|  |
| --- |
| Tibco Software Inc |
| BusinessEvents Docker |
| Functional Specifications Document |

Revision History

|  |  |  |
| --- | --- | --- |
| Author | Date | Summary of Change |
| Vandit Sharma | 1st Sept 2016 | Initial version |

Contents

[1 Requirement 3](#_Toc460487537)

[2 Overview 3](#_Toc460487538)

[3 Functional Specifications 3](#_Toc460487539)

[3.1 BE Dockerfile 3](#_Toc460487540)

[3.1.1 Environment Variables 3](#_Toc460487541)

[3.1.2 Volumes 4](#_Toc460487542)

[3.1.3 Ports 4](#_Toc460487543)

[3.2 BE Dockerfile image building script 4](#_Toc460487544)

[3.3 BE Application Dockerfile 5](#_Toc460487545)

[3.4 Build Application Docker image 7](#_Toc460487546)

[4 Working with Docker 8](#_Toc460487547)

[4.1 Install Docker 8](#_Toc460487548)

[4.2 Create a bridge network 8](#_Toc460487549)

[4.3 Running the BE application 8](#_Toc460487550)

[4.3.1 Run cache PU 9](#_Toc460487551)

[4.3.2 Run inference PU 9](#_Toc460487552)

[4.3.3 Running RMS 9](#_Toc460487553)

# 

# Requirement

The ability to run a BE application in docker. To provide the necessary tools and scripts to Dockerize a BE application

# Overview

A BE application comprises of a common BE runtime and project (application) specific BE code running inside the BE runtime. Hence, a parallel approach is taken to dockerize a BE application. i.e;

**A BE product base image**

This is not a runnable image. Running this image would result in an error, since it does not contain application code.

**A BE application specific Image**

This image builds on the product base image by adding project specific artifacts to the image. This is a runnable image which essentially runs the BE application.

# Functional Specifications

## BE Dockerfile

Dockerfile for the BE product installation shall be provided. It will be located in BE\_HOME/docker/bin

The following section explains certain aspects of this Dockerfile

### Environment Variables

The dockerfile exposes the following environment variables :

* CDD\_FILE : The applications CDD.The application’s dockerfile supplies this value.
* EAR\_FILE : The application’s enterprise archive .The application’s dockerfile supplies this value.
* PU : The name of the PU to run .Its value is supplied at the runtime by the user.
* AS\_DISCOVER\_URL : The AS discovery url.Its value is supplied at the runtime by the user.
* ENGINE\_NAME : Always defaults to be-engine.
* LOG\_LEVEL : Defaults to “na” unless overridden at the time of *docker run.*

### Volumes

Sometimes users may want to persist their data across docker runs. Examples are for ActiveSpaces Shared Nothing file stores and log file locations. To facilitate this, the following Docker volumes are created and all internal file paths are rooted to these directories. The following values are overridden in the CDD which built into the applicaition image

* Log Directory :The log file directory. It is overridden to **/mnt/tibco/be/logs**
* Shared Nothing’s Data Store : overridden to **/mnt/tibco/be/data-dir**

These directories are created as separate volumes in the docker image.

With this, if users want to persist data across runs, they would have to map the volume to the host file system during docker run

### Ports

By default the following ports are exposed by the base image.

* 50000 : This is the AS listen port exposed by the base image.
* 5555 : This is the JMX port exposed by the base image.

These can be port mapped during docker run

## BE Dockerfile image building script

A script BE\_HOME/docker/bin/build\_be\_image.sh(build\_be\_image.bat on windows) to build the product docker image shall be provided. To run, go to BE\_HOME/docker/bin and run

./build\_be\_image.sh -l <INSTALLERS\_LOCATION> -v <BE\_VERSION> -e <BE\_EDITION> -i <IMAGE\_VERSION> -a <BE\_ADDONS> --hf=<BE\_HOTFIX> --as-hf=<AS\_HOTFIX> -d <DOCKERFILE>

It is required that you run the script from BE\_HOME/docker/bin folder as there are other files in that folder that the script depends upon. This script will make necessary validations to ensure that a consistent docker image will be produced.

