Device Registration Subsystem 2.0.0

API Installation Guide

DRS-API-Installation-Guide-2.0.0

March 16, 2019

**Revision history**

| **Revision** | **Date** | **Description** |
| --- | --- | --- |
| A |  | Release 2.0.0 |

Contents

[1 Introduction 4](#_Toc527742167)

[1.1 Purpose & Scope 4](#_Toc527742168)

[1.2 Definitions, Acronyms & Abbreviations 4](#_Toc527742169)

[1.3 References 4](#_Toc527742170)

[1.4 Getting Started 4](#_Toc527742171)

[2 Installation 5](#_Toc527742172)

[2.1 System Requirements 5](#_Toc527742173)

[2.1.1 Software Requirements 5](#_Toc527742174)

[2.1.2 Hardware Requirements 5](#_Toc527742175)

[2.1.3 Operating System 5](#_Toc527742176)

[2.1.4 Database Support 5](#_Toc527742177)

[2.2 Extracting Software Release 6](#_Toc527742178)

[2.3 Manual Installation 6](#_Toc527742179)

[3 Configuration 7](#_Toc527742180)

[3.1 Nginx Configuration 7](#_Toc527742181)

[3.2 uWSGI Configuration 8](#_Toc527742182)

[3.3 uWSGI Service Configuration 9](#_Toc527742183)

[3.4 DRS Configuration and Initialization 9](#_Toc527742184)

[4 Testing 11](#_Toc527742185)

**Tables**

[Table 1- Definitions, Acronyms & Abbreviations 4](#_Toc527623685)

# Introduction

## Purpose & Scope

This document provides:

* Installation instructions for the Device Registration Subsystem
* Instructions for running test commands to verify DRS API installation

## Definitions, Acronyms & Abbreviations

**Table 1- Definitions, Acronyms & Abbreviations**

| Term | Explanation |
| --- | --- |
| DIRBS | Device Identification, Registration & Blocking System |
| DRS | Device Registration Subsystem |
| OS | Operating System |
| PostgreSQL | PostgreSQL open source object-relational database system |
| Nginx | An open source, lightweight, high-performance web server or proxy server |
| uWSGI | uWSGI is used for serving Python web applications |
| Rabbit MQ server | RabbitMQ is an open source server and is built on the [Open Telecom Platform](https://en.wikipedia.org/wiki/Open_Telecom_Platform) framework for clustering and failover |
| Celery | Celery is an open source asynchronous task queue or job queue which is based on distributed message passing. While it supports scheduling, its focus is on operations in real time. |

## References

N.A

## Getting Started

The instructions provided in this document assume that the required equipment (hardware, software) has been installed and configured with Ubuntu 16.04. 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

1. The reader acknowledges and agrees that he 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.

## System Requirements

### Software Requirements

* Python 3.X
* Ubuntu 16.0
* PostgreSQL 10
* Nginx 1.14.X
* uWSGI 2.0
* Celery 4.2.0
* Rabbitmq 3.6.10

### Hardware Requirements

Minimum hardware requirements

* At least 512 MB of RAM
* At least 1G of disk space

### Operating System

This subsystem will be installed and configured with Ubuntu 16.04. Refer to the [Ubuntu Installation Guide](https://help.ubuntu.com/lts/installation-guide/i386/install.en.pdf) for additional installation help.

### Database Support

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

A complete guide for PostgreSQL installation and configuration can be found on [PostgreSQL website](https://www.postgresql.org/)

## Extracting Software Release

The DRS software release is distributed as a tar.gz file. To extract the contents of the  
distribution, run:  
tar xvzf dirbs-drs-api-2.0.0.tar.gz

Copy the contents to the web root directory e.g. /var/www/html (default Nginx web root directory)

## Manual Installation

* Ensure the APT package index is updated

apt-get update --fix-missing

* Install basic required packages

apt-get install nginx git python3 virtualenv libpython3-dev python3-pip python3-dev rabbitmq-server

* Go to path /var/www/html/dirbs-drs-api-2.0.0

pushd /var/www/html/dirbs-drs-api-2.0.0

* Create virtual environment and install requirements

virtualenv –p python3 venv

source venv/bin/activate

pip3 install -r requirements.txt

* Nginx does not support python application so we need to install uWSGI to run python application through Nginx, below is the command to install uWSGI

pip3 install uwsgi

deactivate

Follow below steps for celery configuration

* Start the Workers as Daemons so that they are started automatically at server startup
* Create a new service definition file in /etc/systemd/system/celeryd.service. Change the “User” and “Group” properties according to your actual user and group name
* For our setup we have created a user celery using below command

adduser celery

|  |
| --- |
| [Unit]  Description=Celery Service  After=network.target  [Service]  Type=forking  User=**celery**  Group=**celery**  EnvironmentFile=/etc/default/celeryd  WorkingDirectory=/var/www/html/dirbs-drs-api-2.0.0/  ExecStart=/bin/sh -c '${CELERY\_BIN} multi start ${CELERYD\_NODES} -B\  -A ${CELERY\_APP} --pidfile=${CELERYD\_PID\_FILE} \  --logfile=${CELERYD\_LOG\_FILE} --loglevel=${CELERYD\_LOG\_LEVEL} ${CELERYD\_OPTS}'  ExecStop=/bin/sh -c '${CELERY\_BIN} multi stopwait ${CELERYD\_NODES} \  --pidfile=${CELERYD\_PID\_FILE}'  ExecReload=/bin/sh -c '${CELERY\_BIN} multi restart ${CELERYD\_NODES} -B \  -A ${CELERY\_APP} --pidfile=${CELERYD\_PID\_FILE} \  --logfile=${CELERYD\_LOG\_FILE} --loglevel=${CELERYD\_LOG\_LEVEL} ${CELERYD\_OPTS}'  [Install]  WantedBy=multi-user.target |

