DIRBS Core Release 16.0.0

Installation Guide

80-GD079-2 Rev. L

15/03/2021

Revision History

|  |  |  |
| --- | --- | --- |
| Revision | Date | Description |
| A | November 2017 | Initial release |
| B | April 2018 | Updates for DIRBS Core 7.0.0 Release |
| C | May 2018 | Updates for DIRBS Core 8.0.0 Release |
| D | July 2018 | Updates for DIRBS Core 8.0.1 Release |
| E | September 2018 | Updates for DIRBS Core 9.0.0 Release |
| F | March 2019 | Updates for DIRBS Core 10.0.0 Release |
| G | October 2019 | Updates for DIRBS Core 11.0.0 Release |
| H | June 2020 | Updates for DIRBS Core 12.0.0 Release |
| I | September 2020 | Updates for DIRBS Core 13.0.0 Release |
| J | November 2020 | Updates for DIRBS Core 14.0.0 Release |
| K | February 2021 | Updates for DIRBS Core 15.0.0 Release |
| L | March 2021 | Updates for DIRBS Core 16.0.0 Release |

Contents

[1 Introduction 4](#_Toc52530701)

[1.1 Purpose 4](#_Toc52530702)

[1.2 Getting started 4](#_Toc52530703)

[2 Installation 5](#_Toc52530704)

[2.1 Extracting software release 5](#_Toc52530705)

[2.2 Docker installation 5](#_Toc52530706)

[2.2.1 Building Docker images 5](#_Toc52530707)

[2.2.2 Running Docker images 5](#_Toc52530708)

[2.3 Advanced topics 7](#_Toc52530709)

[2.3.1 API and Data Processing blades 7](#_Toc52530710)

[2.3.2 Operator Upload and Data Processing blades 9](#_Toc52530711)

[2.4 Database 9](#_Toc52530712)

[2.4.1 Installation 9](#_Toc52530713)

[2.4.2 Upgrading the database schema 12](#_Toc52530714)

[3 Configuration 13](#_Toc52530715)

[4 Installation Verification 14](#_Toc52530716)

[4.1 Running the automated regression test suite 14](#_Toc52530717)

[4.2 Checking status of the database version 15](#_Toc52530718)

Tables

Table 2-1 Roles and functions........................................................................................................................................11

# Introduction

## Purpose

This document provides:

* Installation instructions for the Device, Identification, Registration, and Blocking System (DIRBS) Core 16.0.0 software
* Instructions for running automated tests to verify DIRBS Core installation

## Getting started

The instructions provided in this document assume that the recommended equipment has been installed and configured with Ubuntu 18.04. Refer to the [Ubuntu Installation Guide](https://help.ubuntu.com/lts/installation-guide/i386/install.en.pdf) for additional installation help.

The installer should be familiar with Linux command line.

# Installation

This installation guide builds a new Docker image by downloading a base Ubuntu 18.04 image and downloading and installing required third party software modules into the image as a convenience to the end user.

1. The reader acknowledges and agrees that it is entirely and solely responsible for the selection and use of all third-party software modules downloaded and installed by this installation method, including securing all appropriate and proper rights of use to any of such third-party software modules and to comply fully with any terms of use that may apply to or accompany any such third-party software modules.

Qualcomm Technologies, Inc. does not undertake any obligations, duties, or other responsibilities in connection with the selection or use by the reader of any of such third-party software modules.

## Extracting software release

The DIRBS software release is distributed as a tar.gz file. To extract the contents of the distribution, run:

tar xvzf dirbs-core-16.0.0.tar.gz

## Docker installation

Docker installation can be run on any platform Docker supports.

For instructions on installing Docker on a specific host platform, see <https://docs.docker.com/engine/installation>

### Building Docker images

If you are in the root folder for the distributables, run the following command to build all sample Docker images:

make -f docker/prd/Makefile

### Running Docker images

This section shows basic commands that run each type of container, assuming the config file contained all settings at the time the config file was built.