**Command line options:**

* INSTALLERS\_LOCATION[-l||--installers-location] : The location where Tibco BusinessEvents and Tibco Activespaces installers are located.[Required]
* BE\_VERSION[-v|--version] : The version of BE for which the docker image is to be generated.[Required]
* IMAGE\_VERSION[-i|--image-version] : The version of the generated image
* BE\_EDITION[-e|--edition] : The BE edition to be installed “standard” or “enterprise”.[Required]
* BE\_ADDONS[-a|--addons] : Comma separated values for required addons.Valid values : “process”/”views”/”datamodeling”/”decisionmanager” [Optional]
* BE\_HOTFIX[--hf] : The number of the hotfix to be installed with BE [Optional]
* AS\_HOTFIX[--as-hf] : The number of the hotfix to be installed with AS [Optional]
* DOCKERFILE[-d|--docker-file] : A Dockerfile to use. The default is Dockerfile [Optional]

*NOTE: Pass the long arguments with “=”*

**Example Command**

./build\_product\_image.sh -l /home/pkgs/5.4 -e standard -v 5.4.0 -i v01 -a process --hf=1

## BE Application Dockerfile

Before you build the application docker image, you have to generate an application specific Dockerfile. A utility to build the application specific docker file shall be provided BE\_HOME/be/5.4/docker/bin/be-docker-gen

To run go to BE\_HOME/docker/bin and run

./be-docker-gen -t <TARGET\_DIRECTORY> -i <BE\_BASE\_DOCKER\_IMAGE> -m <MAINTAINER> -e <EMAIL> -l <LABEL> -h <HELP>

**Command line options**

* TARGET\_DIRECTORY[-t||-target-dir] : The directory will hold the application’s cdd , application’s ear ,external jars[Required]
* BE\_BASE\_DOCKER\_IMAGE[-i|-image] :The name of the base BE Docker image which will be used to run this application.[Required]
* MAINTAINER[-m|-maintainer] : maintainer [Required]
* EMAIL[-e|-email] : email [Required]
* LABEL[-l|-label] : label.Can be multiplel[Optional]
* HELP[-h|-help] : Displays this usage

**Example Command**

./be-docker-gen -t /home/app -i 5.4.0-v01 -m FraudDetection -e [fdcache@abc.com](mailto:fdcache@abc.com) –l ApplicationName=FraudDetection

**Custom Docker File Support**

If user want to include additional environment variables in the generated Dockerfile, then user can include a custom file with an extension “**.custom”** in the target *directory*. All such files will be merged with the generated dockerfile.

**External Jars support**

If an application is dependent on some external/third party jars, user can copy the required jars to the *target directory.* These jars will be copied to the image and added to the engine’s classpath.

**GV support**

While generating the dockerfile, the utility queries the Enterprise archive and cdd and exposes the Global variables as Environment variables in the dockerfile.

If a GV is overridden in the CDD, the CDD global variable will take precedence and the dockerfile will be updated with the CDD value.

**Docker PORT Support**

An application might need to expose its network ports. For any network port that needs to be exposed, use BE Global Variable for such ports and ensure that name of the Global Variable has a suffix “**\_PORT**” in it. All such Global Variables will be exposed using docker EXPOSE

**Docker VOLUME Support**

If an application needs to access an external file system or needs to persist file system data across docker runs, then those file system paths are required to be exposed as Volumes in the dockerfile. Any such file system paths should similarly use Global Variables and the name of the Global Variable should contain a suffix **“\_PATH**”. All such Global Variables will be mapped to Docker VOLUMES

**Backingstore Shared Resource support**

If an application has a Backing Store with “Shared All” mode then the associated Shared Resource‘s properties are exposed in the Docker file as Environment Variables and the same variables are updated in the Shared Resource of the backingstore as Global Variables.

The following properties are exposed as GVs:

* BACKINGSTORE\_JDBC\_DRIVER : Driver
* BACKINGSTORE \_JDBC\_URL : Database Url
* BACKINGSTORE \_JDBC\_USERNAME :Username
* BACKINGSTORE \_JDBC\_PASSWORD : Password
* BACKINGSTORE \_JDBC\_POOL\_SIZE : Maximum Connections
* BACKINGSTORE \_JDBC\_LOGIN\_TIMEOUT :Login Timeout
* BACKINGSTORE \_JDBC\_USE\_SSL : Use SSL

**Note**: Before generating the dockerfile for a Shared Resource, make sure the required driver libraries are present in the target directory.

**Docker File Support**

If an application needs to access an external file for a docker run then such files are needed to be included in the docker image. To include these files, users will have to declare a GV in the application project with the suffix “**\_FILE**” and the path to the file as the GV value. Additionally all such files must be included in the target directory under the folder “**files**” .This utility creates directories for the file paths and adds the files to the specified location in the docker file.

