disable log output by default but want to use the other configuration options when log output is switched on on the command line with `-Ddocker.showLogs`. Logging is enabled by default if a `<log>` section is given.

Element	Description
prefix	Prefix to use for the log output in order to identify the container. You can use placeholders in the prefix which are replaced on the fly: • `%a`: Alias of the image, or, if not set, the short container id. • `%c`: Short container id (i.e. the first 6 chars of the container id • `%C`: The full container id • `%n`: The image name • `%z`: An empty string By default the format is "%a> ".
date	Dateformat to use for log timestamps. If `<date>` is not given no timestamp will be shown. The date specification can be either a constant or a date format. The recognized constants are: • `NONE` Switch off timestamp output. Useful on the command line (`-Ddocker.logDate=NONE`) for switching off otherwise enabled logging. • `DEFAULT` A default format in the form `HH:mm:ss.SSS` • `MEDIUM` java.time medium date time format • `SHORT` java.time short date time format • `LONG` java.time long date time format • `ISO8601` Full ISO-8601 formatted date time with milliseconds As an alternative a date-time format string as recognized by java.time is possible. In order to set a consistent date format, the global configuration parameter `logDate` can be used.
color	Color used for coloring the prefix when coloring is enabled (i.e. if running in a console and `useColor` is set). The available colors are `YELLOW`, `CYAN`, `MAGENTA`, `GREEN`, `RED`, `BLUE`. If coloring is enabled and now color is provided a color is picked for you.
file	Path to a file to which the log output is written. This file is overwritten for every run and colors are switched off.
driver	Section which can specify a dedicated log driver to use. A `<name>` tag within this section depicts the logging driver with the options specified in `<opts>`. See the example below for how to use this.

Example

```
<log>
   <prefix>TC</prefix>
   <date>default</date>
   <color>cyan</color>
</log>
```

The following example switches on the `gelf` logging driver . This is equivalent to the options `--log-driver=gelf --log-opt gelf-address=udp://localhost:12201` when using `docker run`.

```
<log>
   ...
   <driver>
      <name>gelf</name>
      <opts>
         <gelf-address>udp://localhost:12201</gelf-address>
      </opts>
   </driver>
</log>
```

5.2.11. Isolation

Specify isolation technology for container

The following configuration option under <run> session is equivalent of `--isolation <value>` when running a docker container

Example

```
<run>
    <isolation>hyperv</isolation>
</run>
```

This option is useful in situations where you are running Docker containers on Windows. The --isolation <value> option sets a container's isolation technology. On Linux, the only supported is the default option which uses Linux namespaces.

Table 20. On Windows, isolation can take one of these values:

Value	Description
default	Use the value specified by the Docker daemon's --exec-opt or system default (see below).
process	Shared-kernel namespace isolation (not supported on Windows client operating systems older than Windows 10 1809).
hyperv	Hyper-V hypervisor partition-based isolation.

The default isolation on Windows server operating systems is process. The default isolation on

Windows client operating systems is hyperv. An attempt to start a container on a client operating system older than Windows 10 1809 with --isolation process will fail.

See isolation technology for container for a detailed description.

5.3. docker:stop

Stops and removes a docker container. This goal stops every container started with `<docker:start>` either during the same build (e.g. when bound to lifecycle phases when doing integration tests) or for containers created by a previous call to `<docker:start>`

If called within the same build run, only the containers that were explicitly started during the run will be stopped. Existing containers started using `docker:start` for the project will not be affected.

If called as a separate invocation, the plugin will stop and remove any container it finds whose image is defined in the project's configuration. Any existing containers found running whose image name matches but were not started by the plugin will not be affected.

In case the naming strategy for an image is `alias` (i.e. the container name is set to the given alias), then only the container with this alias is stopped. Other containers originating from the same image are not touched.

It should be noted that any containers created prior to version `0.13.7` of the plugin may not be stopped correctly by the plugin because the label needed to tie the container to the project may not exist. Should this happen, you will need to use the Docker CLI to clean up the containers and/or use the `docker.allContainers` option listed below.

For tuning what should happen when stopping there are four global parameters which are typically used as system properties: `allContainers`, `keepContainer`, `keepRunning` and `removeVolumes`.

Table 21. Stop configuration

Element	Description	Parameter
allContainers	Stops and removes any container that matches an image defined in the current project's configuration. This was the default behavior of the plugin prior up to version 0.13.6	`docker.allContainers`
containerNamePattern	Default pattern that **docker:start** uses for naming containers when they are created. See Container Names for details. This should match the setting for **docker:start** goals if the goals are configured in separate executions.	`docker.containerNamePattern`
keepContainer	If set to `true` not destroy container after they have been stopped. Default is false.	`docker.keepContainer`
keepRunning	If set to `true` actually don't stop the container. This apparently makes only sense when used on the command line when doing integration testing (i.e. calling `docker:stop` during a lifecycle binding) so that the container are still running after an integration test. This is useful for analysis of the containers (e.g. by entering it with `docker exec`).	`docker.keepRunning`

Element	Description	Parameter
removeVolumes	If set to `true` will remove any volumes associated to the container as well. This option will be ignored if either `keepContainer` or `keepRunning` are true.	`docker.removeVolumes`
stopNamePattern	If a list of name patterns is provided, any containers matching the patterns will be stopped and removed (depending on the values of `keepContainer` and `keepRunning`), independently of whether there is an image configuration.	`docker.stopNamePattern`
executeStopOnVMShutdown	If `true`, the containers are not stopped right away, but when the build is finished (success or failed). Defaults to `false`.	`docker.executeStopOnVMShutdown`

Example

```
$ mvn -Ddocker.keepRunning clean install
```

5.4. docker:push

This goal uploads images to the registry which have a `<build>` configuration section. The images to push can be restricted with the global option `filter` (see Global Configuration for details). The registry to push is by default `docker.io` but can be specified as part of the images's `name` name the Docker way. E.g. `docker.test.org:5000/data:1.5` will push the image `data` with tag `1.5` to the registry `docker.test.org` at port `5000`. Security information (i.e. user and password) can be specified in multiple ways as described in section Authentication.

By default a progress meter is printed out on the console, which is omitted when using Maven in batch mode (option `-B`). A very simplified progress meter is provided when using no color output (i.e. with `-Ddocker.useColor=false`).

Table 22. Push options

Element	Description	Property
skipPush	If set to `true` the plugin won't push any images that have been built.	`docker.skip.push`
skipTag	If set to `true` this plugin won't push any tags	`docker.skip.tag`
pushRegistry	The registry to use when pushing the image. See Registry Handling for more details.	`docker.push.registry`
retries	How often should a push be retried before giving up. This useful for flaky registries which tend to return 500 error codes from time to time. The default is 0 which means no retry at all.	`docker.push.retries`

5.5. docker:watch

When developing and testing applications you will often have to rebuild Docker images and restart containers. Typing `docker:build` and `docker:start` all the time is cumbersome. With `docker:watch`

you can enable automatic rebuilding of images and restarting of containers in case of updates.

`docker:watch` is the top-level goal which performs these tasks. There are two watch modes, which can be specified in multiple ways:

- `build` : Automatically rebuild one or more Docker images when one of the files selected by an assembly changes. This works for all files included directly in `assembly.xml` but also for arbitrary dependencies.

Example

```
$ mvn package docker:build docker:watch -Ddocker.watchMode=build
```

This mode works only when there is a `<build>` section in an image configuration. Otherwise no automatically build will be triggered for an image with only a `<run>` section. Note that you need the `package` phase to be executed before otherwise any artifact created by this build can not be included into the assembly. As described in the section about `docker:start` this is a Maven limitation. * `run` : Automatically restart container when their associated images changes. This is useful if you pull a new version of an image externally or especially in combination with the `build` mode to restart containers when their image has been automatically rebuilt. This mode works reliably only when used together with `docker:start`.

