CDR-Stats Documentation

Release 3.1.1

Arezqui Belaid

June 07, 2016

Contents

Version 3.1

Release 3.1.1

Date June 07, 2016

Contents:

CHAPTER 1

Getting Started

Web http://www.cdr-stats.org/

Download http://www.cdr-stats.org/download/

Source https://github.com/cdr-stats/cdr-stats/

Keywords VoIP, Freeswitch, Asterisk, Django, Python, Call, Reporting, CDR

–

CDR-Stats is free and open source CDR (Call Detail Record) mediation, rating, analysis and reporting application for Freeswitch, Asterisk and other type of VoIP Switch. It allows you to interrogate your CDR to provide reports and statistics via a simple to use, yet powerful, web interface.

It is based on the Django Python Framework, Celery, Gevent, PostgreSQL and InfluxDB.

1.1 Overview

CDR-Stats is an application that allows rating, browsing and analysing CDR.

Different reporting tools are provided:

- Dashboard: Overview of call activity
- Search CDR: Search, filter, display and export CDR
- Overview: Analyse call traffic by hour, day and month
- Daily Comparison: Compare call traffic day on day
- Country Report: Call statistics by country
- World Map: Call statistics overlaid on a world map
- Call Cost and Carrier Costs
- Mail daily aggregated reports

- Threat Control: Detect abnormal call patterns
- Destination Alerts: Unexpected destination alerts

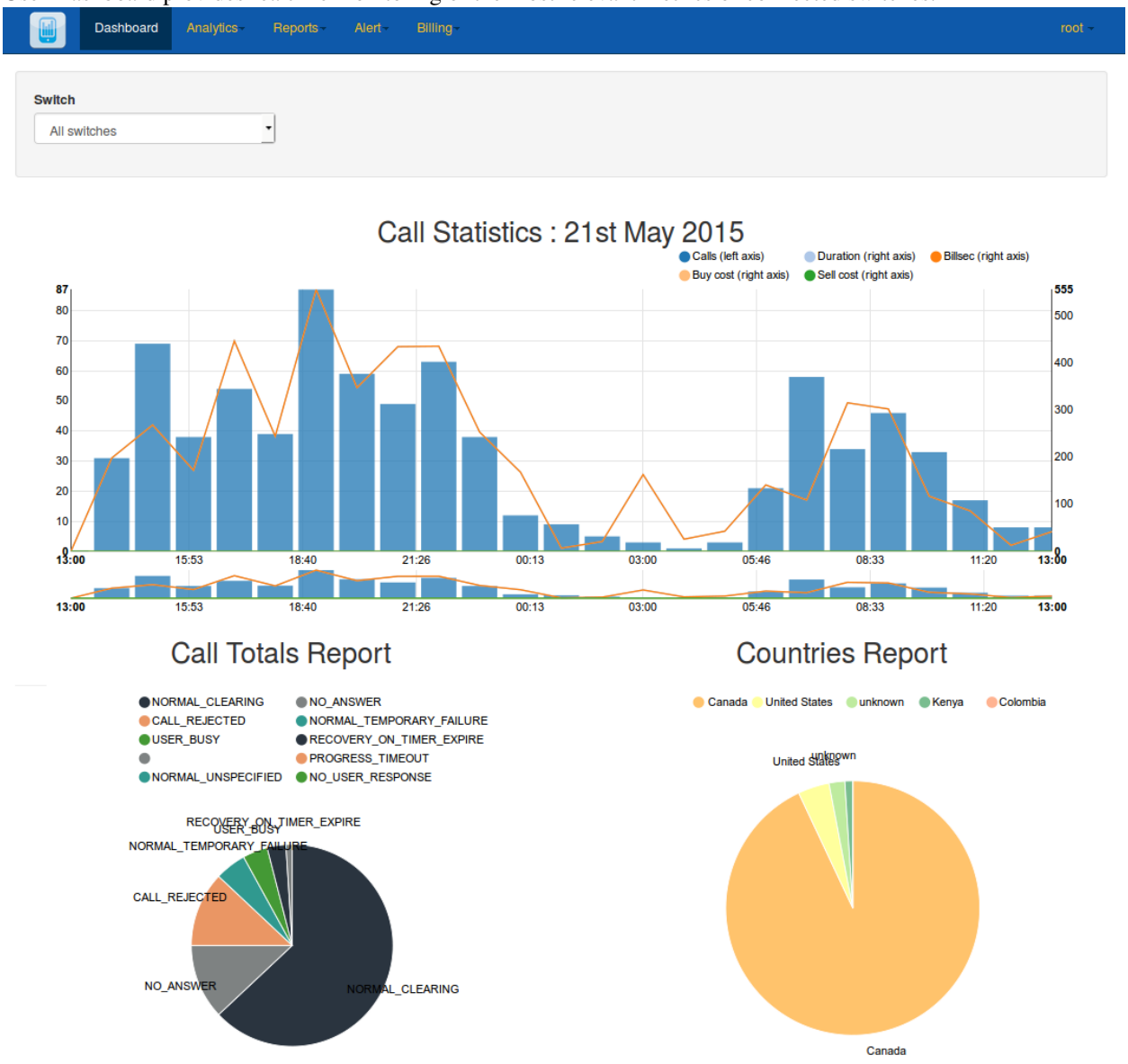
CDR Stats uses PostgreSQL, a scalable, high performance database system used to analyse large quantities of CDR data. PostgreSQL provides Materialized views which make it perfect to build analytic application which do heavy aggregation and recently PostgreSQL cames with Jsonb field which make it easy to store custom data from variety of switch.

Out of the box, CDR-Stats supports Freeswitch, Asterisk, Kamailio, SipWise, Veraz using connectors that get the CDRs and push them to centralized database. Connectors any switch systems can be built.

For list of the last supported Switches, please refere to http://www.cdr-stats.org/pricing/switch-connectors/

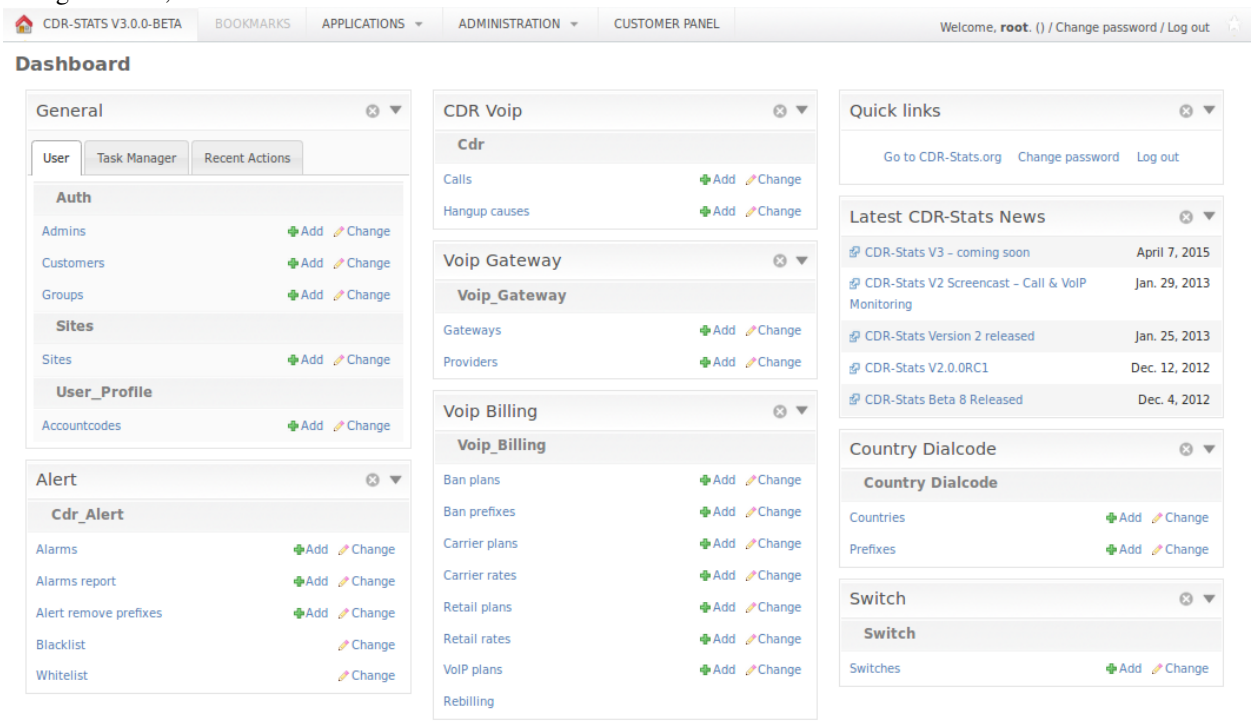
1.2 Dashboard

User Dashboard provides realtime monitoring of the most relevant metrics of connected switches.



1.3 Admin Panel

The Admin Panel allows the administrators to configure the entire reporting platform, import CDR in CSV format, configure users, switch connections and automatic alarms.

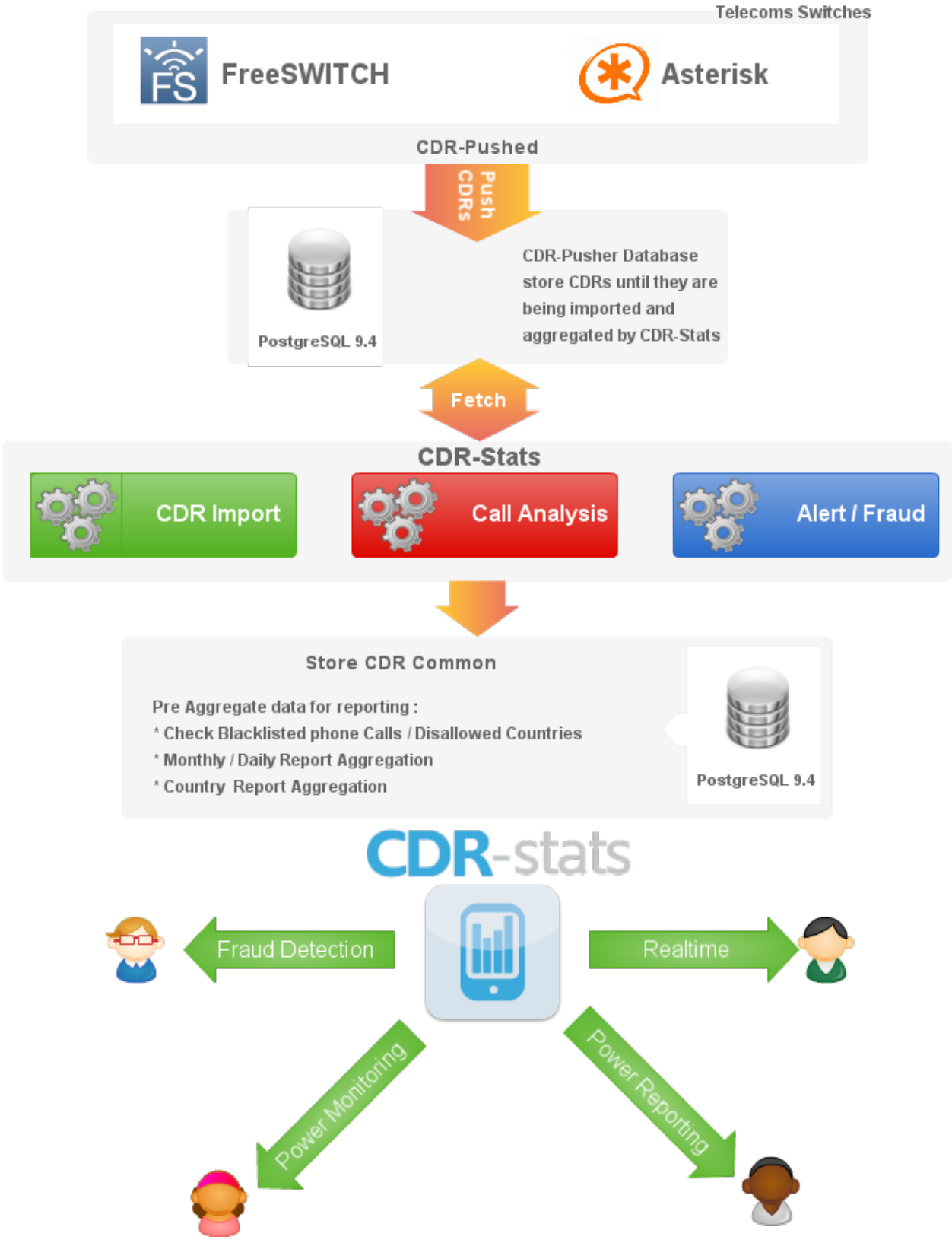


1.4 Architecture

CDR-Stats uses PostgreSQL as the underlying CDR store. PostgreSQL with Materialized view allows querying and analysis of many millions of records without noticeable loss of performance, and can easily be scaled as demand increases.