Also the location of the file is created as a Volume in the dockerfile , so that user can map that location and update/add files at runtime.

## Build Application Docker image

A script BE\_HOME/docker/bin/build\_app\_image.sh(build\_app\_image.bat on windows) to build the application docker image shall be provided. To run go to BE\_HOME/docker/bin and run

./build\_app\_image.sh <APP\_IMAGE\_NAME>:<APP\_IMAGE\_VERSION> <TARGET\_DIRECTORY>

**Command line arguments**

* APP\_IMAGE\_NAME :The name of the application image[Required]
* APP\_IMAGE\_VERSION :The app image’s version[Required]
* TARGET\_DIRECTORY :The directory where the generated Dockerfile , CDD and EAR files are present.[Required]

**Example**

./build\_app\_image fdcache:v01 /home/temp

A Dockerfile will be created in the target directory. If there already exists a Dockerfile, it will be backed up to Dockerfile.1 etc.

There will be one Docker ENV for each of the Global Variables in the project

All Global Variables that have a “PORT” substring in their name will be assumed to be network ports which will be EXPOSED in dockerfile

All Global Variables that have a “\_PATH” in their substring will be exposed as docker VOLUMEs

**Additional engine Properties Support**

Additional engine properties can be baked into the application image. To do that, add a properties file with an extension of “.props” in the target folder. These will be made available to the BE docker image.

**NOTE**:

For windows all arguments must be encapsulated in double quotes.

# Working with Docker

## Install Docker

Install Docker version 1.11 or higher (required for ‘docker network’ feature) for your OS and ensure initial docker setup. See if it works by issuing simple docker commands like

*docker --help* or *docker –version*

## Create a bridge network

User will have to create a bridge network which will be used for inter docker image communication.

docker network create <BRIDGE\_NAME>

* BRIDGE\_NAME :The name of the network bridge[Required]

**Example**

docker network create be\_network

## Running the BE application

To run the application, we will use the docker run command.

*docker run --net=<BRIDGE\_NETWORK> --name=<CONTAINER\_NAME> -e PU=<PU\_NAME> -e <ADDITIONAL\_ENV\_VARS> <APPLICATION\_IMAGE\_NAME>*

* BRIDGE\_NETWORK :The name of the created bridge network[Required]
* CONTAINER\_NAME: The name of the container which will run this image.[Required]
* PU\_NAME : The name of the PU which will be started as part of this command.[Required]
* ADDITIONAL\_ENV\_VARS: Users can pass additional environment variables using the -e flag.

**Example**

docker run --net=be\_netowrk --name=cache -e PU=cache -e fdcache:v01

### Run cache PU

To run the cache PU use the following command

docker run --net=simple-bridge --name=cache -e PU=cache fdcache:v01

### Run inference PU

To run the inference agent use the following command.

docker run --net=simple-bridge --name=inference -P -e AS\_DISCOVER\_URL=tcp://cache:50000 -e PU=default -p 8109:8109 fdcache:v01

Note how the docker name of the cache server “cache” is used in the AS\_DISCOVER\_URL of the inference agent. All agents running on the same docker host can resolve docker “names” to their IP addresses on this network and hence clustering across instances on the same docker host is possible.

### Running RMS

Follow the following steps to generate and run RMS docker image.

1. Generate BE product docker image

Generate a Base BE docker image just as mentioned in previous steps.

1. Generate RMS application Dockerfile

Just the way we generate an application’s dockerfile, you can generate the RMS Dockerfile by putting RMS.ear and RMS.cdd in a target directory.

1. Generate RMS Docker image
2. This step is also same as generating the application docker image.Follow the steps as is.
3. Run the RMS Server

To start the server run the following command:

docker run --net=be\_network --name=<RMS\_CONTAINER\_NAME> -e **COMPONENT=”rms”** -e PU=default <RMS\_IMAGE\_NAME>

* RMS\_CONTAINER\_NAME :The name of the container which will run the rms image[Required]
* RMS\_IMAGE\_NAME : The name of the generated RMS Image

NOTE : Make sure the environment variable **COMPONENT=”rms”** is passed to the command.

### Overriding Log Levels

To override log levels at runtime , users can pass additional Environment variable **LOG\_LEVEL** to override the predefined log levels. Users can specify the desired comma separated log patterns for this variable and the engine will use these as the Log configuration. If not specified , the default log configuration in the Cdd will be used.