* Create a configuration file “celeryd” in /etc/default/ directory

|  |
| --- |
| #The name of the workers. This example will create two workers  CELERYD\_NODES=”worker1 worker2”  # The name of the Celery App, should be the same as the python file  # where the Celery tasks are defined  CELERY\_APP=”app.celery”  # log and PID directories  CELERYD\_LOG\_FILE=”/var/log/celery/%n%I.log”  CELERYD\_PID\_FILE=”/var/run/celery/%n.pid”  #log level  CELERYD\_LOG\_LEVEL=INFO  #Path to celery binary, that is in your virtual environment  CELERY\_BIN=/var/www/html/dirbs-drs-api-2.0.0/bin/celery |

* Create log and pid directories

mkdir /var/log/celery /var/run/celery/

chown celery:celery /var/log/celery /var/run/celery

* Reload systemctl daemon. You should run this command each time you make any change in the service definition file.

systemctl daemon-reload

* Enable the service to startup at boot

systemctl enable celeryd

* Start the service

systemctl start celeryd

# Configuration

## Nginx Configuration

Remove Nginx default configuration and create new configuration file for the DRS app

rm /etc/nginx/sites-enabled/default

* Now create a new configuration file in the root path

nano /var/www/html/dirbs-drs-api-2.0.0/drs.conf

* Copy the below lines

|  |
| --- |
| server {  listen 80;  server\_name localhost;  charset utf-8;  client\_max\_body\_size 75M;  location / {try\_files $uri @dirbs\_intl\_drs\_api;}  location @dirbs-drs  {  include uwsgi\_params;  uwsgi\_pass unix:/var/www/html/dirbs-drs-api-2.0.0/uwsgi.sock;  }  } |

* Symlink the new created file to Nginx’s configuration files directory and restart Nginx

ln -s /var/www/html/dirbs-drs-api-2.0.0/drs.conf /etc/nginx/conf.d/

* Verify nginx configuration

nginx –t

* Restart Nginx Service

service nginx restart

## uWSGI Configuration

* Create a new configuration file in the root path and copy the below lines

nano /var/www/html/dirbs-drs-api-2.0.0/uwsgi.ini

* Add below lines in this configuration file:

|  |
| --- |
| [uwsgi]  #application's base folder  base = /var/www/html/dirbs-drs-api-2.0.0/  #python module to import  app = run  module = %(app)  chdir = %(base)  home = %(base)/venv  pythonpath = %(base)  master = true  processes = 10  cheaper = 2  cheaper-initial = 5  cheaper-step = 1  cheaper-algo = spare  cheaper-overload = 5  enable-threads = true  max-requests = 500  max-worker-lifetime = 120  ignore-sigpipe=true  ignore-write-errors=true  disable-write-exception=true  #socket file's location  socket = /var/www/html/dirbs-drs-api-2.0.0/%n.sock  #permissions for the socket file  chmod-socket = 666  chown-socket = www-data:www-data  #ownership of uwsgi service  uid = www-data  gid = www-data  #the variable that holds a flask application inside the module imported at line #6  callable = app  #location of log files  logto = /var/log/uwsgi/%n.log |

* Create a directory vassals in /etc/uwsgi/

mkdir -p /etc/uwsgi/vassals

* Create Symlink in this directory to uwsgi ini config file

ln -s /var/www/html/dirbs-drs-api-2.0.0/uwsgi.ini \

/etc/uwsgi/vassals/uwsgi.ini

* Create a new directory for log files

mkdir -p /var/log/uwsgi

* Change ownership of the web root directory and logs directory to the web-user

chown -R www-data:www-data /var/www/html/dirbs-drs-api-2.0.0/

chown -R www-data:www-data /var/log/uwsgi/

## uWSGI Service Configuration

Configure the uwsgi to run as a service on the server.

* Create an init script at location

nano /etc/systemd/system/uwsgi.service

* Copy below lines in to the script file

|  |
| --- |
| [Unit]  Description=uWSGI Emperor service  After=syslog.target  [Service]  ExecStart=/var/www/html/dirbs-drs-api-2.0.0/venv/bin/uwsgi –-emperor /etc/uwsgi/vassals/  Restart=always  KillSignal=SIGQUIT  Type=notify  StandardError=syslog  NotifyAccess=all  [Install]  WantedBy=multi-user.target |

* Reload system defaults to update the script in system services

systemctl daemon-reload

* Start uwsgi to start the application

service uwsgi start

* Go to the web-browser and enter the [URL](http://drs-server-ip/) of the server to check that the service is running

## DRS Configuration and Initialization

To configure file according to database server and credentials, edit config.ini in directory /var/www/html/dirbs-drs-api-2.0.0 and edit config.ini.

Change the hostname, port, username, password and database name as per requirements. Within the same directoy of DRS root, activate the virtual environment.

To activate the virtual environment created earlier at /var/www/html/dirbs-drs-api-2.0.0 and start database initialization.

cd /var/www/html/dirbs-drs-api-2.0.0

source venv/bin/activate

Follow below steps for database initialization and migration:

* Clean extra and un-necessary directories

make clean

* Clean python compiled files

make clean-pyc

* Initialize database

make install-db

* Upgrade database

make upgrade-db

* Compile languages

pybabel compile -d app/translations

* Configure to create distribution files

make dist

* Generate full registration list for DIRBS Core

make genlist-full

* Generate delta registration list for DIRBS Core

make genlist-delta

* Restart uWSGI service

service uwsgi restart

# Testing

* To check nginx configuration for errors

nginx –t

* To get detailed logs of uWSGI service. uWSGI can be run without service command in foreground

uwsgi --ini /var/www/dirbs-drs-api-2.0.0/uwsgi.ini