#### Data Processing blade image

After Docker images have been built, start the Data Processing blade image by running:

docker run --rm --tmpfs /tmp -e DIRBS\_OPERATORS=<operator\_list> -p 2222:22 \dirbs-processing

This docker run command uses the following standard Docker options:

* –-rm: Instructs Docker to remove the container from the system once it has been stopped.
* --tmpfs: Mounts a temporary filesystem (tmpfs) to /tmp in the container. Used during the lifetime of the container.
* –e: Sets an environment variable inside the container.
* -p 2222:22: Maps the SSH port (22) inside the container to port 2222 on the host system.

#### Operator Upload blade image

After Docker images have been built, start the Operator Upload blade image by running

docker run --rm --tmpfs /tmp -e DIRBS\_OPERATORS=<operator\_list> -p 2222:22 dirbs-upload

This docker run command uses the following standard Docker options:

* –-rm: Instructs Docker to remove the container from the system once it has been stopped.
* --tmpfs: Mounts a temporary filesystem (tmpfs) to /tmp in the container. Used during the lifetime of the container.
* –e: Sets an environment variable inside the container.
* -p 2222:22: Maps SSH port (22) inside the container to port 2222 on the host system.

<operator\_list> should be a comma-separated list of operator IDs that match the list of operators configured in the config file.

#### API blade image

After Docker images have been built, start the API blade image by running

docker run --sysctl net.core.somaxconn=4096 --rm --tmpfs /tmp -p 5000:5000 dirbs-api

* –sysctl: Modifies the kernel settings on the blade to specify the number of connections that can be waiting in a queue on an individual socket.
* –-rm: Instructs Docker to remove the container from the system once it has been stopped.
* --tmpfs: Mounts a temporary filesystem (tmpfs) to /tmp in the container. Used during the lifetime of the container.
* -p 5000:5000: Maps webserver port (5000) inside the container to port 5000 on the host system.

#### 2.2.2.4 Whitelist blade images

In DIRBS Core v13.0.0, whitelisting capabilities are added. For this feature to work, the whitelist blade images need to be configured. DIRBS Core provides two type of images i.e.

* dirbs-wl-processor: Used for whitelist processing
* dirbs-wl-distributor: Used for the distribution of whitelist across multiple operators.

**Note:** Before deploying Whitelist containers, Apache KAFKA needs to be deployed. An example installation guide is [here](https://www.digitalocean.com/community/tutorials/how-to-install-apache-kafka-on-ubuntu-18-04)

1. After docker images have been built, start the whitelist processing image by running:

docker run **-e DIRBS\_KAFKA\_HOST=<KAFKA HOST NAME>**

**-e DIRBS\_KAFKA\_PORT=<KAFKA PORT> -e DIRBS\_KAFKA\_TOPIC=<KAFKA TOPIC>**

dirbs-wl-processor:latest

2. To start the whitelist distribution image, use:

docker run **-e DIRBS\_KAFKA\_HOST=<KAFKA HOST NAME>**

**-e DIRBS\_KAFKA\_PORT=<KAFKA PORT> -e DIRBS\_KAFKA\_TOPIC=<KAFKA TOPIC>**

dirbs-wl-distributor:latest

For a successful run of whitelist blade images, following actions need to be performed as well:

* In the config file under “**operational”** config section, set **activate\_whitelist** to **true**
* Perform all the steps involving database creation and migration as mentioned in the official installation guide after above step. This will automatically create migrations required for whitelist images in the database.
* Perform the necessary changes in **Broker** config section as and when required.

## Advanced topics

### API and Data Processing blades

#### Mounting a config file inside a container

To mount a config file from the host into the Data Processing and API blade containers so that DIRBS configuration can be modified after build time, use the following option to the docker run command for those images:

-v <abs\_path\_to\_config\_on\_host.yml>:/home/dirbs/.dirbs.yml:ro

This option mounts the file located at abs\_path\_to\_config\_on\_host.yml to /home/dirbs/.dirbs.yml inside the container. This will be read-only inside the container. Changes made on the host will be reflected inside the container.