Example

```
$ mvn docker:start docker:watch -Ddocker.watchMode=run
```

- `both` : Enables both `build` and `run`. This is the default.
- `none` : Image is completely ignored for watching.
- `copy` : Copy changed files into the running container. This is the fast way to update a container, however the target container must support hot deploy, too so that it makes sense. Most application servers like Tomcat supports this.

The mode can also be `both` or `none` to select both or none of these variants, respectively. The default is `both`.

`docker:watch` will run forever until it is interrupted with `CTRL-C` after which it will stop all containers. Depending on the configuration parameters `keepContainer` and `removeVolumes` the stopped containers with their volumes will be removed, too.

When an image is removed while watching it, error messages will be printed out periodically. So don't do that ;-)

Dynamically assigned ports stay stable in that they won't change after a container has been stopped and a new container is created and started. The new container will try to allocate the same ports as the previous container.

If containers are linked together network or volume wise, and you update a container which other containers dependent on, the dependant containers are not restarted for now. E.g. when you have a

"service" container accessing a "db" container and the "db" container is updated, then you "service" container will fail until it is restarted, too.

> A future version of this plugin will take care of restarting these containers, too (in the right order), but for now you would have to do this manually.

This maven goal can be configured with the following top-level parameters:

Table 23. Watch configuration

Element	Description	Property
containerNamePattern	Default pattern for naming all containers when they are created. See Container Names for details.	`docker.containerNamePattern`
keepContainer	As for `docker:stop`, if this is set to `true` (and `keepRunning` is disabled) then all container will be removed after they have been stopped. The default is `true`.	`docker.keepContainer`
keepRunning	If set to `true` all container will be kept running after `docker:watch` has been stopped. By default this is set to `false`.	`docker.keepRunning`
removeVolumes	if set to `true` will remove any volumes associated to the container as well. This option will be ignored if either `keepContainer` or `keepRunning` are `true`.	`docker.removeVolumes`
watchInterval	Interval in milliseconds how often to check for changes, which must be larger than 100ms. The default is 5 seconds.	`docker.watchInterval`
watchMode	Watch mode specifies what should be watched • `build` : Watch changes in the assembly and rebuild the image in case • `run` : Watch a container's image whether it changes and restart the container in case • `copy` : Changed files are copied into the container. The container can be either running or might be already exited (when used as a *data container* linked into a *platform container*). Requires Docker >= 1.8. • `both` : `build` and `run` combined • `none` : Neither watching for builds nor images. This is useful if you use prefactored images which won't be changed and hence don't need any watching. `none` is best used on an per image level, see below how this can be specified.	`docker.watchMode`
watchPostExec	A command which is executed within the container after files are copied into this container when `watchMode` is `copy`. Note that this container must be running.	

Element	Description	Property
watchPostG oal	A maven goal which should be called if a rebuild or a restart has been performed. This goal must have the format `<pluginGroupId>:<pluginArtifactId>:<goal>` and the plugin must be configured in the `pom.xml`. For example a post-goal `io.fabric8:fabric8:delete-pods` will trigger the deletion of PODs in Kubernetes which in turn triggers are new start of a POD within the Kubernetes cluster. The value specified here is the the default post goal which can be overridden by `<postGoal>` in a `<watch>` configuration.	

Image specific watch configuration goes into an extra image-level `<watch>` section (i.e. `<image><watch>...</watch></image>`). The following parameters are recognized:

Table 24. Watch configuration for a single image

Element	Description
mode	Each image can be configured for having individual watch mode. These take precedence of the global watch mode. The mode specified in this configuration takes precedence over the globally specified mode.
interval	Watch interval can be specified in milliseconds on image level. If given this will override the global watch interval.
postGoal	Post Maven plugin goal after a rebuild or restart. The value here must have the format `<pluginGroupId>:<pluginArtifactId>:<goal>` (e.g. `io.fabric8:fabric8:delete-pods`)
postExec	Command to execute after files are copied into a running container when `mode` is `copy`.

Here is an example how the watch mode can be tuned:

Example

```
<configuration>
    <!-- Check every 10 seconds by default -->
    <watchInterval>10000</watchInterval>
    <!-- Watch for doing rebuilds and restarts -->
    <watchMode>both</watch>
    <images>
        <image>
            <!-- Service checks every 5 seconds -->
            <alias>service</alias>
            ....
            <watch>
                <interval>5000</interval>
            </watch>
        </image>
        <image>
            <!-- Database needs no watching -->
            <alias>db<alias>
            ....
            <watch>
                <mode>none</mode>
            </watch>
        </image>
        ....
    </images>
</configuration>
```

Given this configuration

Example

```
mvn package docker:build docker:start docker:watch
```

You can build the service image, start up all containers and go into a watch loop. Again, you need the `package` phase in order that the assembly can find the artifact build by this project. This is a Maven limitation. The `db` image will never be watch since it assumed to not change while watching.

5.6. docker:remove

This goal can be used to clean up images. By default all images with a build configuration are removed. You can tune this by setting the property `removeMode` (property: `docker.removeMode`) to one of the following values:

Table 25. removeMode Values

Value	Description
build	All images with a build configuration
run	All images without a build configuration

Value	Description
`all`	All configured images
`data`	All data images, which are images without a [run configuration]().

Previously, this could be tuned also by providing the property `removeAll` which indicates to remove all images managed by this build. Otherwise only data images were delete before 0.24.0. `removeAll` is deprecated and will be removed soone. Please use `removeMode` instead.

As with the other goals, the configuration `image` can be used to tune the images to remove. All containers belonging to the images are removed as well as the all tags assigned to this image

Considering three images 'db','tomcat' and 'data' where 'data' is the only image with a build configuration:

- `mvn docker:remove` will remove 'data'

- `mvn -Ddocker.removeMode=all docker:remove` will remove all three images

- `mvn -Ddocker.filter=data,tomcat docker:remove` will remove 'data'

- `mvn -Ddocker.filter=data,tomcat -Ddocker.removeMode=all docker:remove` will remove 'data' and 'tomcat'

Table 26. Remove options

Element	Description	Property
skipTag	If set to `true` this plugin won't remove any tags	`docker.skip.tag`
removeNamePattern	If a list of [name patterns]() is provided, any images matching the patterns will be removed, independently of whether there is an [image configuration]() marked for removal.	`docker.removeNamePattern`

5.7. docker:logs

With this goal it is possible to print out the logs of containers started from images configured in this plugin. By default only the latest container started is printed, but this can be changed with a property. The format of the log output is influenced by run configuration of the configured images. The following system properties can the behaviour of this goal:

Table 27. Logging options

Property	Description
docker.logAll	If set to `true` the logs of all containers created from images configured for this plugin are printed. The container id is then prefixed before every log line. These images can contain many containers which are already stopped. It is probably a better idea to use `docker logs` diretly from the command line.
docker.follow	If given will wait for subsequent log output until CRTL-C is pressed. This is similar to the behaviour of `docker logs -f` (or `tail -f`).
docker.filter	Filter to restrict the set of images for which log should be fetched. This can be a comma separated list of image or alias names.

Property	Description
docker.logDate	Date format to use. See "Logging" for available formats.

Example

```
$ mvn docker:logs -Ddocker.follow -Ddocker.logDate=DEFAULT
```

5.8. docker:copy

This goal copies files and directories from a container. When called, then all images which are configured in the project and having copy element in the image configuration are iterated.

