



digital
vision
group

TRIP on Docker

TRIP

Product Documentation

End User License Agreement

All rights to this software, its documentation and logotypes of the TRIP product family and software (altogether “Software”) supplied by DVG Operations GmbH (DVG) are exclusively owned by DVG.

The transfer of this Software, solutions or parts thereof requires the prior written agreement of DVG. Furthermore, the customer has the right to use licensed Software and / or process solutions supplied by DVG to the extent specified in his contract with DVG.

The free-to-use non-commercial version doesn't require a prior written agreement with DVG but such customers, organizations and/or third parties agree by using the software and / or solution of DVG to be strongly obliged to keep all rights to this software, documentation and logotypes of the TRIP product family absolutely uninfringed and protected.



Table of Contents

Introduction	5
About this Document	5
About TRIP on Docker	5
Installation and Configuration	6
Obtaining the Docker Image	6
Container Configuration	6
Container user	6
Data Volume	6
License Installation	7
Token Encryption Key	8
CONTROL Database Migration	9
Optional SSH Daemon	10
Post-Deployment TRIP Configuration	10
Data Volume Directory	12
Data Volume Directory Tree Structure	12
Creating the Volume	12
Notes on Files in the Volume	13
Access Permissions	13
Configuration Files	13
Licenses	13
Log Files	13
User Databases	13
The CONTROL Database	14
Log File Maintenance	14
Migrating to Docker	15
Step 1: Create a Volume and start the container	15
Step 2: Add a (site) license	15
Step 3: Migrate the CONTROL database	15
Step 4: Adapt configuration	16
Set up storage locations on the data volume	16
Logs for accounting (debit)	16
Other settings	17

Step 5: Adapt user databases to the 4096-byte block size	17
Database contents (BAF)	17
Database indexes	18
Using server-side tools	19
Extending the Image.....	20
The Dockerfile	20
Initialization scripts and the entry point	20
Editing tdbbs.conf	20
Background processes and controlled startup and shutdown	20
Example: Adding an ASE library	21
Dockerfile Example	21
Initialization Script Example	22

Introduction

About this Document

This document describes the deployment, administration and use of TRIP in a Docker container environment.

This document does not go into details about Docker itself. Please refer to information on Docker available on the internet and on the Docker website for this purpose.

About TRIP on Docker

Docker is a software for running containers, light-weight virtual runtime environments that run in the context of an existing operating system, emulating the OS rather than the underlying hardware. Docker, along with orchestration tools such as Kubernetes, is a popular example of this type of technology.

The Linux version of the TRIP platform can be deployed as a Docker image. The current images contain TRIPsystem and TRIPcof and are available as based on the following Linux distributions:

- RedHat Enterprise Linux 9.6 (Universal Base Image)

Installation and Configuration

Obtaining the Docker Image

The Docker image for TRIP is available in a Docker registry operated by DVG Operations GmbH (DVG). Send a request to DVG in order to get information about the address and an account for it.

The below examples as well as other examples in this document uses a fake address to exemplify the operations. Replace this (`registry.example.com:5050/trip/`) with the address DVG has provided to you.

After having received an access token from DVG, use it to login to the registry with Docker. Specify only the host and port parts of the registry's address here:

```
docker login registry.example.com:5050
```

When you have successfully authenticated your Docker client, you can retrieve a TRIP image. To pull the RHEL9-based image that contains the backend server software for TRIP 8.4:

```
docker pull registry.example.com:5050/trip/tripserver-rhel9:8.4
```

The latest version is always tagged as `:latest`. Images are also tagged with the version number in the following variations:

- Major only (e.g. `:8`)
- Major and minor (e.g. `:8.4`)
- Major, minor and service (e.g. `:8.4.2`)
- Major, minor, service and patch (e.g. `:8.4.2.1`)

Container Configuration

Container user

In standard Docker, the TRIP processes are set up to run as the user 'trip' with uid 1001 in the container. This user is predefined by the image and the container itself is started as that user. The root user is not used at all in the container.

This default does not apply to orchestration software such as OpenShift that create a random uid for the container user. This means that when deploying, extending or using the TRIP container, you must not rely on a specific pre-defined user or user id.

In either case, the primary group of the container user is the group with ID 0. This must not be overridden.

Data Volume

Docker containers are for all intents and purposes stateless. This means that any data that must persist across container restarts and reinitializations must reside elsewhere. The Docker solution for this involves a so-called Volume.

A Docker volume is a directory pre-declared in the Docker image being used. When a container based on the image is initialized and started, a mapping must be supplied for what external area should be used for that storage.