#### Mounting a .pgpass file inside a container

While database credentials can be edited directly inside the /home/dirbs/.dirbs.yml file, a .pgpass file (standard PostgreSQL connection string file) can be mounted inside the container as well.

Advantages are:

* The person responsible for editing the config YML does not automatically get access to DB credentials.
* .pgpass file can configure a password for multiple users, not just one.

Use the following option to the docker run command for the Data Processing and API blade containers:

-v <abs\_pgpass\_file\_on\_host>:/home/dirbs/.pgpass:ro

This option mounts the file located at abs\_pgpass\_file\_on\_host.yml to /home/dirbs/.pgpass inside the container. This will be read-only inside the container.

1. .pgpass file will be ignored if UNIX file permissions are not stricter than 0700.

#### Passing in environment variables for StatsD and PostgreSQL connection

If set, the following environment variables override configuration file settings. If there is a command-line argument specified, this continues to override environment variables:

* DIRBS\_DB\_HOST: Host that PostgreSQL database runs on (default: localhost)
* DIRBS\_DB\_PORT: Port that PostgreSQL database runs on (default: 5432)
* DIRBS\_DB\_DATABASE: PostgreSQL database name to connect to
* DIRBS\_DB\_USER: PostgreSQL user to connect as
* DIRBS\_DB\_PASSWORD: PostgreSQL password for DIRBS\_DB\_USER
* DIRBS\_STATSD\_HOST: Host that StatsD run on (default: localhost)
* DIRBS\_STATSD\_PORT: Port that StatsD listens on (default: 8125)
* DIRBS\_ENV: Unique environment string used by StatsD to distinguish between different hosts or environments when sending metrics

The values of these environment variables can be passed through to Docker containers in two ways:

* Populate a file with a list of the above KEY=VALUE lines
  + This file can then be passed to the docker run command via the --env-file command-line option
* Specify each key-value pair on the docker run command-line using the -e command-line option

#### Mounting a persistent log directory

To ensure that logs are persisted between container restarts, the following CLI option should be provided to docker run:

-v <abs\_log\_dir\_on\_host>:/var/log/dirbs

This option mounts the host directory located at abs\_log\_dir\_on\_host to /var/log/dirbs in the container.

### Operator Upload and Data Processing blades

#### Mounting a persistent data directory

For Operator Upload and Data Processing blades, the data directory should be persistent between container restarts.

To ensure this is the case, a host directory can be mounted into the container using the following CLI option to docker run:

-v <abs\_data\_dir\_on\_host>:/data

This option mounts the host directory located at abs\_data\_dir\_on\_host to /data in the container.

## Database

### Installation

1. Creating a new database from scratch assumes that you are already running a PostgreSQL instance.
2. Install PostgreSQL HLL extension

DIRBS Core requires the [PostgreSQL HLL extension](https://github.com/citusdata/postgresql-hll) to function.

If using our provided Docker image for PostgreSQL, it is already installed.

If on RDS in AWS, this extension should also be optionally available.

Otherwise, you must build and install the extension (see [README](https://github.com/citusdata/postgresql-hll) in the GitHub repo).

1. Create a PostgreSQL role with superuser privileges

If you are using the provided Docker image for PostgreSQL, the environment variables DB\_ROOT\_USER and DB\_ROOT\_PASSWORD can be used the first time the container is started to automatically create a superuser account.

If you would rather not supply these environment variables, are not using our Docker image, or have an existing PostgreSQL database server, the following SQL command can be run as a superuser to create an appropriate role:

CREATE ROLE <username> WITH SUPERUSER LOGIN ENCRYPTED PASSWORD '<password>';

This account should only be used to create databases and new roles. It is necessary to be a superuser account as the HLL extension cannot be created in a new database by a non-superuser. It should **not** be used as a general account given the privileges granted to it.