Postgresql is used for managing CDR-Stats in terms of users and managing the web framework, Django.

Celery, a task manager runs in the background, and monitors the CDR coming into the system, and alerts the systems administrator when unusual behaviour is discovered. What is determined as unusual behaviour is determined by the administrator who can configure alerts for increases in dropped calls, average length of calls, or calls to unusual destinations.



CDR-Stats works hand in hand with CDR-Pusher which has been built to create an totally independent, easy to install, high performance CDRs Collector. CDR-Pusher is installed on your local Telcoms Switch (e.g. Asterisk), the

application will harvest CDRs in Realtime and push them to a central CDR-Stats Database.

1.5 Features

Many features are provided on CDR-Stats, from browsing millions of CDRs, call rating, providing efficient search facilities to build reporting such as monthly reports and comparing call traffic with previous days.

Telephony Reporting	Leading open source switches Freeswitch, Asterisk, supported as standard.
Multi-switch	Monitor traffic from many switches in one location
Multi-tenant	Sllowing many customers to monitor their own CDR on one instance of CDR-Stats.
Distributed	Runs on one or more machines. Supports broker *clustering* and *HA*. New workers can be set up without central configuration.
Fraud detection	Visualise traffic which helps to identify unusual patterns.
Fraud Alert	Send emails to the administrator when fraud are or suspicious paterns occur
Error Emails	Can be configured to send emails to the administrator if a tasks fails.
Import CDR	Import CDR files in custom format
World Map view	See where the traffic originates and terminates on a Map
Compare traffic	See how your traffic evolves, and patterns change.
Mail Reporting	Send daily mail reports of telecoms traffic
Realtime Reporting	Traffic displayed in realtime
Blacklist	Blacklist Phone number patterns to receive alarms
Geographic alerts	Set alert if calls go to disallowed countries
Call Rating	Each call individually rated

1.6 Utility

CDR-Stats is a simple-to-use tool to provide easy rating and analysis of calls. It is a recommended addition to telephony servers, whether it be a simple in-house PBX or large capacity VoIP switch. It shows in in near realtime what calls are going through, can detect errors and failures, and alert the systems administrator is unexpected traffic is noted.

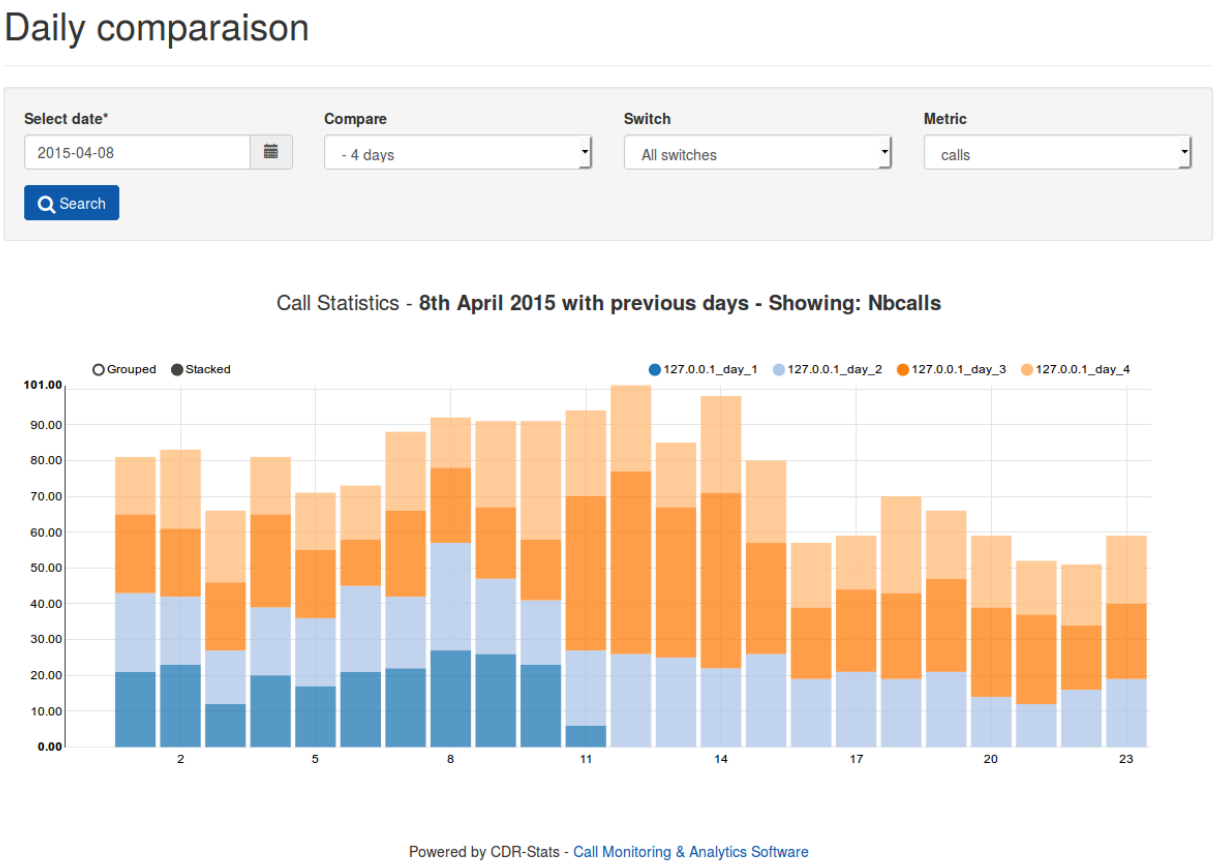
CHAPTER 2

Installation

Contents:

2.1 Overview

CDR-Stats is a web-based telecoms application for analysing, reporting and rating on CDR (Call Detail Records) for multiple tenants delivered from Asterisk, Freeswitch and other supported telecoms switches.



CDR-Stats is built on Open Source Software where the core components are Django, PostgreSQL, Celery, Redis, Socket.IO, Bower and Bootstrap Framework. There are many more Python and Django dependencies needed but if

you are not a developer, you might want to skip those details as CDR-Stats can simply be installed using a script which installs transparently and seamlessly, CDR-Stats and the stack for you.

2.1.1 Install requirements

The *requirements* files provides a way to create an environment where all the dependencies needed for the CDR-Stats are installed.

To get started with CDR-Stats the following must be installed:

- python >= 2.7 (programming language)
- nginx - Http Server
- django Framework >= 1.7 (Python based Web framework)
- celery >= 3.0 (Asynchronous task queue/job queue based on distributed message passing)
- linaro_django_pagination (Utilities for creating robust pagination tools throughout a django application)
- django-uuidfield >= 0.2 (Provides a UUIDField for your Django models)
- kombu >= 1.0.2 (An AMQP - Advanced Message Queuing Protocol messaging framework for Python)
- python-dateutil >= 1.5 (Extensions to the standard datetime module)
- redis >= 2.2.2 (Redis Python Client)
- django-notification >= 0.1.3 (User notification management for the Django web framework)
- django-country-dialcode - Django reusable application to manage Dial code of Countries

and many more, please find a full list of our requirements to our requirements files:

- https://github.com/cdr-stats/cdr-stats/blob/develop/requirements/basic.txt
- https://github.com/cdr-stats/cdr-stats/blob/develop/requirements/django.txt

There is also 2 extra requirements files for developers and to run our tests:

- https://github.com/cdr-stats/cdr-stats/blob/develop/requirements/dev.txt
- https://github.com/cdr-stats/cdr-stats/blob/develop/requirements/test.txt

The requirements must be installed into a virtual environement so that the dependencies of the application do not interfere with other applications on the server. More information can be found about virtualenv at: http://pypi.python.org/pypi/virtualenv

PIP is a tool for installing and managing Python packages, more information about PIP : http://www.pip-installer.org/en/latest/index.html

Using PIP, you can easily install all the requirements:

```
$ pip install -r requirements/all.txt
```

2.1.2 Running CDR-Stats manually

Inside CDR-Stats directory you should run, the following:

```
$ python manage.py syncdb --noinput

$ python manage.py collectstatic
```

```
$ python manage.py migrate

$ python manage.py createsuperuser

$ python manage.py runserver
```

`syncdb` will create a database named test.db in `database` folder of the CDR-Stats directory. CDR-Stats is configured to do this, but can be changed by modifying settings.py where the DATABASES dictionary is constructed. there is more information about this in the Django documentation.

`collectstatic` will fetch all necessary media files and put them into `static` folder defined in the settings module.

`migrate` will applying database migration to update the database schemas of CDR-Stats to its latest version.

`createsuperuser` will create a super user, to access to the admin section of CDR-Stats.

`runserver` runs an embedded webserver to test the site. By default it will run on http://localhost:8000. This is configurable and more information about `runserver` is in Django documentation.

2.2 Install CDR-Stats

2.2.1 Download and Install CDR-Stats

Our install script supports Debian 7.x and Debian 8.x 64 bit version, we recommend the latest version of Debian.

Install CDR-Stats *Master* branch:

This will copy and un the *master* install script:

```
cd /usr/src/ ; rm install-cdr-stats.sh ; wget --no-check-certificate https://raw.github.com/cdr-stats
```

During the installation, a number of self explanatory questions will be asked, including the root username and password.

On completion CDR-Stats will be ready to use once it is configured to your requirements in settings_local.py as described in the next section, and the CDR-Pusher is installed, usually to your switch, to send CDR to CDR-Stats.

2.2.2 Config file - settings.py & settings_local.py

The main config file for CDR-Stats is located at */usr/share/cdrstats/cdr_stats/settings_local.py*

Before importing CDR, there are some settings that need to be changed to suit your location.

Email Backend

Configure these settings to register to your SMTP server for sending outbound mail.

Allowed Hosts

Normally, this IP address will be configured correctly as part of the installation process, however if the IP address changes, or if you are accessing via another IP, e.g. port forwarding through a firewall or you use an FQDN, the additional IP addresses via which you access CDR-Stats will need to be added here enclosed in single 'quotation' marks and separated with a comma.

General

The general settings deal with how the dialled digits are treated in order to normalise them for matching to a rate.

PHONENUMBER_PREFIX_LIMIT_MIN & PHONENUMBER_PREFIX_LIMIT_MAX are used to determine how many digits are used to match against the dialcode prefix database, e.g:

```
PHONENUMBER_PREFIX_LIMIT_MIN = 2
PHONENUMBER_PREFIX_LIMIT_MAX = 5
```

If a phone number has less digits than PHONENUMBER_MIN_DIGITS it will be considered an extension:

```
PHONENUMBER_MIN_DIGITS = 6
PHONENUMBER_MAX_DIGITS = 9
```

If a phone number has more digits than PHONENUMBER_DIGITS_MIN but less than PHONE_DIGITS_MAX then the phone number will be considered as local or national call and the LOCAL_DIALCODE will be added:

```
LOCAL_DIALCODE = 1
```

Set the dialcode of your country (44 for UK, 1 for US):

```
PREFIX_TO_IGNORE = "+,0,00,000,0000,00000,011,55555,99999"
```

List of prefixes to ignore, these prefixes are removed from the phone number prior to analysis.