In addition to the Global configuration, this goal supports the following configuration options:

Table 28. Copy options

Element	Description	Property

createContainers	Whether to create temporary containers or to copy from existing containers.
	
	 If `true` then a temporary container is created (but not started) before the copying and is removed after completion of the copying, even if the copying failed. Container image is pulled from registry following imagePullPolicy.
	
	 If `false` then copying from existing containers:
	
	 • If the goal is called together with **docker:start** goal, then the copying is performed only from containers started by that goal.
	
	 • Otherwise, the copying is performed from containers matching configured image and copyNamePattern property of image configuration is examined:
	
	     ◦ If `copyNamePattern` pattern is defined, then it is used to match containers.
	
	     ◦ Otherwise, image name is used to match containers, i.e. containers created from the image with the same name are examined.
	
	 Temporary containers are created and removed or existing containers are examined in the order of image configurations.
	
	 Defaults to `false`.
		`docker.createContainers`

pullRegistry	The registry used for pulling image when creating temporary container and imagePullPolicy allows or requires pulling respective image. Ignored if `createContainers` is `false`.	`docker.pull.registry`

Element	Description	Property
containerNamePattern	Naming pattern for how to name containers when created. Ignored if `createContainers` is `false`. Defaults to default container naming pattern (`%n-%i`).	`docker.containerNamePattern`
copyAll	Whether to copy from all matching containers or only from the newest ones. Ignored if `createContainers` is `true`. Defaults to `false`.	`docker.copyAll`

The `copy` image configuration element is honored by the goal and has the following sub elements:

Table 29. Copy configuration (Image Configuration)

Element	Description
entries	List of items to copy from a container. Each item is wrapped with `<entry>` and `</entry>` tags. Refer to Copy entry format for the format of a single list item. Copying is performed in the order of list items. Optional, i.e. can be omitted or can have an empty list of items.

Each item in the `entries` list of `copy` element consists of the following sub elements:

Table 30. Copy entry format

Element	Description
containerPath	Path to a container file or a container directory, which needs to be copied. If the path is not absolute, then it is considered relative to the container working directory.
hostDirectory	Path to a host directory where a copied file or a copied directory needs to be placed. "host" means the machine where the `copy` goal is executed, i.e. where respective maven project is built. If the path is not absolute, then it is considered relative to the maven project base directory. If a container directory is copied, then a directory with the same name is created under the path defined by `hostDirectory`, i.e. a whole container directory is copied but not just its content. Optional. If omitted then project base directory is used.

Example of copy configuration

```xml
<plugin>
  <artifactId>docker-maven-plugin</artifactId>
  <!-- ..... -->
  <configuration>
    <images>
      <image>
        <!-- ..... -->
        <copy>
          <entries>
            <!-- Copy this file first -->
            <entry>
              <!-- The container file to copy -->
              <containerPath>/etc/hosts</containerPath>
              <!--
                  Absolute path.
                  ${project.build.directory}/hosts file will be created as a result
of copying
                  and the content of container /etc/hosts file will be placed there.
               -->
              <hostDirectory>${project.build.directory}</hostDirectory>
            </entry>
            <!-- Copy this directory second -->
            <entry>
              <!-- The container directory to copy -->
              <containerPath>/dev</containerPath>
              <!--
                  Relative bath based on project.basedir.
                  ${project.basedir}/target/dev directory will be created as a result
of copying
                  and the content of container /dev directory will be placed there.
               -->
              <hostDirectory>target</hostDirectory>
            </entry>
          </entries>
        </copy>
      </image>
    </images>
  </configuration>
</plugin>
```

5.9. docker:source

The `docker:source` target can be used to attach a docker build archive containing the Dockerfile and all added files to the Maven project with a certain classifier. It reuses the configuration from **docker:build**.

By default, only the first image configuration is used for creating the source archive. You can export

all image configurations by setting the `sourceMode` configuration to `all`:

Export all image configs

```xml
<plugin>
  <artifactId>docker-maven-plugin</artifactId>
  <configuration>
    <!-- source mode can be "first" or "all" -->
    <sourceMode>all</sourceMode>
    <!-- .... -->
  </configuration>
</plugin>
```

For exporting all image configurations, `docker:source` uses the image's alias as part of the classifier, so it is mandatory that the alias is set for this goal to work when all images should be exported this way. The classifier is calculated as `docker-<alias>` so when the alias is set to `service`, then the classifier is `docker-service`.

If you only export the first image configuration (which is the default), then the classifier is just `docker` (without alias).

`docker:source` can be attached to a Maven execution phase, which is `generate-sources` by default.

For example, this configuration will attach the docker build archive to the artifacts to store in the repository:

Example

```xml
<plugin>
  <artifactId>docker-maven-plugin</artifactId>
  <!-- ..... -->
  <executions>
    <execution>
      <id>sources</id>
      <goals>
        <goal>source</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

If not bound to an execution phase, `docker:source` requires that the artifact has been created so you call it best together with `package`

5.10. docker:save

The `docker:save` target saves an image defined in the build configuration to a local file, analogous to `docker save`. If the option `saveFile` is not set, the file name is calculated automatically:

- If `saveAlias` is used then the file is stored as `target/<alias>-<project version>.tar.gz`

- Otherwise the archive is stored as `target/<image name without registry and user part>-<image tag>.tar.gz`

Please note that the exported image contains all image layers and can be quite large (also, it takes a bit to export the image).

Controlling image compression

The file name extension is used to select a compression method for the output.

Extensions	Compression	Type
.tar or unrecognized	No compression	.tar
.tar.gz, .tgz	GZIP compression	.tar.gz
.tar.bz, .tar.bz2, .tar.bzip2	BZIP2 compression	.tar.bz

Attaching the saved image as an artifact

If `saveClassifier` is set, the saved archive will be attached to the project using the provided classifier and the type determined from the file name. The placeholder `%a` will be replaced with the image alias.

Note that using overriding the default to use `docker` or `docker-%a` may lead to a conflict if a source archive is also attached with **docker:source**.

Table 31. Save options

Element	Description	Property
saveName	The name of the image configuration to save. Must not be used together with `alias`.	`docker.save.name`
saveAlias	The alias of the image configuration to save. Must not be used together with `name`.	`docker.save.alias`
saveFile	The filename to save.	`docker.save.file` or `docker.file` or `file`
saveClassifier	If set, attach the the saved archive to the project with the provided classifier. A placeholder of `%a` will be replaced with the image alias.	`docker.save.classifier`
skipSave	A boolean flag whether to skip execution of the goal.	`docker.skip.save`

5.11. docker:tag

The `docker:tag` tags an image so that it becomes part of a repository. You can use it to tag an already built image. Here is an example of it's usage:

```
~/work/repos/docker-maven-plugin/samples/zero-config : $ mvn docker:tag
-Ddocker.image.tag=0.9.0
[INFO] Scanning for projects...
[INFO]
[INFO] ----------< io.fabric8.dmp.samples:demp-sample-zero-config >----------
[INFO] Building demp-sample-zero-config
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- docker-maven-plugin:dmpversion:tag (default-cli) @ demp-sample-zero-config
---
[INFO] DOCKER> Tagging image samples/demp-sample-zero-config:0.9.0 successful!
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.155 s
[INFO] Finished at: 2020-06-27T14:05:33+05:30
[INFO] ------------------------------------------------------------------------
~/work/repos/docker-maven-plugin/samples/zero-config : $ docker images | grep 0.9.0
samples/demp-sample-zero-config                      0.9.0
ac5c5991505d        About an hour ago    479MB
```

Table 32. Supported options

Element	Description	Property
tagName	The name of the new tag.	`docker.image.tag`
repo	The repository to tag in. For example, `someuser/someimage`.	`docker.image.repo`

5.12. docker:volume-create

This goals creates one or more standalone Docker volume, which can be referenced in a **docker:start** configuration for linking to a volume during runtime. Each volume has therefore a unique and referenceable name. Beside the volume driver and driver options can be specified.

Example for a volume configuration