1. Install base roles

After creating a superuser, create the base roles that DIRBS requires. These are all marked NOLOGIN, meaning it is not possible to login as these roles – they are just abstract roles that can be GRANT'ed to real users with LOGIN privilege.

These roles must exist before the database can be created or installed.

* 1. Run the following command on the DIRBS processing blade:

dirbs-db --db-user <username> --db-password-prompt install\_roles

where <username> is the name of the superuser created in Step 2.

You will be prompted for your password.

Since this is a high privilege database account and this is a one-off command, it is more secure to enter the password manually via the prompt than to store it in a config file.

1. Create an empty database

Ownership of the database is important as we must ensure that **every** power user has rights to do everything, rather than just the user creating the DB.

To create an empty database owned by dirbs\_core\_power\_user role:

* 1. Use the psql command to login to the postgres database on the PostgreSQL server using the superuser role created in Step 2.
  2. Run the following SQL command to create the empty database:

CREATE DATABASE <database\_name> OWNER dirbs\_core\_power\_user;

* 1. Connect to the new database using the following command in psql:

\c <database\_name>

* 1. Run the following SQL commands to install the HLL extension in the new database:

CREATE SCHEMA hll;

GRANT USAGE ON SCHEMA hll TO dirbs\_core\_base;

CREATE EXTENSION hll SCHEMA hll;

1. Create user accounts

With an empty database, we can create user accounts while still logged into the PostgreSQL database as the superuser. This can also be done later.

* 1. Create an initial power user (can do everything):

CREATE USER <username> WITH LOGIN ENCRYPTED PASSWORD '<password>' IN ROLE dirbs\_core\_power\_user;

Optionally, extra users with lower privileges can be created to perform specific tasks with minimum required privileges:

CREATE USER <username> WITH LOGIN ENCRYPTED PASSWORD '<password>' IN ROLE <role\_name>;

Roles can be GRANT'ed or REVOKE'd after user creation:

GRANT <role\_name> TO <username>;

REVOKE <role\_name> FROM <username>;

Table 2-1 provides a complete list of roles and their functions.

Table 2-1 Roles and functions

|  |  |
| --- | --- |
| Role | Function |
| dirbs\_core\_api | Runs REST-ful APIs |
| dirbs\_core\_catalog | Runs dirbs-catalog |
| dirbs\_core\_classify | Runs dirbs-classify |
| dirbs\_core\_import\_golden\_list | Runs dirbs-import golden\_list |
| dirbs\_core\_import\_gsma | Runs dirbs-import gsma\_tac |
| dirbs\_core\_import\_operator | Runs dirbs-import operator |
| dirbs\_core\_import\_pairing\_list | Runs dirbs-import pairing\_list |
| dirbs\_core\_import\_registration\_list | Runs dirbs-import registration\_list |
| dirbs\_core\_import\_barred\_list | Runs dirbs-import barred\_list |
| dirbs\_core\_import\_barred\_tac\_list | Runs dirbs-import barred\_tac\_list |
| dirbs\_core\_import\_subscribers\_registration\_list | Runs dirbs-import subscribers\_registration\_list |
| dirbs\_core\_import\_stolen\_list | Runs dirbs-import stolen\_list |
| dirbs\_core\_listgen | Runs dirbs-listgen |
| dirbs\_core\_poweruser | Can do everything (only user that can do dirbs-db and dirbs-prune) |
| dirbs\_core\_report | Runs dirbs-report |
| dirbs\_core\_import\_device\_association\_list | Runs dirbs-import association\_list |
| dirbs\_core\_import\_monitoring\_list | Runs dirbs-import monitoring\_list |
| dirbs\_core\_whitelist | Runs dirbs whitelist processes |

A user can be in multiple roles. To see a list of which users are on the system and which roles they have, we recommend using the psql command \du to view the list of database users.