For more information on volumes, refer to the "Data Volume Directory" chapter later in this document.

License Installation

In order to use TRIP in Docker you will need a site license. Request one from your TRIP distributor. An additional license fee may be charged if you do not already have a site license for TRIP.

Automated License Installation using a Docker Secret

The recommended way to install a license is to pass it to the container as a Docker secret. This also allows for automation of license installation, but requires that you are deploying your TRIP container in an environment that support Docker secrets. This is supported by all the common container orchestration environments as well as by Docker Swarm, but not when using a standalone Docker installation.

TRIP licenses represented as Docker secrets must reside in the `/etc/tripsystem/license` directory in the container. You can name the license files as you wish, but they must always end with the `.lic` suffix.

Example of deploying TRIP with automated license installation using Docker Swarm:

```
# Create the secret containing the license

docker secret create trip-license /path/to/license/trip.lic

# Deploy the container

docker service create --name trip \
  --workdir /opt/trip/sys/current/bin \
  --mount source=trip-data,target=/var/lib/trip \
  --secret source=trip-license,target=/etc/tripsystem/license/trip.lic \
  -p 23567:23567 \
  -u 1001:0 \
  -e TZ=Europe/Berlin \
  registry.example.com:5050/trip/tripserver-rhel9:8.4
```

Note that although the above example is just for the Docker Swarm, the same thing can be accomplished by in other container orchestration environments, although syntactically different. Refer to the documentation for your container environment for more information about how to use secrets.

Automatically Updating a License using a Docker Secret

If you have a installed a license automatically using a Docker secret, you may sooner or later wish to update the license without having to recreate the container.

The exact steps involved in doing this varies depending on which orchestration environment you are using, but in general they are:

1. Create a new secret for the new license, giving it a different ID than the old one.
2. Assign the new secret to the container
3. Remove the old secret from the container

Depending on your environment, steps 2 and 3 may have to be done at the same time or in the reverse order.

Example for updating a license in a Docker Swarm deployed TRIP container:

```
# Create the new secret containing the license

docker secret create license-new /path/to/new/license/trip.lic

# Update the container (causes it to restart)

docker service update \
  --secret-rm trip-license \
```

```
--secret-add source=license-new,target=/etc/tripsystem/license/new.lic \
mytripservice
```

Manual License Installation with TRIPmanager

If you are unable to use a Docker secret to install a TRIP license into the container, you can choose to install the license manually instead. Manual license installation into a container is only supported via TRIPmanager, which ensures that the license data is properly persisted on the data volume with file access privileges set correctly.

Manually installed licenses are persisted on the data volume in the `lic` directory, with the fully qualified path in the container being `/var/lib/trip/lic`.

License Installation Caveats and Notes

Automated and manual license installation are **mutually exclusive**. If you have started your container using an automated approach to install the license, you must continue to use the same automated approach for any license updates you need to make in the container. The reverse also applies; if the initial license has been supplied manually, later use of one of an automated approach with the same container is not supported.

Do NOT be tempted to utilize other means to upload the license file (e.g., to use the “docker cp” command). This is not supported. Doing so anyway will very likely cause the file ownership and access permissions not to be properly assigned (owned by the user that the container starts as and assigned to group 0, with file permissions set to `rw-rw-r`). While the license will be readable to TRIP, you will encounter problems removing or updating the license later on.

Note that the TRIP license is valid for a specific minor version (e.g., 8.2, 8.3, 9.0 etc). If you move from TRIP 8.2 to 8.3, your old licenses will no longer be valid. In this case, remove the old licenses from the container and request new ones from your TRIP distributor.

If you previously have used the manual approach to install the license and wish to start using the automated one based on secrets, you should remove the persisted license file from the data volume manually before recreating the container with the new image and a license secret configured. This can for example be done by starting a shell in the container as the user the container is running as.

Token Encryption Key

Containers that will be used with the access token functionality need to be initialized with a 256-bit AES encryption key. Without a such a key, the access token functions will be disabled.

Prepare this key by creating a 256-bit (32-byte) file filled with cryptographically random data. For example, to create a “tokenkey” file using OpenSSL:

```
openssl rand -out tokenkey 32
```

Key Deployment as a Docker Secret

The best way to make this key available to the container is to use a Docker Secret. Do this in the same way as described for licenses above. In Docker Swarm, the command would be similar to this:

```
docker secret create trip-tokenkey /path/to/tokenkey
```

This will make the key available as `/run/secrets/trip-tokenkey` in the container.