Country Examples

So for the USA, to cope with 10 or 11 digit dialling, PHONENUMBER_MAX_DIGITS would be set to 10, and LOCAL_DIALCODE set to 1. Thus 10 digit numbers would have a 1 added, but 11 digit numbers are left untouched.

In the UK, the number of significant digits is either 9 or 10 after the "0" trunk code. So to ensure that all UK numbers had 44 prefixed to them and the single leading 0 removed, the prefixes to ignore would include 0, the PHONENUMBER_MAX_DIGITS would be set to 10, and the LOCAL_DIALCODE would be 44.

In Spain, where there is no "0" trunk code, and the length of all numbers is 9, then the PHONENUMBER_MAX_DIGITS would be set to 9, and the LOCAL_DIALCODE set to 34.

When any changes are made to this file, then Celery should be restarted to apply the changes.

2.3 Configure Postgresql for Remote Access

2.3.1 2.1 First backup your conf files

Backup postgresql.conf & pg_hba.conf:

```
cp /etc/postgresql/9.4/main/postgresql.conf /etc/postgresql/9.4/main/postgresql.conf.bkup
cp /etc/postgresql/9.4/main/pg_hba.conf /etc/postgresql/9.4/main/pg_hba.conf.bkup
```

2.3.2 2.2 Allow TCP/IP socket

Edit the PostgreSQL configuration file, using a text editor such as vi.

Configure PostgreSQL to listen for remote connections:

```
sed -i "s/#listen_addresses = 'localhost'/listen_addresses = '*'/" /etc/postgresql/9.4/main/postgres
```

2.3.3 2.3 Enable client authentication

Configure PostgreSQL to accept remote connections (from any host on your network):

```
cat >> /etc/postgresql/9.4/main/pg_hba.conf <<EOF
# Accept all IPv4 connections
host    all         all         <SWITCH_IP>/24             md5
EOF
```

Make sure you replace <SWITCH_IP>/24 with your actual network IP address range.

If you want to accept CDR from only from one IP address, then enter the IP in switch, followed by /32, e.g. <SWITCH_IP>/32

2.3.4 2.4 Restart PostgreSQL Server

Restart PostgreSQL for the changes to take effect:

```
/etc/init.d/postgresql restart
```

2.3.5 2.5 Setup firewall Iptables

Make sure iptables is not blocking communication, open port 5432:

```
iptables -A INPUT -p tcp -s 0/0 --sport 1024:65535 -d <SWICH_IP>  --dport 5432 -m state --state NEW,
iptables -A OUTPUT -p tcp -s <SWICH_IP> --sport 5432 -d 0/0 --dport 1024:65535 -m state --state ESTA
```

Restart firewall:

```
/etc/init.d/iptables restart
```

2.3.6 2.6 Test your setup

In order to test, you will need to install PostgreSQL client, on Debian you can install as follows:

```
apt-get install postgresql-client
```

For CentOS:

```
yum install postgresql
```

Use psql command from client system. Connect to remote server using IP address and login using vivek username and sales database, enter:

```
$ psql -h <POSTGRESQL_IP> -U USERNAME -d CDRPUSHER_DBNAME
```

Replace POSTGRESQL_IP, USERNAME and CDRPUSHER_DBNAME, with the one from your CDR-Stats server.

Check *settings_local.py* for the username and password.

2.4 CDR-Pusher Installation

CDR-Pusher is a Go Application that will push your CDRs (Call Detail Record) from your Telco Switch (Asterisk, FreeSWITCH or other supported switch http://www.cdr-stats.org/pricing/switch-connectors/) to the centralized PostgreSQL Database CDR-Pusher on the CDR-Stats server.rebo

2.4.1 3.1 Install / Run

Install Golang dependencies (Debian/Ubuntu):

```
$ apt-get -y install mercurial git bzr bison
$ apt-get -y install bison
```

Install GVM to select which version of Golang you want to install:

```
$ bash < <(curl -s -S -L https://raw.githubusercontent.com/moovweb/gvm/master/binscripts/gvm-installe
$ source /root/.gvm/scripts/gvm
$ gvm install go1.4.2 --binary
$ gvm use go1.4.2 --default
```

Make sure you are running by default Go version >= 1.4.2, check by typing the following:

```
$ go version
```

To install and run the cdr-pusher application, follow these steps:

```
$ mkdir /opt/app
$ cd /opt/app
$ git clone https://github.com/cdr-stats/cdr-pusher.git
$ cd cdr-pusher
$ export GOPATH=`pwd`
$ make build
$ ./bin/cdr-pusher
```

The config file cdr-pusher.yaml is installed at the following location: /etc/cdr-pusher.yaml

2.4.2 3.2 Configuration file

Config file /etc/cdr-pusher.yaml:

```
# CDR FETCHING - SOURCE
# ---------------------

# storage_source_type: DB backend type where CDRs are stored
# (accepted values: "sqlite3" and "mysql")
storage_source: "sqlite3"

# db_file: specify the database path and name
db_file: "/usr/local/freeswitch/cdr.db"

# Database DNS
# Use this with Mysql
db_dns: ""

# db_table: the DB table name
db_table: "cdr"
```

```
# db_flag_field defines the table field that will be added/used to track the import
db_flag_field: "flag_imported"

# max_fetch_batch: Max number of CDR to push in batch (value: 1-1000)
max_fetch_batch: 100

# heartbeat: Frequency of check for new CDRs in seconds
heartbeat: 1

# cdr_fields is list of fields that will be fetched (from SQLite3) and pushed (to PostgreSQL)
# - if dest_field is callid, it will be used in riak as key to insert
cdr_fields:
    - orig_field: uuid
      dest_field: callid
      type_field: string
    - orig_field: caller_id_name
      dest_field: caller_id_name
      type_field: string
    - orig_field: caller_id_number
      dest_field: caller_id_number
      type_field: string
    - orig_field: destination_number
      dest_field: destination_number
      type_field: string
    - orig_field: hangup_cause_q850
      dest_field: hangup_cause_id
      type_field: int
    - orig_field: duration
      dest_field: duration
      type_field: int
    - orig_field: billsec
      dest_field: billsec
      type_field: int
    # - orig_field: account_code
    #   dest_field: accountcode
    #   type_field: string
    - orig_field: "datetime(start_stamp)"
      dest_field: starting_date
      type_field: date
    # - orig_field: "strftime('%s', answer_stamp)" # convert to epoch
    - orig_field: "datetime(answer_stamp)"
      dest_field: extradata
      type_field: jsonb
    - orig_field: "datetime(end_stamp)"
      dest_field: extradata
      type_field: jsonb

# CDR PUSHING - DESTINATION
# -------------------------

# storage_dest_type defines where push the CDRs (accepted values: "postgres" or "riak")
storage_destination: "postgres"

# Used when storage_dest_type = postgres
# datasourcename: connect string to connect to PostgreSQL used by sql.Open
pg_datasourcename: "user=postgres password=password host=localhost port=5432 dbname=cdr-pusher sslmo

# Used when storage_dest_type = postgres
```

```
# pg_store_table: the DB table name to store CDRs in Postgres
table_destination: "cdr_import"

# Used when storage_dest_type = riak
# riak_connect: connect string to connect to Riak used by riak.ConnectClient
riak_connect: "127.0.0.1:8087"

# Used when storage_dest_type = postgres
# riak_bucket: the bucket name to store CDRs in Riak
riak_bucket: "cdr_import"

# switch_ip: leave this empty to default to your external IP (accepted value: ""|"your IP")
switch_ip: ""

# cdr_source_type: write the id of the cdr sources type
# (accepted value: unknown: 0, csv: 1, api: 2, freeswitch: 3, asterisk: 4, yate: 5, kamailio: 6, oper
cdr_source_type: 0

# SETTINGS FOR FAKE GENERATOR
# ---------------------------

# fake_cdr will populate the SQLite database with fake CDRs for testing purposes (accepted value: "ye
fake_cdr: "no"

# fake_amount_cdr is the number of CDRs to generate into the SQLite database for testing (value: 1-10
# this amount of CDRs will be created every second
fake_amount_cdr: 1000
```

2.4.3 3.3 Deployment

CDR-Pusher application aims to be run as Service, it can easily be run by Supervisord.

3.3.1 Install Supervisord

Some Linux distributions offer a version of Supervisor that is installable through the system package manager. These packages may include distribution-specific changes to Supervisor:

```
$ apt-get install supervisor
```

3.3.2 Configure CDR-Pusher with Supervisord

Create an Supervisor conf file for cdr-pusher:

```
$ vim /etc/supervisor/conf.d/cdr-pusher-prog.conf
```

A supervisor configuration could look as follow:

```
[program:cdr-pusher]
autostart=true
autorestart=true
startretries=10
startsecs = 5
directory = /opt/app/cdr-pusher/bin
command = /opt/app/cdr-pusher/bin/cdr-pusher
user = root
```

```
redirect_stderr = true
stdout_logfile = /var/log/cdr-pusher/cdr-pusher.log
stdout_logfile_maxbytes=50MB
stdout_logfile_backups=10
```

Make sure the director to store the logs is created, in this case you should create '/var/log/cdr-pusher':

```
$ mkdir /var/log/cdr-pusher
```

3.3.4 Supervisord Manage

Supervisord provides 2 commands, supervisord and supervisorctl:

```
supervisord: Initialize Supervisord, run configed processes
supervisorctl stop programX: Stop process programX. programX is config name in [program:mypkg].
supervisorctl start programX: Run the process.
supervisorctl restart programX: Restart the process.
supervisorctl stop groupworker: Restart all processes in group groupworker
supervisorctl stop all: Stop all processes. Notes: start, restart and stop won't reload the latest co
supervisorctl reload: Reload the latest configs.
supervisorctl update: Reload all the processes where the config has changed.
```

3.3.5 Supervisord Service

You can also use supervisor using the supervisor service:

```
$ /etc/init.d/supervisor start
```

2.4.4 3.4 Configure CDR-Pusher

Edit */etc/cdr-pusher.yaml*

Get started by configuring the CDR source, this is your original CDR backend, for instance on Asterisk this can be MySQL, SQlite or Postgresql.

For Mysql & PostgreSQL you will need to configure the DNS too: https://github.com/go-sql-driver/mysql

Some of the settings to configure:

```
# storage_source_type: DB backend type where CDRs are stored
# (accepted values: "sqlite3" and "mysql")
storage_source: "mysql"

# Database DNS
db_dns: "root:password@/accounting"
```

Then configure the 'CDR Pushing' section, here you will need to define where the CDRs will go, this will 'almost' always be the 'cdr-pusher' database living on your CDR-Stats server.

Check your CDR-Stats installation, you should find the Database settings for cdr-pusher database in settings_local.py

Some of the settings to configure:

```
# storage_dest_type defines where push the CDRs (accepted values: "postgres", "riak" or "both")
storage_destination: "postgres"
```

```
# Used when storage_dest_type = postgres
pg_datasourcename: "user=postgres password=password host=localhost port=5432 dbname=cdr-pusher sslmo
```

2.4.5 3.5 Configure your Switch CDR with CDR-Pusher

You will need to configure CDR-Pusher and you Telco Switch to work together, for this we put some individual instructions for :

> Configure FreeSWITCH with CDR-Stats and CDR-Pusher - *Configure FreeSWITCH with CDR-Stats and CDR-Pusher*

> Configure Asterisk with CDR-Stats and CDR-Pusher - *Configure Asterisk with CDR-Stats and CDR-Pusher*

> Configure Kamailio with CDR-Stats and CDR-Pusher - *Configure Kamailio with CDR-Stats and CDR-Pusher*

2.4.6 3.6 Restart Supervisord

After changes in CDR-Pusher configuration you will need to restart supervisord, you can do so with gently with:

```
/etc/init.d/supervisor stop
/etc/init.d/supervisor start
```

2.4.7 3.7 Troubleshooting

An easy way to verify that CDR-Stats is running smoothly is to look at the logs.