1. Install a schema

Once user accounts have been created and the connection details configured for DIRBS, install the DB schema:

dirbs-db --db-user <username\_of\_power\_user> --db-password-prompt install

1. This DB user should be the power user created in Step 5a and not the superuser created in Step 2 for security reasons. We do not recommend storing passwords for power users in config files or environment variables for additional security.
2. Choose a number of physical shards

Many DIRBS Core tables are split into multiple physical partitions based on their IMEI. The default number of partitions is 4. For larger deployments, it is recommended to increase the number of partitions to achieve higher potential parallelism and performance.

This is done via the command:

dirbs-db --db-user <username\_of\_power\_user> --db-password-prompt repartition --num-physical-shards <num\_shards>

This can be done at any time, but we recommend initially avoiding a lengthy re-write of all existing tables later.

### Upgrading the database schema

To upgrade a schema from a previous release, run:

dirbs-db --db-user <super\_user> --db-password-prompt install\_roles

dirbs-db --db-user <power\_user> --db-password-prompt upgrade

where

* <super\_user> is the superuser (see Step 2 of Section ‎2.4.1)
* <power\_user> is the power user (see Step 5a of Section ‎2.4.1)

# Configuration

An annotated config file etc/config.yml was distributed with this release. For information on the available settings and what they mean, refer to the comments in the annotated sample config file.

This annotated config file is automatically copied into built Docker images and is in both files:

* /opt/dirbs/etc/config.yml
* /home/dirbs/.dirbs.yml

When DIRBS Core software runs, it looks for a config file in ~/.dirbs.yml. If that file is not found, it looks in the system location /opt/dirbs/etc/config.yml.

Once the Docker image has been built, the configuration can be modified by mounting a config file as a volume to ~/.dirbs.yml to override the config file that the Docker image was originally built with (see Section ‎2.3.1.1).

# Installation Verification

## Running the automated regression test suite

If you have built the Docker processing blade image, use the following command to run the automated regression test suite:

docker run --rm --tmpfs /tmp dirbs-processing:${BUILD\_TAG} gosu dirbs py.test -v /home/dirbs/tests/\*.py

Alternatively, if a container based off the dirbs-processing image is already running, you can SSH into it to run the tests. Once you are SSH’ed in, use the following command to run tests:

py.test -v /home/dirbs/tests/\*.py

(dirbs-venv) dirbs@8d98750dc5aa:~$ py.test -v /home/dirbs/tests/\*.py

========================================================================== test session starts ===========================================================================

platform linux -- Python 3.5.2, pytest-3.0.0, py-1.4.34, pluggy-0.3.1 -- /home/dirbs/dirbs-venv/bin/python3.5

cachedir: tests/.cache

rootdir: /home/dirbs/tests, inifile: pytest.ini

plugins: mock-1.6.0

collected 700 items

tests/app\_logging.py::test\_exception\_logging\_api PASSED

tests/app\_logging.py::test\_exception\_logging\_cli PASSED

…

…

…

tests/tac\_api.py::test\_method\_post\_not\_allowed PASSED

tests/tac\_api.py::test\_method\_delete\_not\_allowed PASSED

====================================================================== 266 passed in 247.90 seconds ======================================================================

(dirbs-venv) dirbs@8d98750dc5aa:~$

A successful install is one where all run test cases have passed.

## Checking status of the database version

Verify that the DIRBS Code schema version is equivalent to the DB schema version:

dirbs-db check

2018-09-30 00:38:56,683 - dirbs.db - INFO - Querying DB schema version for DB dirbs on host <DB\_HOST>

2018-09-30 00:38:56,706 - dirbs.db - INFO - Code schema version: 86

2018-09-30 00:38:56,706 - dirbs.db - INFO - DB schema version: 86

2018-09-30 00:38:56,707 - dirbs.db - INFO - Schema versions match between code and DB.