Rotating the key is a security measure that sometimes is warranted. How often this should be done depends on the security practices of your organization.

For example, after a new key file has been created, the command would be similar to this if you are using Docker Swarm:

```
docker secret create trip-tokenkey-new /path/to/new/trip-tokenkey-new

docker service update \
  --secret-rm trip-tokenkey \
  --secret-add trip-tokenkey-new \
  mytripservice
```

Note that it is important that you create a new file for this and not overwriting the old one.

For further details about Docker Secrets, please refer to the official Docker documentation available on the docs.docker.com website.

Key Deployment as a Mounted File

Deploying the token encryption key as a file mount is less secure, but can be useful in some circumstances (e.g. a local development environment with less sensitive data).

For example, using the Docker `run` command, there would be a “`--mount`” option similar to this:

```
--mount type=bind,source=/path/to/trip-tokenkey,\
target=/run/secrets/trip-tokenkey,readonly
```

One small advantage with this variant is that key rotation is a bit simpler. It just involves stopping the container, updating the token key file on the host, then restarting the container.

Purge Tokens After Token Key Update

Rotating the token encryption key will immediately **invalidate all existing tokens**, so a rotation of this key should be followed up by a full purge of all tokens.

Rotating the token encryption key is one way to deal with a suspected token leak, although it means that all users need to acquire new tokens, not just the users directly affected by the leak. If you know which user or users are affected by a token leak, a better option might therefore be to revoke their keys only (refer to the TRIPsystem Token Access document for more information).

Configuring TRIP with the Token Encryption Key

In order for TRIP to be able to use the token encryption key you have made available for it requires the `tdbs.conf` file to specify the container-internal path to the key file in the `TDBS_TOKEN_KEY` privileged variable. A convenient way to do this during container startup is to specify the key path as an environment variable.

For example, with Docker `run`, there would be an `-e` option similar to this:

```
-e TDBS_TOKEN_KEY=/run/secrets/trip-tokenkey
```

This will edit the container's `tdbs.conf` file as needed during container startup.

CONTROL Database Migration

MODCON is always run when the container starts. This ensures that upgrading the image to a newer version is a transparent operation that does not require any additional adjustments to the environment.

However, please be aware that downgrading is NOT SUPPORTED! This is because the CONTROL database design changes that MODCON may introduce can be such that any attempt to use a newer CONTROL database with an older TRIPsystem is likely to fail due to the newer changes being seen as data corruption by older TRIPsystem versions. This “corruption” may not be immediately apparent, though.

The MODCON tool will from version 8.4-0 of TRIPsystem and its Docker image perform a check if it can reliably run (i.e. the version is the same as before or newer). If it can't, the container startup will abort, logging an error message "Unsupported CONTROL database migration path". Previous versions of TRIPsystem and the Docker image did not have this check, which could have led to a corrupt CONTROL database if a downgrade was attempted, despite it not being supported.

Optional SSH Daemon

The TRIP Docker image comes with an OpenSSH server installed. It is not activated by default. To activate it, specify the environment variable `USE_SSHD=1` as input (to "docker run" or to your deployment configuration) when creating the container.

There is no password-based login, so in order to actually access the container using ssh, you need to copy file `/etc/ssh/id_rsa` from the container to you client machine. If you are deploying a container under standard Docker, you can use the following command:

```
docker cp CONTAINERNAME:/etc/ssh/id_rsa container-key
```

Which will put the key in the local file "container-key" (replacing "CONTAINERNAME" with the actual name of your container).

If your container is running under SPSC OpenShift at DVG Operations GmbH, you will get a copy of the key from them should your agreement allow for ssh access.

Before proceeding, you also need to ensure that only the calling user has read permissions to the key file. For example, if you are using an SSH client on Linux or UNIX:

```
chmod 600 container-key
```

Depending on the port that the deployed container exposes the ssh service as (the default port is in this case 22222), you will use an SSH command similar to following in order to access the container:

```
ssh -i container-key -p 22222 trip@mycontainer.address.example.com
```

Post-Deployment TRIP Configuration

If you wish to edit the one of the TRIP configuration files (e.g. `tdbs.conf`), log in to the container. If you are already logged on to the Docker host, you can issue a command like this (replacing "CONTAINERNAME" with the actual name of your container):

```
docker exec -it -u trip CONTAINERNAME /bin/bash
```

You can also use SSH to access the container (see section "Optional SSH Daemon" above for more details).