Example:

docker run --net=simple-bridge --name=cache -e PU=cache –e **LOG\_LEVEL=\*:debug** fdcache:v01

Note : The pattern configuration is same as given in the CDD.

### RMS Volumes

By default the BE base image exposes various volumes where RMS and Webstudio persists data. User can map these volumes to their local file system to persist the data. They are:

* **/opt/tibco/be/*BE\_SHORT\_VERSION*/rms/config/security**

This directory hold the RMS application’s ACL(permission configuration) and user.pwd files.

* **/opt/tibco/be/*BE\_SHORT\_VERSION*/examples/standard/WebStudio**

This directory is the repo directory for Webstudio which will hold all projects.

* **/opt/tibco/be/*BE\_SHORT\_VERSION* /rms/config/notify**

Email notification configuration files are store here.

* **/opt/tibco/be/*BE\_SHORT\_VERSION* /rms/lib/locales**

This directory holds the User locale configuration.

* **/opt/tibco/be/*BE\_SHORT\_VERSION* /rms/bin/logs**

RMS and Webstudio logs are stored here.

**BE\_SHORT\_VERSION:** The be version in the short form. For example for BE Version 5.4.0 BE\_SHORT\_VERSION will be 5.4.

### Remote JMX Support

To enable remote JMX connections , instances must be started with the **DOCKER\_HOST** environment variable , whereas its value will be the address of the host where *docker runis* executed.

docker run … -e DOCKER\_HOST=xx.xx.xx.xx ………

Note:The default JMX port for engines running inside docker is **5555** , this port must be mapped with a local port at runtime using *–p* flag.

Example :

docker run --net=be\_network --name=sample –p 5555:5555-e PU=default –e DOCKER\_HOST=10.97.123.56 sample:v01

### AS Remote Support

To start a node as a Proxy node, user will have to start the instance with **the AS\_PROXY\_NODE=true** flag. :

docker run ... –e AS\_PROXY\_NODE=true ...

The port: **50001** will be the default AS remote listen port. User can specify this port while connecting to the proxy node in the following manner:

docker run ... -e AS\_DISCOVER\_URL=tcp://<container\_name>:50001?remote=true ...

for example :

AS\_DISCOVER\_URL=tcp://cache:50001?remote=true

### Environment Variables

The following are the list of environment variables supported at runtime :

* **AS\_DISCOVER\_URL** : Denotes the discover url , which enables members to discover each other.
* **COMPONENT**: This environment variable is needed to run RMS.Must be supplied with the value “rms”.
* **PU**: Specifies the PU to be started.
* **LOG\_LEVEL**: Overrides the predefined log levels with the specified value. The value pattern is same as provided in the log-config of the cdd.
* **DOCKER\_HOST**: Specifies the Host, where the ‘*docker run*’ is executed. It is necessary for remote JMX connections to the running container.
* **AS**\_ **PROXY**\_ **NODE:** This flag denotes the running container will run as a Proxy node or not.Pass the value as “true” if the node is needed to be started in proxy mode.

# Docker across hosts

## Setup a Swarm

Follow the instructions described by the following article to setup a Swarm Cluster:

<https://docs.docker.com/engine/userguide/networking/get-started-overlay/>

## Running BE

Once your Swarm setup is complete including the overlay network, user can deploy/start BE nodes just as a single docker host is started. All name/IP resolution is provided by the overlay network and the distribution of service (in this case BE nodes) is provided by Swarm. You can add "–constraint:" parameters to control on which docker host to start the instance instead of relying on Swarm to determine the host.

Extending the aforementioned *fdcache* example.

Install the BE product and docker image on the ***mhs-demo0*** and ***mhs-demo1*** nodes (if it is done via the Swarm manager environment, it only creates the image on a non-manager node)

Setup docker client to point to the Swarm manager node.

eval $(docker-machine env --swarm mhs-demo0)

* Run Cache PU

docker run --net=be\_network --name=cache --env="**constraint:node==mhs-demo0**" -e PU=cache fdcache:v01

* Run Inference PU

docker run --net= be\_network --name=inference --env**="constraint:node==mhs-demo1"** -e AS\_DISCOVER\_URL=tcp://cache:50000 fdcache:v01

Now the cache and inference nodes are now running on different hosts (mhs-demo0 and mhs-demo1) respectively and are able to form a cluster.