```
<plugin>
  <configuration>
    <volumes>
      <volume>
        <name>temp-volume</name>
        <driver>local</driver>
        <opts>
          <type>tmpfs</type>
          <device>tmpfs</device>
          <o>size=100m,uid=1000</o>
        </opts>
        <labels>
          <volatileData>true</volatileData>
        </labels>
      </volume>
    </volumes>
    ...
  </configuration>
</plugin>
```

Configuration

The following options are available when creating volumes:

Table 33. Volume configuration

Element	Description
name	Name of the volume
driver	Volume driver to use. By default the driver `local` is used which is created on the local file system. Please refer to your Docker installation which additional drivers are available.
opts	Driver specific options passed in as custom `<key>value</key>` where its maps to `key=value` pairs for driver options as they can be provided from the Docker CLI, too. Each volume driver supports different options. The options supported by the `local` driver are the well known Linux mount options.
labels	Labels given as `<key>value</key>` similar to image labels described in Environment and Labels. These labels are used to tag the volumes themselves.

5.13. docker:volume-remove

This goals is the counterpart to **docker:volume-create** and removes a volume. Docker volumes are configured outside of Docker images, but can be referenced by them. The configuration is the same

as for **docker:volume-create**

Example:

```xml
<plugin>
  <configuration>
    <volumes>
      <volume>
        <name>temp-volume</name>
        ....
      </volume>
    </volumes>
    ...
  </configuration>
</plugin>
```

Configuration

The configuration is quite simple. Only the name of the volume to delete is required.

Table 34. Volume configuration

Element	Description
name	Name of the volume

Chapter 6. External Configuration

For special configuration needs, there is the possibility to get the runtime and build configuration from places outside the plugin's configuration. This is done with the help of `<external>` configuration sections which at least has a `<type>` subelement. This `<type>` element selects a specific so called "handler" which is responsible for creating the full image configuration. A handler can decide to use the `<run>` and `<build>` configuration which could be provided in addition to this `<external>` section or it can decide to completely ignore any extra configuration option.

A handler can also decide to expand this single image configuration to a list of image configurations. The image configurations resulting from such a external configuration are added to the *regular* `<image>` configurations without an `<external>` section.

The available handlers are described in the following.

6.1. Properties

For simple needs the image configuration can be completely defined via Maven properties which are defined outside of this plugin's configuration. Such a property based configuration can be selected with an `<type>` of `properties`. As extra configuration a prefix for the properties can be defined which by default is `docker`.

For single-image configurations it is also possible to active property based configuration via an externally set property.

By default, property based configuration uses only properties, ignoring any `<build>` and `<run>` sections. To combine values from both sources, use the property mode configuration.

Properties are read from the Maven project (defined in `<properties>` or global Maven configuration from `settings.xml`) and, since 0.25.0, from any `-D` flags given to Maven (takes priority over project properties).

Example