To edit the `tdbs.conf` file using the `vi` editor in the container:

```
vi /var/lib/trip/conf/sys/tdbs.conf
```

Note that running TRIP processes will not be affected by a configuration change. Only sessions and processes started after the change has been introduced will be affected.

If you are making a change that will affect the daemon processes (e.g. `tripd` or `tripnetd`), you can stop them from within the console:

```
tripd -k 0
```

```
kill -TERM `ps a | grep tripnetd | grep -v grep | awk '{print $1}'`
```

This will cause the supervisor to restart the daemons. Note that if you wish to keep existing sessions alive during the tripnetd restart, you should use `kill -9` instead of `kill -TERM` to force the tripnetd restart. Using `-TERM` will cause all sessions (tbserver and TRIPclassic) to be shut down as well.



Data Volume Directory

When used in a Docker container, TRIP is set up to use a separate directory tree to hold all information that are regarded as persistent data. This is normally databases, configuration files and licenses.

This directory tree is set up as a Docker Volume by the TRIP Docker image, and is available to the container as:

```
/var/lib/trip
```

When you start your TRIP container, this directory must be mapped to a Docker volume (see section Data Volume in the chapter Installation and Configuration in this document).

IMPORTANT: Use a different volume with each container. Not doing so may cause runtime errors such as file access privilege problems that in some cases may cause container to stop working or not start at all. Each container must have its own unique volume assigned to it. The only circumstance in which a volume can be reused for other TRIP containers (e.g., after upgrading to a newer version of the TRIP Docker image) is if no other containers are using it or are registered for its use.

Data Volume Directory Tree Structure

The structure of the storage volume directory should be as follows:

```
/var/lib/trip
|
|-- conf
|   |-- sys          // TRIPsystem configuration (tdbs.conf, etc)
|   |-- cof          // TRIPcof configuration files
|
|-- ctl              // Location for TDBS_CTL
|-- db               // Location for TDBS_BASES
|-- lic              // TRIP license directory
|-- log              // Log file directory
```

Creating the Volume

While there are several ways to store data persistently with a Docker container, the one supported by TRIP is named volumes. To create a named volume, use the following Docker command:

```
docker volume create trip-volume
```

Note that the above command creates a volume that resides on the Docker host. While that may work for test installations and some small-scale production environments, such volumes are less suitable for larger scale production environments. Drivers for other types of volumes exist. Please consult with the Docker documentation and with your cloud / infrastructure provider for available options and how to configure them for Docker. Whatever type of volume driver you end up using, keep in mind that TRIP is very I/O intensive and that the volume therefore must have excellent access, read and write times.

Notes on Files in the Volume

Access Permissions

TRIP will normally manage the access permissions for the files and directories in the data volume. However, should you need manipulate the contents of manually from within the container or from elsewhere, please note the following.

Maintaining proper file ownership and access permissions is critical. Make sure the access permissions for all directories and files in the volume are set up so that all users in the container that need to read and/or write files here can do so. These operating system users will be the users that all server-side TRIP processes will be running as. When running the container under Docker itself (as opposed to under orchestration software such as OpenShift), this is normally the user “trip” (uid 1001 and gid 0). When running under orchestration software such as OpenShift the user ID is normally different and assigned when deploying the container.

The degree to which access permissions should be considered may depend on the kind of volume driver you are using, but generally speaking all files stored in the volume should be writable by both user and group.

Configuration Files

TRIPsystem will read the `tdbs.conf` file from the `conf/sys` directory of the data volume, and TRIPcof will read its configuration files from the `conf/cof` directory of the data volume. When using TRIP under Docker, no other configuration file locations are supported.

Licenses

When running TRIP in a Docker container, it will read license files installed manually using TRIPmanager from the `lic` directory of the data volume (automatically installed licenses are not stored on the data volume). License files must have the “.lic” suffix.

Use TRIPmanager to set the license manually after the container has been created, or use a Docker secret to provide a license in an automated fashion during container creation. Refer to section “License Installation” in this document for more information.

NOTE: Manually copying the license file into the TRIP container is not supported and is very likely to result in problems with file ownership and access permissions.

Log Files

The TRIP Docker image configures the `log` directory of the data volume to be the location where log files for the TRIP software are written. TRIPsystem log files are written to the `log/sys` directory and TRIPcof log files are written to the `log/cof` directory.

If you are using accounting (debit) logging, you should configure the `TDBS_ACCDIR` logical name in `tdbs.conf` instead of relying on the default location under `TDBS_SYS`. Put the accounting logs somewhere under the `log` directory in the data volume (e.g. `log/acc`).