Find the import log activity on CDR-Stats at:

```
tail -f  /var/log/cdr-stats/djcelery_error.log
```

Find the import log activity on CDR-Pusher at:

```
tail -f /var/log/cdr-pusher/cdr-pusher.log
```

Check out the CDR-Stats Database 'import_cdr' to see realtime import:

```
python manage.py dbshell --database=import_cdr
```

Specific configuration per switch:

2.5 Configure Asterisk with CDR-Stats and CDR-Pusher

Asterisk supports many backends to store CDRs: SQLite3, PostgreSQL, MySQL and many more.

In this document, we will explain how to configure Asterisk to store CDRs in SQLite3 or Mysql then configure CDR-Pusher to send the CDR to CDR-Stats. Sqlite3 is the one we will recommend as this is by far the easiest to setup.

2.5.1 Store Asterisk CDRs to SQLITE3

The cdr_sqlite module was deprecated and has been removed. Users of this module should use the cdr_sqlite3_custom module instead.

If Asterisk is compiled from source, then providing that SQLite3 is installed, then during make menuselect under Call Detail Recording, cdr_sqlite3_custom can be selected for installation.

For those using Asterisk via RPMs such as in the popular free PBX system, then something like yum install asterisk11-sqlite3.x86_64. Do *yum search sqlite3* to find the correct module for your version of Asterisk.

There is only one config file for the *cdr_sqlite3_custom.so* module, this is configured at */etc/asterisk/cdr_sqlite3_custom.conf* and the default settings are as follows:

```
;
; Mappings for custom config file
;
[master] ; currently, only file "master.db" is supported, with only one table at a time.
table => cdr
columns => calldate, clid, dcontext, channel, dstchannel, lastapp, lastdata,source, destination, dura
values => '${CDR(start)}','${CDR(clid)}','${CDR(dcontext)}','${CDR(channel)}','${CDR(dstchannel)}','$
```

After installation, restart asterisk. When CDR are written, they will be found at */var/log/asterisk/master.db*.

To check that CDR are being written to the SQLite3 DB with the following:

```
$ sqlite3 /var/log/asterisk/master.db
$ SELECT * FROM cdr LIMIT 10;
```

The result will be:

```
SQLite version 3.6.20
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>

For readability, type
.header on
.mode column
Then you can list your CDR with standard SQL commands, e.g.
select * from cdr;

CTRL-D exits the SQLite console
```

2.5.2 Store Asterisk CDRs to MySQL

There is only one config file for the *cdr_mysql.so* module, this is configured at */etc/asterisk/cdr_mysql.conf* and the default settings are as follows:

```
;
; Note - if the database server is hosted on the same machine as the
; asterisk server, you can achieve a local Unix socket connection by
; setting hostname=localhost
;
; port and sock are both optional parameters. If hostname is specified
; and is not "localhost", then cdr_mysql will attempt to connect to the
; port specified or use the default port. If hostname is not specified
; or if hostname is "localhost", then cdr_mysql will attempt to connect
; to the socket file specified by sock or otherwise use the default socket
; file.
;
[global]
hostname=localhost
dbname=asteriskcdrdb
```

```
password=password
user=asteriskcdruser
table=cdr
;port=3306
;sock=/tmp/mysql.sock
;userfield=1
```

Enable the last option *userfield* if you wish to use SetCDRUserField.

Configure with your hostname, dbname, password, user and table.

After installation, restart asterisk.

To check that CDR are being written to the MySQL DB with the following:

```
$ mysql -uasteriskcdruser -pasteriskcdrdb asteriskcdrdb
$ SELECT * FROM cdr LIMIT 10;
```

The result will be:

```
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4862
Server version: 5.5.44-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> select * from cdr LIMIT 10;
...
...

CTRL-D exits the MySQL console
```

2.5.3 Configure CDR-pusher to collect CDRs

Once your CDRs will be stored to a SQLite Database, you will have to install CDR-Pusher on your Asterisk server. You can find instructions how to install CDR-Pusher here: https://github.com/cdr-stats/cdr-pusher

To install Supervisor on CentOS 6 or RHEL6, the procedure is more complex, here it's how we do it:

```
$ yum -y install python-setuptools

$ easy_install supervisor

$ wget https://raw.githubusercontent.com/cdr-stats/cdr-stats/develop/install/supervisor/centos/superv

$ wget https://raw.githubusercontent.com/cdr-stats/cdr-stats/develop/install/supervisor/centos/superv

$ chmod +x /etc/init.d/supervisor

$ supervisord --version
```

```
$ /etc/init.d/supervisor stop ; sleep 2 ; /etc/init.d/supervisor start
```

Also make sure you have recent version of Git.

Check your git version with:

```
git $ version
```

If your git version <= 1.7.4, then you will need to install a recent version, you can follow the instructions here how to install a recent Git on CentOS6 here: http://tecadmin.net/how-to-upgrade-git-version-1-7-10-on-centos-6/

After installation of CDR-Pusher you can find the configuration file at '/etc/cdr-pusher.yaml'. You will need to configure properly some settings in order to connect CDR-pusher to your SQLite or MySQL CDR backend and to your CDR-Stats server.

2.5.4 Configure CDR-Pusher for SQLite3

Here some of the settings you need to change to fetch SQLite CDR form Asterisk, edit '/etc/cdr-pusher.yaml':

```
# storage_source_type: type to CDRs to push
storage_source: "sqlite3"

# db_file: specify the database path and name
db_file: "/var/log/asterisk/master.db"

# db_table: the DB table name
db_table: "cdr"

# cdr_fields is list of fields that will be fetched (from SQLite3) and pushed (to PostgreSQL)
# - if dest_field is callid, it will be used in riak as key to insert
cdr_fields:
    - orig_field: uniqueid
      dest_field: callid
      type_field: string
    - orig_field: "'' AS cidnum"
      dest_field: caller_id_number
      type_field: string
    - orig_field: clid
      dest_field: caller_id_name
      type_field: string
    - orig_field: destination
      dest_field: destination_number
      type_field: string
    - orig_field: "CASE WHEN disposition='ANSWER' THEN 16 WHEN disposition='ANSWERED' THEN 16 WHEN di
      dest_field: hangup_cause_id
      type_field: int
    - orig_field: CAST(duration AS INTEGER)
      dest_field: duration
      type_field: int
    - orig_field: CAST(billsec AS INTEGER)
      dest_field: billsec
      type_field: int
    - orig_field: "datetime(calldate)"
      dest_field: starting_date
      type_field: date
    - orig_field: accountcode
      dest_field: accountcode
```

```
      type_field: string
    - orig_field: channel
      dest_field: extradata
      type_field: jsonb
    - orig_field: lastapp
      dest_field: extradata
      type_field: jsonb
    - orig_field: dcontext
      dest_field: extradata
      type_field: jsonb
```

2.5.5 Configure CDR-Pusher for MySQL

Here some of the settings you need to change to fetch MySQL CDR from Asterisk, edit '/etc/cdr-pusher.yaml':

```
# storage_source_type: type to CDRs to push
storage_source: "mysql"

# db_file: specify the database path and name
db_file: ""

# Database DNS
# Use this with MySQL
db_dns: "username:password@/database"

# db_table: the DB table name
db_table: "cdr"

# cdr_fields is list of fields that will be fetched and pushed (to PostgreSQL)
# - if dest_field is callid, it will be used in riak as key to insert
cdr_fields:
    - orig_field: uniqueid
      dest_field: callid
      type_field: string
    - orig_field: clid
      dest_field: caller_id_name
      type_field: string
    - orig_field: "'' AS cidnum"
      dest_field: caller_id_number
      type_field: string
    - orig_field: dst
      dest_field: destination_number
      type_field: string
    - orig_field: "CASE disposition WHEN 'ANSWER' THEN 16 WHEN 'ANSWERED' THEN 16 WHEN 'BUSY' THEN 1
      dest_field: hangup_cause_id
      type_field: int
    - orig_field: duration
      dest_field: duration
      type_field: int
    - orig_field: billsec
      dest_field: billsec
      type_field: int
    - orig_field: accountcode
      dest_field: accountcode
      type_field: string
    - orig_field: calldate
      dest_field: starting_date
```

```
      type_field: date
    - orig_field: userfield
      dest_field: extradata
      type_field: jsonb
    - orig_field: dcontext
      dest_field: extradata
      type_field: jsonb
    - orig_field: channel
      dest_field: extradata
      type_field: jsonb
    - orig_field: lastapp
      dest_field: extradata
      type_field: jsonb
    - orig_field: lastdata
      dest_field: extradata
      type_field: jsonb
```

CDR-Pusher always needs a Primary Key to import CDRs, therefore if you use MySQL, please ensure that you have a Primary Key in your *cdr* table as it will not be there by default.

You can create a Primary Key with:

```
ALTER TABLE cdr ADD COLUMN id int(10) UNSIGNED PRIMARY KEY AUTO_INCREMENT FIRST;
```

2.5.6 Send CDRs from backend to the CDR-Stats Core DB

The application cdr-pusher will need your correct CDR-Stats server settings to push CDRs properly to the core DB, you set this in '/etc/cdr-pusher.yaml' by changing:

```
pg_datasourcename: "user=postgres password=password host=localhost port=5432 dbname=cdr-pusher sslmo
```

Replace 'postgres', 'password' and 'localhost' by your CDR-Stats server settings and make sure you configured Remote Access to PostgreSQL, this is described in our documentation here *Configure Postgresql for Remote Access*.

You may need to configure these settings as well:

```
# switch_ip: leave this empty to default to your external IP (accepted value: ""|"your IP")
switch_ip: ""

# cdr_source_type: write the id of the cdr sources type
# (accepted value: unknown: 0, csv: 1, api: 2, freeswitch: 3, asterisk: 4, yate: 5, kamailio: 6, oper
cdr_source_type: 4
```

2.5.7 Restart CDR-Pusher

After changes in '/etc/cdr-pusher.yaml' CDR-pusher will need to be restarted, do this with the following command:

```
$ /etc/init.d/supervisor stop
$ /etc/init.d/supervisor start
```

2.6 Configure FreeSWITCH with CDR-Stats and CDR-Pusher

FreeSWITCH supports many backed to store CDRs, we will cover SQLite here.

2.6.1 Collect CDRs from SQLITE

FreeSWITCH mod_cdr_sqlite is used to locally store the CDRs, to configure CDR SQLite backend in FreeSWITCH you can find instruction here: https://wiki.freeswitch.org/wiki/Mod_cdr_sqlite

Once your CDRs will be stored to a SQLite Database, you will have to install CDR-Pusher on your FreeSWITCH server. You can find instruction how to install CDR-Pusher here: https://github.com/cdr-stats/cdr-stats

After installation of CDR-Pusher you can find the configuration file at '/etc/cdr-pusher.yaml'. You will need to configure properly some settings in order to connect CDR-pusher to your SQLite CDR backend and to your CDR-Stats server.

By tweaking the configuration of Mod_cdr_sqlite and CDR-Pusher you can define custom fields that you want to import to CDR-stats.