```xml
<image>
  <external>
    <type>properties</type>
    <prefix>docker</prefix> <!-- this is the default -->
    <mode>only</mode> <!-- this is the default -->
  </external>
</image>
```

Given this example configuration a single image configuration is built up from the following properties, which correspond to the corresponding values in the `<build>` and `<run>` sections. A build configuration is only created when a `docker.from` or a `docker.fromExt` is set.

Table 35. External properties

docker.alias	Alias name

docker.args.B UILDVAR	Set the value of a build variable. The syntax is the same as for specifying environment variables (see below).
docker.assem bly.baseDir	Directory name for the exported artifacts as described in an assembly (which is `/maven` by default).
docker.assem bly.descriptor	Path to the assembly descriptor when building an image
docker.assem bly.descriptor Ref	Name of a predefined assembly to use.
docker.assem bly.exportBas eDir	If `true` export base directory
docker.assem bly.ignorePer missions	If set to `true` existing file permissions are ignored when creating the assembly archive. Deprecated, use a permission mode of `ignore` instead.
docker.assem bly.permissio ns	can be `ignore` to use the permission as found on files regardless on any assembly configuration, `keep` to respect the assembly provided permissions, `exec` for setting the executable bit on all files (required for Windows when using an assembly mode `dir`) or `auto` to let the plugin select `exec` on Windows and `keep` on others. `keep` is the default value.
docker.assem bly.dockerFile Dir	specifies a directory containing an external Dockerfile that will be used to create the image. This is deprecated please use `docker.dockerFileDir` or `docker.dockerFile` instead.
docker.noCac he	Don't use Docker's build cache. This can be overwritten by setting a system property `docker.noCache` when running Maven.
docker.bind.id x	Sets a list of paths to bind/expose in the container. See List Properties.
docker.build.n etwork	Set the networking mode for the RUN instructions during build
docker.buildA rg.VARIABLE	Set a ARG to be available during build of image. **Note**: this is handled separately from external configuration, and is always available. See Build Args for more details.
docker.buildx. builderName	Name of buildx builder
docker.buildx. cache	Location of image cache
docker.buildx. configFile	Configuration file for buildx builder
docker.buildx. platform	Platform for multi-architecture image
docker.capAd d.idx	List of kernel capabilities to add to the container. See List Properties.
docker.capDr op.idx	List of kernel capabilities to remove from the container. See List Properties.

docker.sysctls.KEY	Sets a namespaced kernel parameter (sysctl) in the container.
docker.cleanup	Cleanup dangling (untagged) images after each build, including any stopped containers created from them. Also cleanup dangling images as a result of image tagging, auto-pulling a base image, or auto-pulling a cacheFrom image. Default is `try`, which tries to remove the old image, but doesn't fail the build if this is not possible (e.g. because the image is still used by a running container). Other possible values are `remove`, if you want to fail the build, or `none`, to skip cleanup altogether.
docker.cmd	Command to execute. This is used both when running a container and as default command when creating an image.
docker.copy.entries.idx.containerPath	`containerPath` sub element of copy configuration entry. `idx` can be an integer number, or a string. Empty string is allowed. `idx` defines order of copying. Entries which `idx` is an integer number are copied first in ascending order by the parsed value of `idx`. Entries which `idx` is not an integer number are copied second in ascending alphabetical order by `idx`.
docker.copy.entries.idx.hostDirectory	`hostDirectory` sub element of copy configuration entry. `idx` should match the one from `docker.copy.entries.idx.containerPath` external property.
docker.copyNamePattern	Set copyNamePattern of image configuration.
docker.cpus	Specify how much of the available CPU resources a container can use
docker.cpuset	Limit the container to specific CPUs or cores. This can be provided either as a comma-separated list or a hyphen-separated range.
docker.cpushares	Set the proportion of the host machine's cpu cycles available to the container
docker.cacheFrom.idx	Defines a list of image names to use as cache sources. See List Properties.
docker.domainname	Container domain name
docker.dns.idx	List of dns servers to use. See List Properties.
docker.dnsSearch.idx	List of dns search domains. See List Properties.
docker.dockerArchive	specify an archive which can be loaded with `docker load`. Use this as an alternative to `docker.dockerFile` or `docker.dockerFileDir`
docker.dockerFile	specifies a Dockerfile to use. This property must point to the Dockerfile itself.
docker.dockerFileDir	specifies a directory containing an external dockerfile that will be used to create the image. The dockerfile must be name `Dockerfile`
docker.entrypoint	Container entry point
docker.exposedPropertyKey	Property part for the exposed container properties like internal IP addresses as described in docker:start.

docker.env.VARIABLE	Sets an environment variable used in build and run. E.g. `<docker.env.JAVA_OPTS>-Xmx512m</docker.env.JAVA_OPTS>` sets the environment variable `JAVA_OPTS`. Multiple such entries can be provided. This environment is used both for building images and running containers. The value cannot be empty but can contain Maven property names which are resolved before the Dockerfile is created.
docker.envBuild.VARIABLE	Sets an environment variable used in build only. E.g. `<docker.envBuild.JAVA_OPTS>-Xmx512m</docker.envBuild.JAVA_OPTS>` sets the environment variable `JAVA_OPTS`. Multiple such entries can be provided. This environment is building images only. The value cannot be empty but can contain Maven property names which are resolved before the Dockerfile is created.
docker.envRun.VARIABLE	Sets an environment variable used in run only. E.g. `<docker.envRun.JAVA_OPTS>-Xmx512m</docker.envRun.JAVA_OPTS>` sets the environment variable `JAVA_OPTS`. Multiple such entries can be provided. This environment is used both for running containers only. The value cannot be empty but can contain Maven property names which are resolved before the Dockerfile is created.
docker.envPropertyFile	specifies the path to a property file whose properties are used as environment variables in run. The environment variables takes precedence over any other environment variables specified.
docker.extraHosts.idx	List of `host:ip` to add to `/etc/hosts`. See List Properties.
docker.filter	Enable and set the delimiters for property replacements. By default properties in the format `${..}` are replaced with Maven properties. You can switch off property replacement by setting this property to `false`. When using a single char like `@` then this is used as a delimiter (e.g `@···@`). See Filtering for more details.
docker.from	Base image for building an image. Must be set when an image is created (or `fromExt`)
docker.fromExt.VARIABLE	Base image for building an image (extended format), which also triggers a build of an image.
docker.healthcheck.cmd	Command to use for a healthcheck
docker.healthcheck.interval	Interval for how often to run a healthcheck (in seconds or with a given time unit)
docker.healthcheck.mode	If se to `none` disable a healthcheck from a base image
docker.healthcheck.retries	Number of retries for how often to retry a healthcheck until it is considered to have failed
docker.healthcheck.startPeriod	Initialization time for containers that need time to bootstrap. Probe failure during that period will not be counted towards the maximum number of retries. However, if a health check succeeds during the start period, the container is considered started and all consecutive failures will be counted towards the maximum number of retries. (in seconds or with a given time unit)
docker.healthcheck.timeout	Timeout after which a healthcheck command is considered to be failed (in seconds or with a given time unit)
docker.hostname	Container hostname

docker.image PropertyConfiguration	Special property to activate property configuration without altering XML file (see Activating property configuration externally).
docker.image PullPolicy.build	Specific pull policy used when building images. See imagePullPolicy for the possible values.
docker.image PullPolicy.run	Specific pull policy used for downloading images to run. See imagePullPolicy for the possible values.
docker.labels. LABEL	Sets a label which works similarly like setting environment variables.
docker.loadNamePattern	Search the archive specified in `docker.dockerArchive` for the specified image name and creates a tag from the matched name to the build image name specified in `docker.name`.
docker.log.enabled	Use logging (default: `true`)
docker.log.prefix	Output prefix
docker.log.color	ANSI color to use for the prefix
docker.log.date	Date format for printing the timestamp
docker.log.driver.name	Name of an alternative log driver
docker.log.driver.opts.VARIABLE	Logging driver options (specified similarly as in `docker.env.VARIABLE`)
docker.links.idx	defines a list of links to other containers when starting a container. For example `<docker.links.1>db</docker.links.1>` specifies a link to the image with alias 'db'. See List Properties.
docker.maintainer	defines the maintainer's email as used when building an image
docker.memory	Memory limit in bytes.
docker.memorySwap	Total memory limit (memory + swap) in bytes. Set `docker.memorySwap` equal to `docker.memory` to disable swap. Set to `-1` to allow unlimited swap.
docker.name	Image name
docker.namingStrategy	Container naming (either `none` or `alias`)
docker.network.mode	Network mode to use which can be `none`, `host`, `bridged`, `container` or `custom`
docker.network.name	Name of the custom network when mode is `custom`, or for mode `container` the image alias name used to create the container.
docker.network.alias.idx	One or more aliase for a custom network. Only used when the network mode is `custom`. See List Properties.

docker.noCache	Don't use a cache when building the image
docker.squash	Squash newly built layers into a single layer (API 1.25+, need to be enabled in the Docker daemon configuration)
docker.optimise	if set to true then it will compress all the `runCmds` into a single RUN directive so that only one image layer is created.
docker.portPropertyFile	specifies a path to a port mapping used when starting a container.
docker.ports.idx	Sets a port mapping. For example `<docker.ports.1>jolokia.ports:8080<docker.ports.1>` maps the container port 8080 dynamically to a host port and assigns this host port to the Maven property `${jolokia.port}`. See Port mapping for possible mapping options. When creating images images only the right most port is used for exposing the port. For providing multiple port mappings, the index should be count up. See List Properties for more information about list properties.
docker.registry	Registry to use for pushing images.
docker.restartPolicy.name	Container restart policy
docker.restartPolicy.retry	Max restart retries if `on-failure` used
docker.run.idx	List of commands to `RUN` when creating the image. See List Properties.
docker.securityOpts.idx	List of `opt` elements to specify kernel security options to add to the container. For example `docker.securityOpt.1=seccomp=unconfined`. See List Properties.
docker.shmsize	Size of `/dev/shm` in bytes.
docker.tags.idx	List of tags to apply to a built image. See List Properties.
docker.tmpfs.idx	One or more mount points for a **tmpfs**. Add mount options after a `:`. See List Properties.
docker.ulimits.idx	Ulimits for the container. Ulimit is specified with a soft and hard limit `<type>=<soft limit>[:<hard limit>]`. For example `docker.ulimits.1=memlock=-1:-1`. See List Properties.
docker.user	User to switch to at the end of a Dockerfile. Not to confuse with `docker.username` which is used for authentication when interacting with a Docker registry.
docker.volumes.idx	defines a list of volumes to expose when building an image. See List Properties.
docker.volumesFrom.idx	defines a list of image aliases from which the volumes should be mounted of the container. For examples `<docker.volumesFrom.1>data</docker.volumesFrom.1>` will mount all volumes exported by the `data` image. See List Properties.
docker.wait.http.url	URL to wait for during startup of a container
docker.wait.http.method	HTTP method to use for ping check

docker.wait.http.status	Status code to wait for when doing HTTP ping check
docker.wait.time	Amount of time to wait during startup of a container (in ms)
docker.wait.log	Wait for a log output to appear.
docker.wait.exec.postStart	Command to execute after the container has start up.
docker.wait.exec.preStop	Command to execute before command stops.
docker.wait.exec.breakOnError	If set to "true" then stop the build if the a `postStart` or `preStop` command failed
docker.wait.shutdown	Time in milliseconds to wait between stopping a container and removing it.
docker.wait.tcp.mode	Either `mapped` or `direct` when waiting on TCP connections
docker.wait.tcp.host	Hostname to use for a TCP wait checks
docker.wait.tcp.port.idx	List of ports to use for a TCP check. See List Properties.
docker.wait.kill	Time in milliseconds to wait between sending SIGTERM and SIGKILL to a container when stopping it.
docker.workdir	Container working directory where the image is build in
docker.workingDir	Current Working dir for commands to run in when running containers

Multiple property configuration handlers can be used if they use different prefixes. As stated above the environment and ports configuration are both used for running container and building images. If you need a separate configuration you should use explicit run and build configuration sections.

List Properties

List properties refer to XML configurations items that accept a list of values, like `<build><tag>` or `<run><ports>`. To specify values using properties, you must declare a property for each value you want to add to the list, and add a *idx* suffix to the property name to determine its position in the resulting list. For example:

```
<docker.ports.1>80<docker.ports.1>
<docker.ports.2>8080<docker.ports.2>

<docker.tags.jenkins>${BUILD_TIMESTAMP}</docker.tags.jenkins>
<docker.tags.current>latest</docker.tags.current>
```

The *idx* suffix defines the order of copying. Entries which *idx* is an integer number are copied first

in ascending order by the parsed value of *idx*. Entries which *idx* is not an integer number are copied second in ascending alphabetical order by *idx*.

Combining property and XML configuration

By default the property handler will only consider properties and ignore any other image configuration in the XML/POM file. This can be changed by adding the `<mode>` configuration (since version 0.25.0), which can have one of the following values:

Table 36. Property mode

`only`	Only look at properties, ignore any `<run>` or `<build>` sections for this image. This is the default, and also the behavior in versions before 0.25.0.
`override`	Use property if set, else fall back to value found in `<run>` or `<build>` sections for this image.
`fallback`	Use value found in `<run>` or `<build>` sections for this image, else fall back to to property value.
`skip`	Effectively disable properties, same as not specifying the `<external>` section at all.

Activating property configuration externally

It also possible to activate property configuration by setting the property `docker.imagePropertyConfiguration` to a valid `property mode`, without adding an `<external>` section. The plugin will then use any properties with default `docker.` prefix. This can be useful if most of the configuration is specified in XML/POM file, but there is need to override certain configuration values without altering the POM file (instead add this to a parent POM or global settings.xml).

If set in parent POM, but not wanted in specific project, the property could be overriden locally with the value `skip` to disabled property configuration for that particular project. If set in settings.xml however, by Maven design, that value will always take precedence over any properties defined in pom.xml.

For configurations with multiple images, using this property will by default produce an error. All images would then use the same `docker` property prefix, resulting in multiple identical configurations. This can be overruled by adding an explicit <external> configuration element with an explicit <prefix> to all images (or at least all but one). Normally you'd want to use different prefix for each image, but if explicitly set it does allow you to use the same prefix (even `docker`) on all images. This is useful in case you just want to share a few properties. This only makes sense when `property mode` is *override* or *fallback* and image-specific configuration are defined in the POM configuration.

For examples, see [here](#)

Merging POM and property values

For some fields it may be desired to merge values from both POM and properties. For example, in a certain run environment we might want to inject a `http_proxy` environment variable, but we do not want to add this to the POM file.

This is solved using a **Combine policy** which can be either `replace` or `merge`. Merge is only available for configuration of Map or List type. For scalar values such as strings and integers, it is not supported. For Maps, both sources are merged, with the priority source taking precedence. For

Lists, they are concatenated, with values from the priority source being added first.

Combine policy is specified per configuration key/property, and the default in most cases is currently `replace`. The following keys have `merge` as default policy:

- `docker.args`
- `docker.envBuild`
- `docker.envRun`
- `docker.labels`
- `docker.ports`
- `docker.tags`

This can be overridden individually for all configuration keys (of map/list type) by setting an additional property suffixed `._combine`. For example, to not merge ports, set `docker.ports._combine=replace`, and to enable merging of dns, set `docker.dns._combine=merge`.

Example, properties only

```
<properties>
  <docker.name>jolokia/demo</docker.name>
  <docker.alias>service</docker.alias>
  <docker.from>consol/tomcat:7.0</docker.from>
  <docker.assembly.descriptor>src/main/docker-
assembly.xml</docker.assembly.descriptor>
  <docker.env.CATALINA_OPTS>-Xmx32m</docker.env.CATALINA_OPTS>
  <docker.label.version>${project.version}</docker.label.version>
  <docker.ports.jolokia.port>8080</docker.ports.jolokia.port>
  <docker.wait.url>http://localhost:${jolokia.port}/jolokia</docker.wait.url>
</properties>

<build>
  <plugins>
    <plugin>
      <groupId>io.fabric8</groupId>
      <artifactId>docker-maven-plugin</artifactId>
      <configuration>
        <images>
          <image>
            <external>
              <type>properties</type>
              <prefix>docker</prefix>
            </external>
          </image>
        </images>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Example, combining properties and XML/POM configuration

```xml
<properties>
  <docker.assembly.descriptor>src/main/docker-
assembly.xml</docker.assembly.descriptor>
  <docker.env.CATALINA_OPTS>-Xmx32m</docker.env.CATALINA_OPTS>
  <docker.label.version>${project.version}</docker.label.version>
  <docker.ports.jolokia.port>8080</docker.ports.jolokia.port>
  <docker.wait.url>http://localhost:${jolokia.port}/jolokia</docker.wait.url>
</properties>

<build>
  <plugins>
    <plugin>
      <groupId>io.fabric8</groupId>
      <artifactId>docker-maven-plugin</artifactId>
      <configuration>
        <images>
          <image>
            <external>
              <type>properties</type>
              <prefix>docker</prefix>
              <mode>override</mode>
            </external>

            <name>jolokia/demo</name>
            <alias>service</alias>

            <build>
              <from>consol/tomcat:7.0</from>
              <labels>
                <software>tomcat</software>
              </labels>
            </build>
          </image>
        </images>
      </configuration>
    </plugin>
  </plugins>
</build>
```

This would build the same image as the previous example. If instead built with `mvn docker:build -Pdocker.from=console/tomcat:8.0 -Ddocker.tags.0=tc8-test` it would build from that image instead, and also add that tag to the image.

If `-Ddocker.labels.status=beta` is added, the image would be given two labels: `status=beta` and `software=tomcat`. If `-Ddocker.labels._combine=replace` is added, the image would be given one label only: `status=beta`.

Example, external activation of property configuration, single image

Global ~/.m2/settings.xml file:

```xml
<profiles>
  <profile>
    <id>http-proxy</id>
    <properties>
      <docker.buildArg.http_proxy>
http://proxy.example.com:8080</docker.buildArg.http_proxy>
      <docker.runArg.http_proxy>
http://proxy.example.com:8080</docker.runArg.http_proxy>
      <docker.imagePropertyConfiguration>override</docker.imagePropertyConfiguration>
    </properties>
  </profile>
</profiles>
```

```xml
<build>
  <plugins>
    <plugin>
      <groupId>io.fabric8</groupId>
      <artifactId>docker-maven-plugin</artifactId>
      <configuration>
        <images>
          <image>
            <name>jolokia/demo</name>
            <alias>service</alias>
            <build>
              <from>consol/tomcat:7.0</from>
            </build>
          </image>
        </images>
      </configuration>
    </plugin>
  </plugins>
</build>
```

When the plugin is executed, on a machine with the given settings.xml, the plugin will see the `docker.imagePropertyConfiguration` configuration and enable the property merging feature. When building, it will inject the http_proxy build ARG, and when running, it will inject the http_proxy ENV variable. The rest of the configuration will be sourced from the XML, unless the Maven project has any other `docker.*` properties defined.