User Databases

The `db` directory in the data volume is configured by the TRIP Docker image to be where user databases are located by default. This location is set up in the `tdbs.conf` file under the logical name `TDBS_BASES`.

You may choose another directory for your databases if you so choose, but remember to always use a logical name for the database file location and not specify them by a fully qualified path.

The CONTROL Database

The `ctl` directory of the data volume is dedicated to the TRIPsystem CONTROL database files. Aside from actions involving migrating to and from a Docker deployment of TRIP, you will normally not have to touch these files.

Log File Maintenance

The TRIP container comes with `logmaint`, a background process for automatic log file removal. It removes old log files from `tbserver`, Toolkit API, Kernel and XPI, as well as session index files (SIF) and `tripd` batch job logs for PRINT, UPDATE, INDEX, LOAD and LOADIX tasks.

Files associated with existing sessions are not removed even if they should meet the criteria for removal.

The following settings in the `tdbs.conf` configuration file control the behavior of this tool:

Logical Name	Default	Description
TDBS_LOGPRUNE_INTERVAL	60	The interval in minutes between scans for log files to remove.
TDBS_LOGPRUNE_MAX_LOGS	100	The max number of log files of each type to retain unless they are older than the configured max age.
TDBS_LOGPRUNE_MAXAGE	30	The max number of days to retain log files
TDBS_LOGPRUNE_MAXAGE_BATCH	7	The max number of days to retain logs for successful batch logs. The maximum age for batch job logs containing errors is defined by <code>TDBS_LOGPRUNE_MAXAGE</code>

For more information on these settings, please refer to the “TRIPsystem Environment” document in the TRIPsystem documentation set.

Migrating to Docker

Migrating a TRIP installation on a physical or virtual machine to a Docker container based one involves a few steps that should be done with some care. This chapter describes what you need to consider in such a circumstance.

1. Create a volume and associate the volume with a container and start the container
2. Add a (site) license
3. Migrate control database
4. Adapt configuration
5. Adapt user databases to 4k block size

Step 1: Create a Volume and start the container

If you don't already have a Docker volume you wish to use, create a new one. Refer to the previous chapters in this document and to Docker documentation for details.

When a volume is available, associate it with the Docker container you are about to deploy and start the container. Refer to the documentation for Docker and (if relevant) your orchestration software for details about how to do this.

Step 2: Add a (site) license

TRIP on Docker requires a site license in order to work. Request one from your TRIP distributor and install it on the container as described in the "Installation and Configuration" chapter of this document.

Note that if you wish to install the license in an automated fashion, this step is actually done as part of step 1.

Step 3: Migrate the CONTROL database

Copy the CONTROL database files (BAF, BIF and VIF) from the TRIP installation you are migrating from to the container. Place them in the `/var/lib/trip/ctl` directory. Take care when doing this since this may result in bad file ownership and access permissions, resulting in problems in accessing or even removing the files – especially if you are using the common "docker cp" command. Consider enabling the SSH server when deploying the container so that you can use the `scp` command to copy the files.

Verify that the `P_CONTROL.BAF`, `P_CONTROL.BIF` and `P_CONTROL.VIF` files are present in the `/var/lib/trip/ctl` directory. If they are not, copy them from the TRIPsystem's 'sys' directory, which normally is `/opt/trip/sys/current/sys`. Do NOT use `P_CONTROL` files from any other version of TRIP than the one that is running in the container. Doing so anyway will result in an incomplete migration.

When the CONTROL database files to migrate are in place, log in to the container and execute the `migrate_control` script which you can find in the `sbin` directory of the TRIPsystem installation (replacing "CONTAINERNAME" with the actual name of your container):

```
docker exec -it CONTAINERNAME /bin/bash

cd /opt/trip/sys/current/sbin

./migrate_control

exit
```

Alternatively, instead of logging in to the container and running `migrate_control` manually, you can just restart the container after having copied your existing CONTROL database files into it. The control databases will then automatically be migrated.

The procedure will create backup copies of the old CONTROL database files. The backup files are located in the same directory, but has a timestamp appended to the file suffix. At most 10 old backups are kept. If ten old backup copies already exist when the procedure starts, the oldest one will be deleted.

Step 4: Adapt configuration

Do NOT copy your existing `tdbs.conf` file to the Docker installation. Doing so may render it inoperable. Instead, introduce changes to the `tdbs.conf` in your Docker container as described below.