Here an example of 'cdr_sqlite.conf' that show how custom fields can be defined to store some specific CDR variables to your CDR backend:

```
<configuration name="cdr_sqlite.conf" description="SQLite CDR">
  <settings>
    <!-- SQLite database name (.db suffix will be automatically appended) -->
    <!-- <param name="db-name" value="cdr"/> -->
    <!-- CDR table name -->
    <!-- <param name="db-table" value="cdr"/> -->
    <!-- Log a-leg (a), b-leg (b) or both (ab) -->
    <param name="legs" value="a"/>
    <!-- Default template to use when inserting records -->
    <param name="default-template" value="example"/>
    <!-- This is like the info app but after the call is hung up -->
    <!--<param name="debug" value="true"/>-->
  </settings>
  <templates>
    <!-- Note that field order must match SQL table schema, otherwise insert will fail -->
    <template name="example">"${caller_id_name}","${caller_id_number}","${destination_number}","${cor
  </templates>
</configuration>
```

2.6.2 Configure CDR-pusher to collect CDRs

Here some of the settings you need to change to fetch CDR form Asterisk, edit '/etc/cdr-pusher.yaml':

```
# storage_source_type: DB backend type where CDRs are stored
# (accepted values: "sqlite3" and "mysql")
storage_sourcestorage_source: "sqlite3"

# db_file: specify the database path and name
# db_file: "/usr/local/freeswitch/cdr.db"

# cdr_fields is list of fields that will be fetched (from SQLite3) and pushed (to PostgreSQL)
# - if dest_field is callid, it will be used in riak as key to insert
cdr_fields:
    - orig_field: uuid
      dest_field: callid
      type_field: string
    - orig_field: caller_id_name
      dest_field: caller_id_name
      type_field: string
    - orig_field: caller_id_number
```

```
      dest_field: caller_id_number
      type_field: string
    - orig_field: destination_number
      dest_field: destination_number
      type_field: string
    - orig_field: hangup_cause_q850
      dest_field: hangup_cause_id
      type_field: int
    - orig_field: duration
      dest_field: duration
      type_field: int
    - orig_field: billsec
      dest_field: billsec
      type_field: int
    # - orig_field: account_code
    #   dest_field: accountcode
    #   type_field: string
    - orig_field: "datetime(start_stamp)"
      dest_field: starting_date
      type_field: date
    # - orig_field: "strftime('%s', answer_stamp)" # convert to epoch
    - orig_field: "datetime(answer_stamp)"
      dest_field: extradata
      type_field: jsonb
    - orig_field: "datetime(end_stamp)"
      dest_field: extradata
      type_field: jsonb
```

2.6.3 Send CDRs from backend to the CDR-Stats Core DB

The application cdr-pusher will need your correct CDR-Stats server settings to push CDRs properly to the core DB, you set this in '/etc/cdr-pusher.yaml' by changing:

```
pg_datasourcename: "user=postgres password=password host=localhost port=5432 dbname=cdr-pusher sslmo
```

Replace 'postgres', 'password' and 'localhost' by your CDR-Stats server settings and make sure you configured Remote Access to PostgreSQL, this is described in our documentation here *Configure Postgresql for Remote Access*.

You may want to configure properly those 2 settings also:

```
# switch_ip: leave this empty to default to your external IP (accepted value: ""|"your IP")
switch_ip: ""

# cdr_source_type: write the id of the cdr sources type
# (accepted value: unknown: 0, csv: 1, api: 2, freeswitch: 3, asterisk: 4, yate: 5, kamailio: 6, oper
cdr_source_type: 3
```

2.6.4 Restart CDR-Pusher

After changes in '/etc/cdr-pusher.yaml' CDR-pusher will need to be restarted, do this with the following command:

```
$ /etc/init.d/supervisor stop
$ /etc/init.d/supervisor start
```

2.7 Configure Kamailio with CDR-Stats and CDR-Pusher

In Kamailio, you can store easily your CDR using Mysql, using the 'acc module' (kamailio.org/docs/modules/4.0.x/modules/acc.html). You will need to configure Kamailio to store CDRs to Mysql and afterwards you will have to install CDR-Pusher on your Kamailio server to push those CDRs to the CDR-Stats server.

2.7.1 Collect CDRs from Kamailio MYSQL Database

Kamailio and module acc can help you storing your CDRs to a Mysql database. Here you can find some of the SQL schema and procedure that will be needed to achieve it http://siremis.asipto.com/install-accounting/

Simeris have some documentation on how to setup accounting services: http://kb.asipto.com/siremis:install40x:accounting

You will end up with a Mysql cdr table similar to this one:

```
CREATE TABLE `cdrs` (
  `cdr_id` bigint(20) NOT NULL AUTO_INCREMENT,
  `src_username` varchar(64) NOT NULL DEFAULT '',
  `src_domain` varchar(128) NOT NULL DEFAULT '',
  `dst_username` varchar(64) NOT NULL DEFAULT '',
  `dst_domain` varchar(128) NOT NULL DEFAULT '',
  `dst_ousername` varchar(64) NOT NULL DEFAULT '',
  `call_start_time` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  `duration` int(10) unsigned NOT NULL DEFAULT '0',
  `sip_call_id` varchar(128) NOT NULL DEFAULT '',
  `sip_from_tag` varchar(128) NOT NULL DEFAULT '',
  `sip_to_tag` varchar(128) NOT NULL DEFAULT '',
  `src_ip` varchar(64) NOT NULL DEFAULT '',
  `cost` int(11) NOT NULL DEFAULT '0',
  `rated` int(11) NOT NULL DEFAULT '0',
  `created` datetime NOT NULL,
  PRIMARY KEY (`cdr_id`),
  UNIQUE KEY `uk_cft` (`sip_call_id`,`sip_from_tag`,`sip_to_tag`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=latin1;
```

You will have to install the stored procedure 'kamailio_cdrs' & 'kamailio_rating' and call them from your Kamailio config.

In order to register failed calls to missed_calls, you will need to set flag 'FLT_ACCFAILED' and 'FLT_ACCMISSED' as follow:

```
if (is_method("INVITE"))
{
setflag(FLT_ACC); # do accounting
    setflag(FLT_ACCFAILED); # -- this is added to record failed calls
    setflag(FLT_ACCMISSED);
}
```

2.7.2 Install Triggers to regroup CDRs

The triggers will push your new Kamailio CDRs to a new table *collection_cdrs*. This table helps to merge both table entries 'cdr and 'missed_calls', that way we could send the CDRs easily from CDR-Pusher application.

Connect to your Kamailio Mysql Database and create the following table and triggers:

```
DROP TABLE IF EXISTS `collection_cdrs`;

CREATE TABLE `collection_cdrs` (
    `id` bigint(20) NOT NULL auto_increment,
    `cdr_id` bigint(20) NOT NULL default '0',
    `src_username` varchar(64) NOT NULL default '',
    `src_domain` varchar(128) NOT NULL default '',
    `dst_username` varchar(64) NOT NULL default '',
    `dst_domain` varchar(128) NOT NULL default '',
    `dst_ousername` varchar(64) NOT NULL default '',
    `call_start_time` datetime NOT NULL default '0000-00-00 00:00:00',
    `duration` int(10) unsigned NOT NULL default '0',
    `sip_call_id` varchar(128) NOT NULL default '',
    `sip_from_tag` varchar(128) NOT NULL default '',
    `sip_to_tag` varchar(128) NOT NULL default '',
    `src_ip` varchar(64) NOT NULL default '',
    `cost` integer NOT NULL default '0',
    `rated` integer NOT NULL default '0',
    `sip_code` char(3) NOT NULL default '',
    `sip_reason` varchar(32) NOT NULL default '',
    `created` datetime NOT NULL,
    `flag_imported` integer NOT NULL default '0',
    PRIMARY KEY  (`id`)
);

DELIMITER //
CREATE TRIGGER copy_cdrs
AFTER INSERT
    ON cdrs FOR EACH ROW
BEGIN
    INSERT INTO collection_cdrs SET
        cdr_id = NEW.cdr_id,
        src_username = NEW.src_username,
        src_domain = NEW.src_domain,
        dst_username = NEW.dst_username,
        dst_domain = NEW.dst_domain,
        dst_ousername = NEW.dst_ousername,
        call_start_time = NEW.call_start_time,
        duration = NEW.duration,
        sip_call_id = NEW.sip_call_id,
        sip_from_tag = NEW.sip_from_tag,
        sip_to_tag = NEW.sip_to_tag,
        src_ip = NEW.src_ip,
        cost = NEW.cost,
        rated = NEW.rated,
        sip_code = 200,
        sip_reason = ''
        ;
END; //
DELIMITER ;

DELIMITER //
CREATE TRIGGER copy_missed_calls
AFTER INSERT
    ON missed_calls FOR EACH ROW
BEGIN
    INSERT INTO collection_cdrs SET
        cdr_id = NEW.cdr_id,
```

```
        src_username = NEW.src_user,
        src_domain = NEW.src_domain,
        dst_username = NEW.dst_user,
        dst_domain = NEW.dst_domain,
        dst_ousername = NEW.dst_ouser,
        call_start_time = NEW.time,
        duration = 0,
        sip_call_id = NEW.callid,
        sip_from_tag = NEW.from_tag,
        sip_to_tag = NEW.to_tag,
        src_ip = NEW.src_ip,
        cost = 0,
        rated = 0,
        sip_code = NEW.sip_code,
        sip_reason = NEW.sip_reason
        ;
END; //
DELIMITER ;
```

2.7.3 Import previous CDRs and Missed Calls

If you were already collecting CDRs in Kamailio, you may want to import the existing ones to the table 'collection_cdrs', you can do the following with those SQL commands:

```
-- !!! Only do the following once !!!

-- import cdrs
INSERT collection_cdrs (cdr_id, src_username, src_domain, dst_username, dst_domain, dst_ousername, ca


-- import missed_calls
INSERT collection_cdrs (cdr_id, src_username, src_domain, dst_username, dst_domain, dst_ousername, ca
```

2.7.4 Install CDR-Pusher

Once your CDRs will be stored to a Mysql Database, you will have to install CDR-Pusher on your Kamailio server. You can find instruction how to install CDR-Pusher here: https://github.com/cdr-stats/cdr-stats

After installation of CDR-Pusher you can find the configuration file at '/etc/cdr-pusher.yaml'. You will need to configure properly some settings in order to connect CDR-pusher to your Mysql CDR backend and to your CDR-Stats server.

2.7.5 Configure CDR-pusher to collect CDRs

Here some of the settings you need to change to fetch CDR form Kamailio, edit '/etc/cdr-pusher.yaml':

```
# storage_source_type: DB backend type where CDRs are stored
# (accepted values: "sqlite3" and "mysql")
storage_source: "mysql"

# Database DNS
db_dns: "username:password@/database"

# db_table: the DB table name
```

```
db_table: "collection_cdrs"

# cdr_fields is list of fields that will be fetched (from SQLite3) and pushed (to PostgreSQL)
# - if dest_field is callid, it will be used in riak as key to insert
cdr_fields:
    - orig_field: sip_call_id
      dest_field: callid
      type_field: string
    - orig_field: src_username
      dest_field: caller_id_number
      type_field: string
    - orig_field: src_username
      dest_field: caller_id_name
      type_field: string
    - orig_field: dst_username
      dest_field: destination_number
      type_field: string
    - orig_field: "CASE sip_code WHEN '400' THEN 41 WHEN '401' THEN 21 WHEN '402' THEN 21 WHEN '403'
      dest_field: hangup_cause_id
      type_field: int
    - orig_field: CONVERT(duration,UNSIGNED INTEGER)
      dest_field: duration
      type_field: int
    - orig_field: CONVERT(duration,UNSIGNED INTEGER)
      dest_field: billsec
      type_field: int
    - orig_field: "call_start_time"
      dest_field: starting_date
      type_field: date
```

2.7.6 Send CDRs from backend to the CDR-Stats Core DB

The application cdr-pusher will need your correct CDR-Stats server settings to push CDRs properly to the core DB, you set this in '/etc/cdr-pusher.yaml' by changing:

```
pg_datasourcename: "user=postgres password=password host=localhost port=5432 dbname=cdr-pusher sslmo
```

Replace 'postgres', 'password' and 'localhost' by your CDR-Stats server settings and make sure you configured Remote Access to PostgreSQL, this is described in our documentation here *Configure Postgresql for Remote Access*.