Example, external activation of property configuration, two images

Using the same global ~/.m2/settings.xml file as in previous example, but with two image definitions and no extra configuration will cause an error, saying that you cannot use property docker.imagePropertyConfiguration on projects with multiple images.

By adding an explicit external configuration directive with the same prefix in both images, this error is disabled.

```
<build>
  <plugins>
    <plugin>
      <groupId>io.fabric8</groupId>
      <artifactId>docker-maven-plugin</artifactId>
      <configuration>
        <images>
          <image>
            <external>
              <type>properties</type>
              <prefix>docker</prefix>
              <mode>override</mode>
            </external>

            <name>jolokia/demo</name>
            <alias>service</alias>
            <build>
              <from>consol/tomcat:7.0</from>
            </build>
          </image>

          <image>
            <external>
              <type>properties</type>
              <prefix>docker</prefix>
              <mode>override</mode>
            </external>

            <name>jolokia/demo2</name>
            <alias>service2</alias>
            <build>
              <from>consol/tomcat:7.0</from>
            </build>
          </image>
        </images>
      </configuration>
    </plugin>
  </plugins>
</build>
```

The behaviour will now be same as previous example. Note that you must explicitly state `<mode>override</mode>`, otherwise it will use the default `only`.

6.2. Docker Compose

This plugin supports also configuration via a docker-compose file, especially for running containers specified in `docker-compose.yml`. Docker Compose handling is available also as an external configuration provider.

Example

```
<image>
  <alias>webapp</alias> ①
  <name>fabric8/compose-demo:latest</name>

  <external> ②
      <type>compose</type> ③
      <basedir>src/main/docker</basedir> ④
      <composeFile>docker-compose.yml</composeFile>
  </external>

  <build> ⑤
    <assembly>....</assembly>
  </build>
  <run>...</run>
  <watch>...</watch>
</image>
```

① The alias of the image is used as correlation key mapping to a service in the Docker Compose file

② An `<external>` configuration handler needs to be used for Docker Compose support

③ The type for the external configuration provider must be set to **compose**

④ Additional configuration for the handler where to find the compose file

⑤ Extra `<build>`, `<run>` and `<watch>` configuration can be provided which are used as default configuration for the Docker compose service `webapp` (as specified with the alias)

The following options can be provided:

Table 37. Docker compose configuration

Element	Description	Default
basedir	Basedir where to find the compose file and which is also used as the current directory when examing the compose file. Any relative volume bindings will be resolved relative to this directory.	`${basedir}/src/main/docker`
composeFile	Name of the compose file to use	`docker-compose.yml`
ignoreBuild	Ignore the compose file's `build:` section and use the plugin's build configuration exclusively.	`false`

The Docker Compose file can contain variables as described in the [Docker Compose documentation](#). These are substituted with Maven project properties. Please note, when the `docker-compose.yml` with variables is to be used with the `docker-compose` CLI command, that these variables must also be valid environment variables (i.e. must not contain a `.`).

In addition to the `docker-compose.yml` you can add all known options for `<build>` , `<run>` and `<watch>` configuration elements which are then used as defaults and are overwritten by the configuration defined in the `docker-compose.yml` file. The merging between the XML configuration and the information found in `docker-compose.yml` is correlated via the `<alias>` name. E.g. if the XML

image configuration is aliased with `webapp` then its is used as a default configuration for a Docker Compose *service* name `webapp`. All other services defined in the compose file are left untouched.

6.2.1. Limitations

The following Docker Compose file keywords are not yet supported:

- `cgroup_parent`, `devices`, `env_file`, `expose`, `pid`, `security_opt`, `stop_signal`, `cpu_quota`, `ipc`, `mac_address`, `read_only` are not yet supported (but might be in a future version).

- `extend` for including other Docker Compose files is not yet implemented.

- Only **services** are currently evaluated, there is no supported yet for **volumes** and **networks**.

Chapter 7. Registry handling

Docker uses registries to store images. The registry is typically specified as part of the name. I.e. if the first part (everything before the first `/`) contains a dot (`.`) or colon (`:`) this part is interpreted as an address (with an optionally port) of a remote registry. This registry (or the default `docker.io` if no registry is given) is used during push and pull operations. This plugin follows the same semantics, so if an image name is specified with a registry part, this registry is contacted. Authentication is explained in the next section.

There are some situations however where you want to have more flexibility for specifying a remote registry. This might be because you do not want to hard code a registry into `pom.xml` but provide it from the outside with an environment variable or a system property.

This plugin supports various ways of specifying a registry:

- If the image name contains a registry part, this registry is used unconditionally and can not be overwritten from the outside.

- If an image name doesn't contain a registry, then by default the default Docker registry `docker.io` is used for push and pull operations. But this can be overwritten through various means:

  - If the `<image>` configuration contains a `<registry>` subelement this registry is used.

  - Otherwise, a global configuration element `<registry>` is evaluated which can be also provided as system property via `-Ddocker.registry`.

  - Finally an environment variable `DOCKER_REGISTRY` is looked up for detecting a registry.

This registry is used for pulling (i.e. for autopull the base image when doing a `docker:build`) and pushing with `docker:push`. However, when these two goals a are combined on the command line like in `mvn -Ddocker.registry=myregistry:5000 package docker:build docker:push` the same registry is used for both operation. For a more fine grained control, separate registries for *pull* and *push* can be specified.

- In the plugin's configuration with the parameters `<pullRegistry>` and `<pushRegistry>`, respectively.

- With the system properties `docker.pull.registry` and `docker.push.registry`, respectively.

```xml
<configuration>
  <registry>docker.jolokia.org:443</registry>
  <images>
    <image>
      <!-- Without an explicit registry ... -->
      <name>jolokia/jolokia-java</name>
      <!-- ... hence use this registry -->
      <registry>docker.ro14nd.de</registry>
      ....
    <image>
    <image>
      <name>postgresql</name>
      <!-- No registry in the name, hence use the globally
           configured docker.jolokia.org:443 as registry -->
      ....
    </image>
    <image>
      <!-- Explicitely specified always wins -->
      <name>docker.example.com:5000/another/server</name>
    </image>
  </images>
</configuration>
```

There is some special behaviour when using an externally provided registry like described above:

- When *pulling*, the image pulled will be also tagged with a repository name **without** registry. The reasoning behind this is that this image then can be referenced also by the configuration when the registry is not specified anymore explicitly.

- When *pushing* a local image, temporarily a tag including the registry is added and removed after the push. This is required because Docker can only push registry-named images.

Chapter 8. Authentication

When pulling (via the `autoPull` mode of `docker:start`) or pushing image, it might be necessary to authenticate against a Docker registry.

There are six different locations searched for credentials. In order, these are:

- Providing system properties `docker.username` and `docker.password` from the outside.

- Providing system properties `registry.username` and `registry.password` from the outside.

- Using a `<authConfig>` section in the plugin configuration with `<username>` and `<password>` elements.