Set up storage locations on the data volume

If you have databases whose file locations for BAF, BIF, VIF and (optionally) LOG are set to a fully qualified path or that use a storage location logical name different than `TDBS_BASES`, these must be changed to use a logical name set up in the `tdbs.conf` file.

Note that no configuration action is required for those of your databases that use `TDBS_BASES` as storage location, i.e. has `TDBS_BASES` as part of the file name, like this:

```
TDBS_BASES:MYDATABASE.BAF
TDBS_BASES:MYDATABASE.BIF
TDBS_BASES:MYDATABASE.VIF
```

For those of your databases that use another storage location that you have defined in your `tdbs.conf`, and for databases that use a fully qualified path to the database files, you will have to do one of the following:

- Update the `tdbs.conf` file in the Docker container with a logical name for your storage location. Set its path to a directory in the Volume (i.e. somewhere under `/var/lib/trip`), or
- Modify the database design to use `TDBS_BASES` as storage location instead.

When done, copy the database files to the relevant volume directory in container (e.g. to `/var/lib/trip/db` for databases that use `TDBS_BASES`). Consider enabling the SSH server when deploying the container so that you can use the `scp` command to copy the files.

Logs for accounting (debit)

If you are using accounting (debit) logging, you should define the `TDBS_ACCDIR` logical name in the privileged section of the `tdbs.conf` configuration file to a path that points to a directory on the data volume. Do not use the default location (`TDBS_SYS`) since changes to files therein will not persist.

Create the directory and assign the group ownership to `gid 0`, give the group write permissions (`rw-rw-r-x`), then introduce the change to `tdbs.conf`. For example:

```
[Privileged]
TDBS_ACCDIR=/var/lib/trip/log/acc
```

You should also set `TDBS_ACCFLG` to control the naming and contents of the log.

Other settings

In addition to locations for databases the accounting log, there are several other kinds logical names that you may be using that can be added to the `tdbs.conf` configuration file in your Docker container. For example:

- LDAP authentication
- Login tickets
- Event logging

Step 5: Adapt user databases to the 4096-byte block size

The Docker image is preconfigured to use the `TDBS_BLOCK_SIZE` configuration setting with a value of 4096. This reduces the number of disk I/O requests TRIP has to perform when accessing data and indexes. However, in order to benefit from this setting, your databases and indexes must be adapted.

If you do not adapt your databases to the 4096-byte block size, the databases can still be used in the container, but will not be as performant due to increased disk I/O.

NOTE: once you have adapted a database to the 4096-byte block size, its files cannot safely be used with older versions of TRIP than 8.1.

Database contents (BAF)

While adapting the BAF file to the 4096-byte block size is optional, it is strongly recommended especially for large databases. To do this, you have two options. Either:

1. Run `PACKIT` on the database on a system with `TDBS_BLOCK_SIZE` set to 4096, or
2. Perform a `TFORM` export of all the database contents on the source system, then run `loadix` to import the data into your target Docker-based TRIP installation.

If you choose to run `PACKIT`, you can either do this in the Docker container (see section on using server-side tools, below), or outside the container using a non-container installation of TRIPsystem version 8.1 or later with the `TDBS_BLOCK_SIZE` logical name set to 4096. Place the resulting BAF file in its storage location directory in the data volume (e.g. in the “db” directory of the data volume if you are using the `TDBS_BASES` as storage location for the database).

If you choose to perform a `TFORM` export, copy the `TFORM` file to the data volume so that your TRIP container can access it. Then run the `loadix` tool (see section on using server-side tools, below) to import the data. **IMPORTANT:** Make sure that the BAF file for the database does not exist before running `loadix`, or the resulting database may not be configured for a 4096-byte block size after all.

When done, you can verify that the database is indeed using the 4096-byte block size by running the `tracer` tool on the database in the Docker container. For example:

```
**** TRIP System Utility BAF TRACER - Examine BAF structures ****
      Version 8.1-0:1    21-Apr-2021 07:44:48

Username   : TRIPDBA
Password   :
Database   : MYDATABASE
Blocksize  : 4096
```

```
[R]ecord, [B]lock, All re[C]ords, A[L]l blocks, Quit? [Q] : Q
```

```
Elapsed: 00:00:07
```

Database indexes

To adapt the database indexes (BIF and VIF) to the 4096-byte block size, you must completely re-index your databases. This must be done with the BIF and VIF files **completely absent**. If they are present, move them to a backup location or delete them before starting the re-index run!

Before running this step, make sure the BAF file is present in the data volume. Consider to first adapt it to the 4096-byte block size as described above.