You may want to configure properly those 2 settings also:

```
# switch_ip: leave this empty to default to your external IP (accepted value: ""|"your IP")
switch_ip: ""

# cdr_source_type: write the id of the cdr sources type
# (accepted value: unknown: 0, csv: 1, api: 2, freeswitch: 3, asterisk: 4, yate: 5, kamailio: 6, oper
cdr_source_type: 6
```

2.7.7 Restart CDR-Pusher

After changes in '/etc/cdr-pusher.yaml' CDR-pusher will need to be restarted, do this with the following command:

```
/etc/init.d/supervisor stop
/etc/init.d/supervisor start
```

CHAPTER 3

Configuration and Defaults

Contents:

3.1 General Configuration

Some of the more important parts of the configuration module for the cdr_stats, `settings_local.py`, are explained below.

`APPLICATION_DIR` now contains the full path of the project folder and can be used elsewhere in the `settings.py` module so that the project may be moved around the system without having to worry about changing any hard-coded paths:

```
import os.path
APPLICATION_DIR = os.path.dirname(globals()['__file__'])
```

Turns on debug mode allowing the browser user to see project settings and temporary variables.

```
DEBUG = True
```

Sends all errors from the production server to the admin's email address:

```
ADMINS = ( ('xyz', 'xyz@abc.com') )
```

Sets up the options required for Django to connect to your database engine:

```
DATABASES = {
    'default': {
        # Add 'postgresql_psycopg2','postgresql','mysql','sqlite3','oracle'
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'DATABASENAME',
        'USER': 'DB_USERNAME',
        'PASSWORD': 'DB_PASSWORD',
        'HOST': 'DB_HOSTNAME',
        'PORT': 'DB_PORT',
        'OPTIONS': {
            #Postgresql Autocommit
            'autocommit': True,
        }
    },
    'import_cdr': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'cdr-pusher',
        'USER': 'postgres',
```

```
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '5433',
        'OPTIONS': {
            'autocommit': True,
        }
    }
}
```

There are 2 database connections, 'default' is the main database of CDR-Stats this contains all the tables. The second database 'import_cdr' is used to import the CDRs from your switch. This database could be on another database server but putting it on the CDR-Stats server is ideal.

CDR-Stats doesn't pull CDRs from your switch, it's the job of the switch to push the CDRs to CDR-Stats.

A mechanism is required to get your CDRs to the 'import_cdr' database, to assist with this, we created CDR-pusher project. CDR-Pusher will usually be installed on your switch server, CDR-Pusher is a Go application that can be extended, it could import CDRs from a different CDRs Database (SQlite, PostgreSQL) and/or from CDR logs files. For more info please visit https://github.com/cdr-stats/cdr-pusher

Tells Django where to find your media files such as images that the `HTML templates` might use.

```
MEDIA_ROOT = os.path.join(APPLICATION_DIR, 'static')

ROOT_URLCONF = 'urls'
```

Tells Django to start finding URL matches at in the `urls.py` module in the `cdr_stats` project folder.

```
TEMPLATE_DIRS = ( os.path.join(APPLICATION_DIR, 'templates'), )
```

Tells Django where to find the HTML template files:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.admin',
    ...
    'cdr',
    'cdr_alert',
    ...
)
```

Tells Django which applications (custom and external) to use in the project. The custom applications, `cdr` etc. are stored in the project folder along with these custom applications.

3.1.1 Mail server

To configure the SMTP client so that reports and alerts are sent via email, edit /usr/share/cdrstats/cdr_stats/settings_local.py, and identify the email section:

```
#EMAIL BACKEND
#=============
# Email configuration
DEFAULT_FROM_EMAIL = 'CDR-Stats <cdr-...@localhost.com>'
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
```

```
EMAIL_PORT = 587
EMAIL_HOST_USER = 'user...@gmail.com'
EMAIL_HOST_PASSWORD = 'password'
EMAIL_SUBJECT_PREFIX = '[CDR-Stats] '
```

Fill in the details to match your SMTP server. The above example is for Gmail. When done, restart Celery and Apache.

To test that the email is working, from the command line type:

```
$ cd /usr/src/cdr-stats/
$ workon cdr-stats
$ python manage.py send_daily_report
```

3.2 Country Reporting

CDR-Stats is able to identify the destination country of the call. This is a useful fraud prevention measure, so that calls to unexpected destinations are immediately apparent. Places that should not be called should be added in the Blacklist in the admin section so that these destinations are highlighted in the call data records.

However, in order to get accurate reporting, the call detail records have to be in international format, e.g. in the USA, this means 11 digit numbers, beginning with a 1, and for other countries, the numbers called should be prefixed with the international dial code.

There is a facility for manipulating the dialled digits reported in the call detail records, as well as identifying calls as internal calls. This is done in the "general" section of /usr/share/cdrstats/cdr_stats/settings_local.py.

3.2.1 1. Prefix Limits

PREFIX_LIMIT_MIN & PREFIX_LIMIT_MAX are used to determine how many digits are used to match against the dialcode prefix database, e.g:

```
PREFIX_LIMIT_MIN = 2
PREFIX_LIMIT_MAX = 5
```

3.2.2 2. Phone Number Length

If a phone number has less significant digits than PN_MIN_DIGITS it will be considered an extension:

```
PN_MIN_DIGITS = 6
PN_MAX_DIGITS = 9
```

NB The Number of significant digits does not include national (0) or international dialing codes (00 or 011), or where 9 is pressed for an outside line.

3.2.3 3. Adding Country Code

If a phone number has more digits than PN_DIGITS_MIN but less than PN_DIGITS_MAX then the phone number will be considered as local or national call and the LOCAL_DIALCODE will be added:

```
LOCAL_DIALCODE = 1
```

Set the dialcode of your country e.g. 44 for UK, 1 for US

3.2.4 4. Prefixes to Ignore

List of prefixes to ignore, these prefixes are removed from the phone number prior to analysis. In cases where customers dial 9 for an outside line, 9, 90 or 900 may need to be removed as well to ensure accurate reporting:

```
PREFIX_TO_IGNORE = "+,0,00,000,0000,00000,011,55555,99999"
```

3.2.5 Examples

So for the USA, to cope with 10 or 11 digit dialling, PN_MAX_DIGITS would be set to 10, and LOCAL_DIALCODE set to 1. Thus 10 digit numbers would have a 1 added, but 11 digit numbers are left untouched.

In the UK, the number of significant digits is either 9 or 10 after the "0" trunk code. So to ensure that all UK numbers had 44 prefixed to them and the single leading 0 removed, the prefixes to ignore would include 0, the PN_MAX_DIGITS would be set to 10, and the LOCAL_DIALCODE would be 44.

In Spain, where there is no "0" trunk code, and the length of all numbers is 9, then the PN_MAX_DIGITS would be set to 9, and the LOCAL_DIALCODE set to 34.

NB: After changing this file, then both celery and apache should be restarted.

3.3 Configuration for Asterisk

3.3.1 Import configuration for Asterisk

Review your database settings and ensure the second database exists and that is configured correctly:

```
# DATABASE SETTINGS
# =================
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'cdrstats-billing',
        'USER': 'postgres',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '5433',
        'OPTIONS': {
            # Postgresql Autocommit
            'autocommit': True,
        }
    },
    'import_cdr': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'cdr-pusher',
        'USER': 'postgres',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '5433',
        'OPTIONS': {
            'autocommit': True,
        }
    }
}
```

You will need to push CDRs from the Asterisk CDR datastore to a CDR-Stats 'import_cdr' database. To help on this job we created CDR-Pusher, please visit the website and the instructions there to install and configure CDR-Stats correctly: https://github.com/cdr-stats/cdr-stats

3.4 Realtime configuration for Asterisk

The Asterisk Manager settings allow CDR-Stats to retrieve Realtime information to show the number of concurrent calls both in realtime and historically.

In Asterisk, add a new user in manager.conf, or one of its #include's for CDR-Stats. Further information about Asterisk Manager can be found here : http://www.voip-info.org/wiki/view/Asterisk+config+manager.conf

The collection of realtime information is done via Collectd (https://collectd.org/) and InfluxDB (http://influxdb.com/.

3.5 Configuration for FreeSWITCH

3.5.1 Import configuration for FreeSWITCH

Review your database settings and ensure the second database exists and that is configured correctly:

```
# DATABASE SETTINGS
# =================
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'cdrstats-billing',
        'USER': 'postgres',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '5433',
        'OPTIONS': {
            # Postgresql Autocommit
            'autocommit': True,
        }
    },
    'import_cdr': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'cdr-pusher',
        'USER': 'postgres',
        'PASSWORD': 'password',
        'HOST': 'localhost',
        'PORT': '5433',
        'OPTIONS': {
            'autocommit': True,
        }
    }
}
```

You will need to push your CDRs from FreeSWITCH CDR datastore to a CDR-Stats 'import_cdr' database. To help on this job we created CDR-Pusher, please visit the website and the instructions there to install and configure CDR-Stats correctly: https://github.com/cdr-stats/cdr-stats

3.6 Realtime configuration for FreeSWITCH

The FreeSWITCH Event Socket Library allow CDR-Stats to retrieve Realtime information to show the number of concurrent calls both in realtime and historically.

The collection of realtime information is done via Collectd (https://collectd.org/) and InfluxDB (http://influxdb.com/.

CDR-Stats can get CDR from both Freeswitch and Asterisk, or a combination of both. Other Telco Switches are supported, please contact us for further information.

3.7 Resetting CDR Data

Sometimes, some experimentation is required to get the optimum settings for country reporting, to achieve this the data can be removed from CDR-Stats and re-imported from the CDR data store correctly.

3.7.1 1. Stop Celery

Stop CDR-Stats celery:

```
/etc/init.d/cdr-stats-celeryd stop
```

3.7.2 2. Empty the CDR-Stats dbshell

Enter in the virtualenv and launch dbshell the following commands:

```
$ workon cdr-stats
$ cd /usr/share/cdrstats/
$ python manage.py dbshell
```

Now you are connected on PostgreSQL cli, this is the internal database of CDR-Stats.

The following command will delete all the CDRs, make sure you know what are you doing here and that your CDRs are backed in the upstream CDR data store.

$ DELETE FROM voip_cdr;

CTRL-D exits the console.

3.7.3 3. Flag the CDR records for reimport

Enter in the virtualenv and launch dbshell the following commands:

```
$ workon cdr-stats
$ cd /usr/share/cdrstats/
$ python manage.py dbshell --database=import_cdr
```

Enter the postgresql password found in *settings_local_py* conf file.

Now you are connected on PostgreSQL cli, you can flag CDRs for reimport:

```
$ UPDATE cdr_import SET imported=FALSE;

CTRL-D exits the console.
```

3.7.4 4. Start Celery

Start CDR-Stats celery:

```
/etc/init.d/cdr-stats-celeryd start
```

3.7.5 5. Wait while the CDR are re-imported

Go to the diagnostic page to check if CDR-Stats is correctly configured and if data is being imported.

3.8 Celery Configuration

3.8.1 After installing Broker (Redis or Rabbitmq)

1. Redis Settings

This is a configuration example for Redis.

```
# Redis Settings
CARROT_BACKEND = "ghettoq.taproot.Redis"

BROKER_HOST = "localhost"  # Maps to redis host.
BROKER_PORT = 6379         # Maps to redis port.
BROKER_VHOST = "0"         # Maps to database number.

CELERY_RESULT_BACKEND = "redis"
REDIS_HOST = "localhost"
REDIS_PORT = 6379
REDIS_DB = 0
#REDIS_CONNECT_RETRY = True
```

2. Rabbitmq Settings

This is a configuration example for Rabbitmq.