- Using OpenShift configuration in `~/.config/kube`

- Using a `<server>` configuration in `~/.m2/settings.xml`

- Login into a registry with `docker login` (credentials in a credential helper or in `~/.docker/config.json`)

Using the username and password directly in the `pom.xml` is not recommended since this is widely visible. This is easiest and transparent way, though. Using an `<authConfig>` is straight forward:

```
<plugin>
  <configuration>
    <image>consol/tomcat-7.0</image>
    ...
    <authConfig>
      <username>jolokia</username>
      <password>s!cr!t</password>
    </authConfig>
  </configuration>
</plugin>
```

The system property provided credentials are a good compromise when using CI servers like Jenkins. You simply provide the credentials from the outside:

Example

```
mvn -Ddocker.username=jolokia -Ddocker.password=s!cr!t docker:push
```

The most *mavenish* way is to add a server to the Maven settings file `~/.m2/settings.xml`:

```
<servers>
  <server>
    <id>docker.io</id>
    <username>jolokia</username>
    <password>s!cr!t</password>
  </server>
  ....
</servers>
```

The server id must specify the registry to push to/pull from, which by default is central index `docker.io` (or `index.docker.io` / `registry.hub.docker.com` as fallbacks). Here you should add your docker.io account for your repositories. If you have multiple accounts for the same registry, the second user can be specified as part of the ID. In the example above, if you have a second account 'fabric8io' then use an `<id>docker.io/fabric8io</id>` for this second entry. I.e. add the username with a slash to the id name. The default without username is only taken if no server entry with a username appended id is chosen.

The most *secure* way is to rely on docker's credential store or credential helper and read confidential information from an external credentials store, such as the native keychain of the operating system. Follow the instruction on the docker login documentation.

As a final fallback, this plugin consults `$DOCKER_CONFIG/config.json` if `DOCKER_CONFIG` is set, or `~/.docker/config.json` if not, and reads credentials stored directly within this file. This unsafe behavior happened when connecting to a registry with the command `docker login` from the command line with older versions of docker (pre 1.13.0) or when docker is not configured to use a credential store.

8.1. Pull vs. Push Authentication

The credentials lookup described above is valid for both push and pull operations. In order to narrow things down, credentials can be provided for pull or push operations alone:

In an `<authConfig>` section a sub-section `<pull>` and/or `<push>` can be added. In the example below the credentials provider are only used for image push operations:

Example

```xml
<plugin>
  <configuration>
    <image>consol/tomcat-7.0</image>
    ...
    <authConfig>
      <push>
          <username>jolokia</username>
          <password>s!cr!t</password>
      </push>
    </authConfig>
  </configuration>
</plugin>
```

When the credentials are given on the command line as system properties, then the properties `docker.pull.username` / `docker.pull.password` and `docker.push.username` / `docker.push.password` are used for pull and push operations, respectively (when given). Either way, the standard lookup algorithm as described in the previous section is used as fallback.

8.2. OpenShift Authentication

When working with the default registry in OpenShift, the credentials to authenticate are the OpenShift username and access token. So, a typical interaction with the OpenShift registry from the outside is:

```
oc login
...
mvn -Ddocker.registry=docker-registry.domain.com:80/default/myimage \
    -Ddocker.username=$(oc whoami) \
    -Ddocker.password=$(oc whoami -t)
```

(note, that the image's username part ("default" here) must correspond to an OpenShift project with the same name to which you currently connected account has access).

This can be simplified by using the system property `docker.useOpenShiftAuth` in which case the plugin does the lookup. The equivalent to the example above is

```
oc login
...
mvn -Ddocker.registry=docker-registry.domain.com:80/default/myimage \
    -Ddocker.useOpenShiftAuth
```

Alternatively the configuration option `<useOpenShiftAuth>` can be added to the `<authConfig>` section.

For dedicated *pull* and *push* configuration the system properties `docker.pull.useOpenShiftAuth` and `docker.push.useOpenShiftAuth` are available as well as the configuration option `<useOpenShiftAuth>`

in an `<pull>` or `<push>` section within the `<authConfig>` configuration.

If `useOpenShiftAuth` is enabled then the OpenShift Konfiguration will be looked up in `$KUBECONFIG` or, if this environment variable is not set, in `~/.kube/config`.

8.3. Password encryption

Regardless which mode you choose you can encrypt password as described in the Maven documentation. Assuming that you have setup a *master password* in `~/.m2/security-settings.xml` you can create easily encrypt passwords:

Example

```
$ mvn --encrypt-password
Password:
{QJ6wvuEfacMHklqsmrtrn1/ClOLqLm8hB7yUL23KOKo=}
```

This password then can be used in `authConfig`, `docker.password` and/or the `<server>` setting configuration. However, putting an encrypted password into `authConfig` in the `pom.xml` doesn't make much sense, since this password is encrypted with an individual master password.

8.4. Extended Authentication

Some docker registries require additional steps to authenticate. Amazon ECR requires using an IAM access key to obtain temporary docker login credentials. The `docker:push` and `docker:pull` goals automatically execute this exchange for any registry of the form *<awsAccountId>* **.dkr.ecr.** *<awsRegion>* **.amazonaws.com**, unless the `skipExtendedAuth` configuration (`docker.skip.extendedAuth` property) is set true.

Note that for an ECR repository with URI `123456789012.dkr.ecr.eu-west-1.amazonaws.com/example/image` the d-m-p's `docker.registry` should be set to `123456789012.dkr.ecr.eu-west-1.amazonaws.com` and `example/image` is the `<name>` of the image.

You can use any IAM access key with the necessary permissions in any of the locations mentioned above except `~/.docker/config.json`. Use the IAM **Access key ID** as the username and the **Secret access key** as the password. In case you're using temporary security credentials provided by the AWS Security Token Service (AWS STS), you have to provide the **security token** as well. To do so, either specify the `docker.auth` system property or provide an `<auth>` element alongside username & password in the `authConfig`.

d-m-p will attempt to read AWS credentials from some well-known spots in case there is no explicit configuration:

- it will pick up ENV variables as documented for the AWS CLI

- it will pick up temporary credentials of the IAM role of an EC2 instance

- it will pick up temporary credentials of the IAM role of a fargate task (OR ECS with EC2 with ECS_AWSVPC_BLOCK_IMDS as "true")

If any of these authentication information is accessible, it will be used.

> For a more complete, robust and reliable authentication experience, you can add the AWS SDK for Java as a dependency.

```
<plugins>
    <plugin>
        <groupId>io.fabric8</groupId>
        <artifactId>docker-maven-plugin</artifactId>
        <dependencies>
            <dependency>
                <groupId>com.amazonaws</groupId>
                <artifactId>aws-java-sdk-core</artifactId>
                <version>1.11.707</version>
            </dependency>
        </dependencies>
    </plugin>
</plugins>
```

> This extra dependency allows the usage of all options that the AWS default credential provider chain provides.
>
> If the AWS SDK is found in the classpath, it takes precedence over the custom AWS credentials lookup mechanisms listed above.

Chapter 9. Implicit properties

There are some implicit configurations in docker maven plugin that are not so straightforward. These are simply workarouds to get docker-maven-plugin's flow right; just to overcome limitations of Maven and other things. Some of these are mentioned below:

- If the only value of the `env` parameter is a docker-maven-plugin internal property which has been set implicitly you have to prefix the property with a single `+` like in `+${docker.container.test.ip}`. This is necessary due to some Maven limitations which simply interpolates a lone, non defined property, to an empty string which can't then be replaced by this plugin after the initial interpolation phase.

- When providing port mapping in a format like `host.ip:host.port:80`, you need to prefix property with a single `+`. In this form, the host ip of the container will be placed into a Maven property name host.ip. If docker reports that value to be 0.0.0.0, the value of docker.host.address will be substituted instead. In the event you want to use this form and have the container bind to a specific hostname/ip address, you can declare a Maven property of the same name (host.ip in this example) containing the value to use. host:port works in the same way as described above.

Chapter 10. Further reading

- Examples:

    ◦ Examples are below `samples/` and contain example setups which you can use as blueprints for your own projects.

    ◦ A Shootout for comparing docker maven plugins

    ◦ Another sample project with a Microservice and a Database.

- ChangeLog has the release history of this plugin.

- Contributing explains how you can contribute to this project. Pull requests are highly appreciated!