With the index files for the database fully absent, issue the following call in the docker container (replacing MYDATABASE with the name of your database):

```
index --reindex -d MYDATABASE
```

When done, you can verify that the indexes are indeed using the 4096-byte block size by running the `exif` tool on the database in the Docker container. For example:

```
*** TRIP System Utility EXIF - BIF/VIF examination ***
    Version 8.1-0:1    21-Apr-2021 07:49:09

BIF or VIF file name           : TDBS_BASES:MYDATABASE.BIF

Block size   : 4096
Entry width  : 15

...
```

Using server-side tools

TRIP comes with several server-side command line tools that are both useful and necessary when operating a TRIP installation.

The easiest way to utilize those tools is to first log in to the Docker container:

```
docker exec -it CONTAINERNAME -u trip /bin/bash
```

You can also use SSH to access the container (see section “Optional SSH Daemon” above for more details).

This will land you in a bash shell as the user `trip`. From here you have access to all the regular tools such as `load`, `index`, `packit`, `trip`, etc.

Alternatively, you can run the tool you wish to run directly instead of going via the bash shell. Note that you will have to specify the fully qualified path to the tool. For example (the line break is for formatting purposes in this document only):

```
docker exec -it CONTAINERNAME -u trip  
/opt/trip/sys/v824/bin/index -d MYDB
```

Some tools (e.g. the `vi` editor) may be sensitive to the configured dimensions of the console window and can behave erratically if those dimensions are not properly set. To make sure these are behaving properly you should define the `COLUMNS` and `LINES` variables such that they fit the dimensions of the terminal you are using. For example, when used from a Linux host:

```
docker exec -it CONTAINERNAME -u trip \  
-e COLUMNS=`tput cols` -e LINES=`tput lines` \  
/bin/bash
```

Extending the Image

A common way to add custom behavior and functionality to a Docker container is to extend the image it is based on, effectively creating a new image. This section describes what to keep in mind when doing that with the TRIP Docker image.

NOTE: Consult with your TRIP distributor before taking any such customized image into production use, as modifications to the TRIP operating environment may affect your support and warranty.

The Dockerfile

Initialization scripts and the entry point

If you have special needs for your custom image that cannot be realized in the Dockerfile alone, you may be tempted to create a custom entry point. If you do that, you will disrupt the TRIP installation in the container and thereby render it inoperable. So, don't do that!

Instead of overriding the entry point, you should add customization initialization scripts to the folder `/etc/trip-extended-image.d` when declaring your image using your Dockerfile. These scripts will be executed near the end of the normal entry point, but before the entry point execs into the supervisord process that monitors the TRIP background processes (tripd, tripnetd, etc).

The scripts you place in the `/etc/trip-extended-image.d` folder must have the suffix `.sh` and be marked as executable. If you have multiple scripts (or if the TRIP image has been extended through multiple steps), you are invited to name them so that their execution order is predictable. Do this by adding a two-digit number at the front of the script name (e.g. 10-first.sh, 20-second.sh, etc).

For example, in your Dockerfile:

```
RUN mkdir -p /etc/trip-extended-image.d
ADD my-entrypoint.sh /etc/trip-extended-image.d
RUN chmod +x /etc/trip-extended-image.d/my-entrypoint.sh
```

Signal success by having the script return 0 on exit. Any other exit value will signal to the entry point that the script has failed, which will also cause the container startup to fail.

Note that scripts you place in `/etc/trip-extended-image.d` must not require root privileges. Any actions that require root access must be done in the Dockerfile.

Editing tdb.conf

Consider calling the `tdbsconf` command line tool if you need to add or update custom logical names in the `tdbs.conf` configuration file. This simplifies the task of editing the `tdbs.conf` while providing guardrails against common mistakes. Refer to the TdbsConf manual in the TRIPsystem documentation set for more information on this utility program.

Background processes and controlled startup and shutdown

If you have background processes that at some point manipulates persistent state, you are strongly recommended to integrate such programs in a way so that their shutdown is controlled, rather than just being killed abruptly when the container is stopped. Leaving such processes to be killed at container stop may cause corruption to the persistent state that you are keeping.

The TRIP Docker image utilizes supervisord to control the startup and shutdown of such processes. The Dockerfile for your extended Docker image can instruct supervisord to include your services by

placing “.conf” files containing supervisord-formatted “program”-sections in the directory “/etc/supervisor-ext.d”.

For example, by placing the configuration for your service in “my_service.conf”, you could have the following statement in your Dockerfile:

```
ADD my_service.conf /etc/supervisor-ext.d
```

While the term “background process” is used here, the process you start must actually run as a console application, and not in the background by turning itself into a daemon. If the program is not running in the foreground, supervisord will not be able to properly monitor it. Your service must run as a normal console program in a terminal shell without putting itself into the background.

The my_service.conf file is in this example expected to be a valid program section for the supervisord.conf file. For example:

```
[program:myservice]
command=/opt/ext/myservice/bin/myservice
redirect_stderr=true
directory=/var/lib/trip
autostart=true
autorestart=true
numprocs=1
startsecs=3
stopsignal=TERM
stdout_logfile=/var/lib/trip/log/myservice.log
stdout_logfile_maxbytes=50MB
stdout_logfile_backups=10
stdout_loglevel=info
```

Example: Adding an ASE library

A simple way to add an ASE library without building a new image is to copy the ASE library to the data volume and edit the tdb.conf file. While this may be OK for simple libraries that are unlikely to be affected by upgrading the software, there are other reasons as to why this kind of approach is less suitable. To that end, this section describes how to create an extended image that contains an ASE library with its associated files and how to configure the TRIP installation to suit.

Dockerfile Example

```
FROM registry.example.com:5050/trip/tripserver-rhel9:8.4
LABEL Description="TRIP NoSQL server with custom ASE library"
USER 0
RUN mkdir -p /opt/example/aselib && \
    mkdir -p /etc/trip-extended-image.d
ADD myase.so /opt/example/aselib
ADD myase-init.sh /etc/trip-extended-image.d
RUN chown -R 1001:0 /opt/example/aselib && \
    chmod +x /opt/example/aselib/*.so && \
    chmod +x /etc/trip-extended-image.d/myase-init.sh
USER 1001
```

Initialization Script Example

The initialization script (referred to in the example Dockerfile as `myase-init.sh`) runs when the TRIP container starts up. In this example, the script must modify the TRIPsystem `tdbs.conf` file to insert a logical name to the SO library of the ASE, and to modify the value of the `TDBS_ASELIBS` list. This cannot be done in the Dockerfile since any necessary `tdbs.conf` adjustments (including creating it on first startup) that is performed by TRIP itself is done at container start.

```
#!/bin/bash
add_to_tdbas_ase_libs()
{
    if [ "$2" = "" ]; then
        echo $1
    else
        echo $1 | grep $2 2> /dev/null 1> /dev/null
        if [ $? -eq 0 ]; then
            echo $1
        else
            if [ "" = "$1" ]; then
                echo $2
            else
                echo $2,$1
            fi
        fi
    fi
}
TMPCONF=`mktemp`
cat /var/lib/trip/conf/sys/tdbs.conf | while read -r LINE; do
    TRIMLINE="$(echo "$LINE" | sed -e 's/^[[:space:]]*//' -e 's/[[:space:]]*$//')"
    if [ "$TRIMLINE" = "" ]; then
        echo "$LINE" >> $TMPCONF
        continue
    fi
    if [ "$TRIMLINE" = \#* ]; then
        echo "$LINE" >> $TMPCONF
        continue
    fi
    SYM_NAME=`echo "$TRIMLINE" | cut -d= -f1`
    SYM_VALUE=`echo "$TRIMLINE" | cut -d= -f2- | sed -e 's/^[[:space:]]*//`
    if [ "$SYM_NAME" = "MYASE_PATH" ]; then
        echo "MYASE_PATH=/opt/example/ase/lib/myase.so" >> $TMPCONF
        continue
    fi
    if [ "$SYM_NAME" = "TDBS_ASELIBS" ]; then
        ASELIBS=`add_to_tdbas_ase_libs $SYM_VALUE MYASE_PATH`
        echo "TDBS_ASELIBS=$ASELIBS" >> $TMPCONF
        continue
    fi
    echo "$LINE" >> $TMPCONF
done
```

```
grep -q ^MYASE_PATH $TMPCONF 2> /dev/null
if [ $? -ne 0 ]; then
    echo "MYASE_PATH=/opt/example/aselib/myase.so" >> $TMPCONF
fi
grep -q ^ TDBS_ASELIBS $TMPCONF 2> /dev/null
if [ $? -ne 0 ]; then
    echo "TDBS_ASELIBS=MYASE_PATH" >> $TMPCONF
fi
cp /var/lib/trip/conf/sys/tdbs.conf /var/lib/trip/conf/sys/tdbs.conf.myase
mv $TMPCONF /var/lib/trip/conf/sys/tdbs.conf
```