```
BROKER_HOST = "localhost"
BROKER_PORT = 5672
BROKER_USER = "root"
BROKER_PASSWORD = "root"
BROKER_VHOST = "localhost"

CELERY_RESULT_BACKEND = "amqp"
```

3.8.2 Launch celery/celerybeat in debug mode

To run celeryd

```
$ python manage.py celeryd -E -l debug
```

To run celerybeat

```
$ python manage.py celerybeat --schedule=/var/run/celerybeat-schedule
```

To run both

```
$ python manage.py celeryd -E -B -l debug
```

3.8.3 Running celeryd/celerybeat as a daemon (Debian/Ubuntu)

To configure celeryd as a daemon, it is necessary to configure the location of celeryconfig

```
$ cd install/celery-init/etc/default/
```

1. Open celeryd in text editor & change the following variables

   Configuration file: /etc/default/celeryd

   Init script: celeryd.

   Usage : /etc/init.d/celeryd {start|stop|force-reload|restart|try-restart|status}:

```
# Where to chdir at start
CELERYD_CHDIR="/path/to/cdr-stats/"

# Path to celeryd
CELERYD="/path/to/cdr-stats/manage.py celeryd"

# Extra arguments to celeryd
CELERYD_OPTS="--time-limit=300"

# Name of the celery config module.
CELERY_CONFIG_MODULE="celeryconfig"

# Extra Available options
# %n will be replaced with the nodename.
# Full path to the PID file. Default is /var/run/celeryd.pid.
CELERYD_PID_FILE="/var/run/celery/%n.pid"

# Full path to the celeryd log file. Default is /var/log/celeryd.log
CELERYD_LOG_FILE="/var/log/celery/%n.log"

# User/Group to run celeryd as. Default is current user.
# Workers should run as an unprivileged user.
CELERYD_USER="celery"
CELERYD_GROUP="celery"
```

2. Open celeryd (for periodic task) in text editor & add the following variables

   Configuration file: /etc/default/celerybeat or /etc/default/celeryd

   Init script: celerybeat

   Usage: /etc/init.d/celerybeat {start|stop|force-reload|restart|try-restart|status}:

```
# Path to celerybeat
CELERYBEAT="/path/to/cdr-stats/manage.py celerybeat"

# Extra arguments to celerybeat
CELERYBEAT_OPTS="--schedule=/var/run/celerybeat-schedule"
```

3. Copy the configuration file & init scripts to /etc dir:

```
    $ cp etc/default/celeryd /etc/default/

    $ cp etc/init.d/celeryd /etc/init.d/

    $ cp etc/init.d/celerybeat /etc/init.d/
```

4. Run/Start or Stop celery as a daemon:

```
    $ /etc/init.d/celeryd start or stop

    $ /etc/init.d/celerybeat start or stop
```

3.8.4 Troubleshooting

If celeryd will not start as a daemon, try running it in verbose mode:

```
$ sh -x /etc/init.d/celeryd start

$ sh -x /etc/init.d/celerybeat start
```

3.9 ACL Control

One of the benefits of CDR-Stats is ACL access, allowing numerous people to access CDR-Stats each viewing their own CDR with permissions assigned to allow viewing different parts of the interface.

3.9.1 Add Customer

To add a new user, enter the admin screen and Add Customer. Enter a username and password, (twice for authentication), optionally add address details, then enter the accountcode of the customer which corresponds to the accountcode that is delivered in the CDR. When done, click save, and the customer details will be saved and the page reloaded and now displays the user permissions available.

Permissions can be added individually by selecting the permission and then pressing the right arrow to move the permission from the left field to the right field. When done, click save. The permissions to assign to the user are those beginning with user_profile and cdr_alert.

3.9.2 Group Permissions

When you have many customers who are all to have the same permissions, you can add a group, assign the group the desired permissions, then add the customer to the group.

From the admin screens, Click add group, give it a name, assign permissions then save. Finally edit the customer, select the groups to which the customer will belong, then click save. The customer will then inherit permissions from their group.

CHAPTER 4

Celery

4.1 Celery Installation

4.1.1 Celery

Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well.

You can install Celery either via the Python Package Index (PyPI) or from source:

```
$ pip install celery
```

Downloading and installing from source

To Download the latest version click here.

You can install it by doing the following:

```
$ tar xvfz celery-X.X.X.tar.gz

$ cd celery-X.X.X

$ python setup.py build

$ python setup.py install # as root
```

Using the development version

You can clone the repository by doing the following:

```
$ git clone git://github.com/ask/celery.git
```

CHAPTER 5

Troubleshooting

5.1 Where to find the log files

All the logs are centralized into one single directory **/var/log/cdr-stats/**

cdr-stats.log : All the logger events from Django

cdr-stats-db.log : This contains all the Database queries performed by the UI

gunicorn_cdr_stats.log : All the logger events from Gunicorn

djcelery_error.log : This contains celery activity

djcelerybeat_error.log : This contains celerybeat activity

5.2 Run in debug mode

Make sure services are stopped first:

```
$ /etc/init.d/supervisor stop
```

Then run in debug mode:

```
$ workon cdr-stats
$ cd /usr/share/cdrstats/
$ python manage.py celeryd -EB --loglevel=DEBUG
```

5.3 Celerymon

- https://github.com/ask/celerymon

Running the monitor :

Start celery with the –events option on, so celery sends events for celerymon to capture:

```
$ workon cdr-stats
$ cd /usr/share/cdrstats/
$ python manage.py celeryd -E
```

Run the monitor server:

```
$ workon cdr-stats
$ cd /usr/share/cdrstats/
$ python manage.py celerymon
```

However, in production the monitor is best run in the background as a daemon:

```
$ workon cdr-stats
$ cd /usr/share/cdrstats/
$ python manage.py celerymon --detach
```

For a complete listing of the command line arguments available, with a short description, use the help command:

```
$ workon cdr-stats
$ cd /usr/share/cdrstats/
$ python manage.py help celerymon
```

Visit the webserver celerymon stats by going to: http://localhost:8989

CHAPTER 6

User Guide

6.1 Overview

CDR-Stats is a web based application built on a Django Web framework which uses PostgreSQL as the CDR data store.

Celery (http://celeryproject.org/) is an asynchronous task queue/job queue based on distributed message. It is used to build the backend system to monitor CDR, detect unusual activity, and react by sending an alert email.

CDR Stats Management Features

- CDR Mediation
- CDR Rating
- Multi-tenant design that allows call detail records from multiple switches or PBX systems.
- Custom alarm triggers can be set to email the administrator for a range of conditions including unusual average call durations, failed calls, and unexpected destinations called.
- Graphical tools help detect unusual call patterns which may indicate suspicious or fraudulent activity.
- Import Call Detail Records in CSV format
- Configure Switches for import
- Create Customer and assign accountcode
- Configure alert to detect unsual increase/decrease of Traffic

CDR Stats Customer Portal Features

- Password management
- Call Details Record
- Monthly, Daily, Hourly Call reporting
- Impact Reporting
- Country Reporting
- Realtime Reporting of calls in progress
- View Fraudulent Calls
- Concurrent Call Statistic
- Configure Mail Reporting

- Top 10 destination Traffic
- Export to CSV
- Automated daily reporting.
- Call cost reports

6.1.1 How to use CDR-Stats

CDR-Stats has two main areas, the admin screen and the customer portal. The admin and customer areas are described in detail in the following pages.

CDR-Stats has been designed to be responsive, that is to say the the layout changes depending on the size and resolution of the browser viewing the pages.

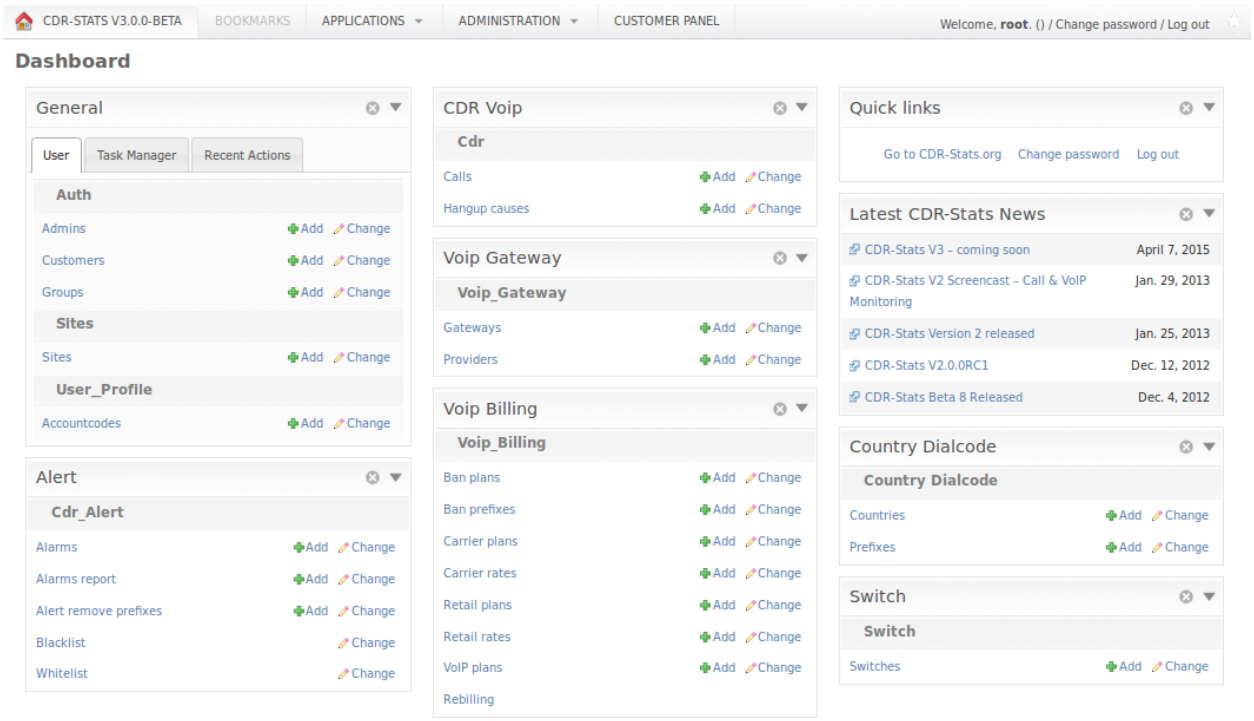
Admin Panel

The Admin section allows you to create administrators who have access the admin screens. Levels of access can be set.

The Admin UI is located at http://localhost:8000/admin/

Dashboard

Dashboard page for the admin interface after successful login with superuser credentials



CDR Manual Import / Export

There is a tool in CDR-Stats to manually import CDRs from CSV, JSON, Excel & YAML. The first line of the import file should contain the following header names "switch,cdr_source_type,callid,caller_id_number,caller_id_name,destination_number,dialcode,state,channel,starting_date,duration,bills

Example of a CDR csv file to import:

```
switch,cdr_source_type,callid,caller_id_number,caller_id_name,destination_number,dialcode,state,chann
127.0.0.1,1,96aa82fe-7bd1-11e5-a230-5c514f6a0f72,904151440,CallerIDName,+34798400122,34,,,2015-10-21
127.0.0.1,1,c9135e4a-7bd1-11e5-a230-5c514f6a0f72,904234320,CallerIDName,+34798401111,34,,,2015-10-21
127.0.0.1,1,cfaf8b56-7bd1-11e5-a230-5c514f6a0f72,901110380,CallerIDName,+34650104877,34,,,2015-10-21
127.0.0.1,1,3c64a168-7bd2-11e5-a230-5c514f6a0f72,904234320,CallerIDName,+34798401111,34,,,2015-10-21
127.0.0.1,1,41b20dd9-7bd2-11e5-a230-5c514f6a0f72,904231111,CallerIDName,+34650104877,34,,,2015-10-21
```

Note:

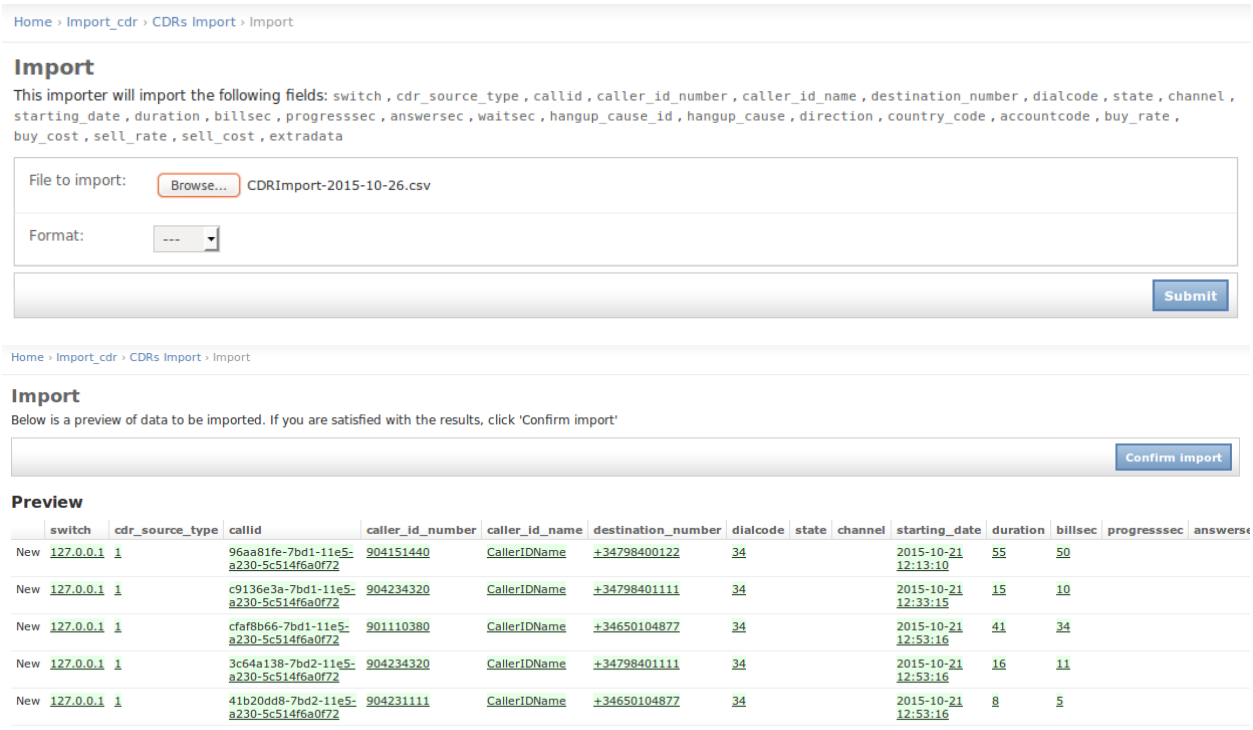
```
- cdr_source_type is an integer to define from where the CDR comes from (UNKNOWN = 0, CSV = 1, API =

- extradata is a JSON field, if empty you have to set it as `{}`
```

From CDR Import admin page, you will also be able to export your CDRs to CSV, JSON, HTML, ODS, Excel & YAML.

URL:

- http://localhost:8000/admin/import_cdr/cdrimport/import/
- http://localhost:8000/admin/import_cdr/cdrimport/export/



Alarm

The alarm list will be displayed from the following URL. You can add a new alarm by clicking `Add alarm` and adding the name of the alarm and its description, Also from the alarm list, click on the alarm that you want to update.

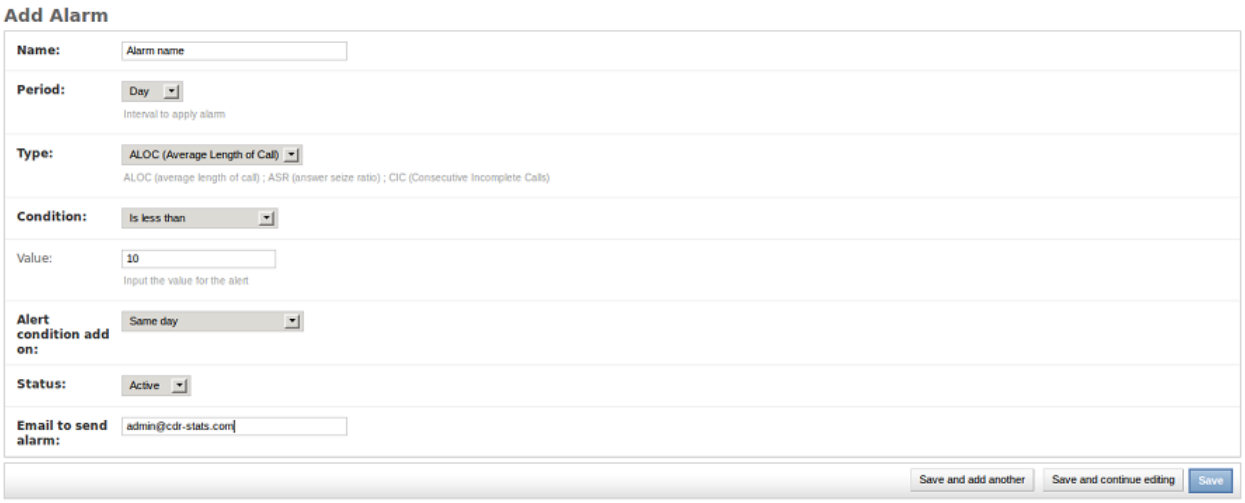
URL:

- http://localhost:8000/admin/cdr_alert/alarm/

To Add/Update alarm

URL:

- http://localhost:8000/admin/cdr_alert/alarm/add/
- http://localhost:8000/admin/cdr_alert/alarm/1/



Alarm-report

The alarmreport will be displayed from the following URL.

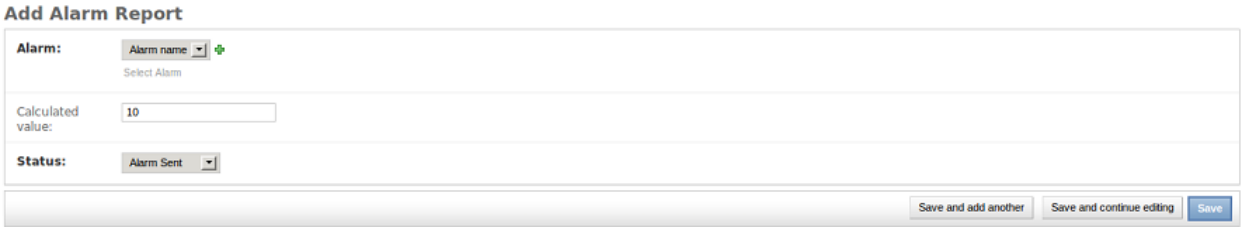
URL:

- http://localhost:8000/admin/cdr_alert/alarmreport/

To Add/Update alarmreport

URL:

- http://localhost:8000/admin/cdr_alert/alarmreport/add/
- http://localhost:8000/admin/cdr_alert/alarmreport/1/

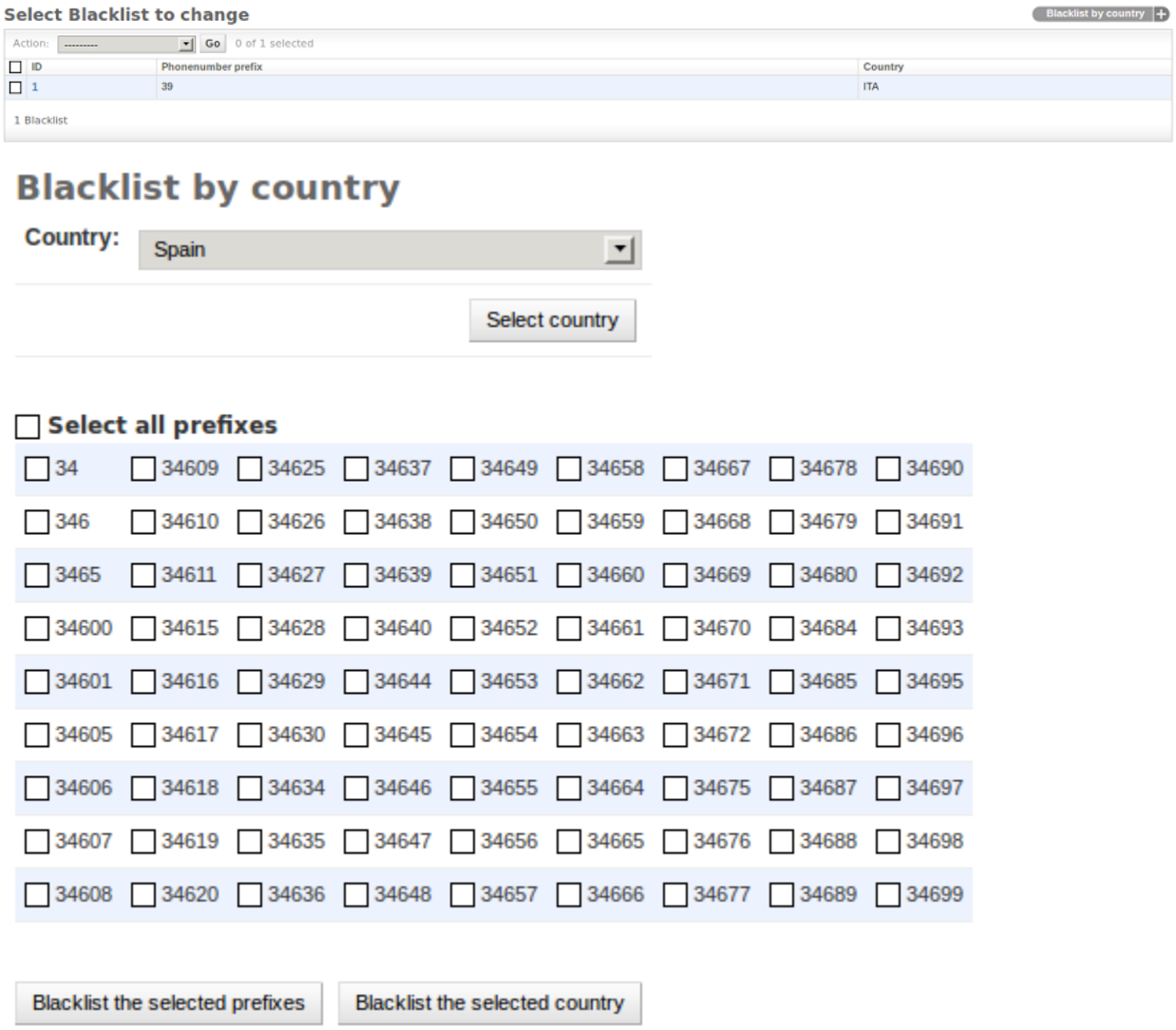


Blacklist

The blacklist will be displayed from the following URL. You can add a new blacklist by clicking `Blacklist by country` and selecting the country name and its prefixes, Also from the blacklist, click on the blacklist that you want to update.

URL:

- http://localhost:8000/admin/cdr_alert/blacklist/



Whitelist

The whitelist will be displayed from the following URL. You can add a new Whitelist by clicking `Whitelist by country` and selecting the country name and its prefixes, Also from the whitelist, click on the blacklist that you want to update.

URL:

- http://localhost:8000/admin/cdr_alert/whitelist/

Alert-remove-prefix

The alert remove prefix will be displayed from the following URL. You can add a new remove prefix by clicking `Add alert remove prefix` and selecting the remove prefix, Also from the alert remove prefix, click on the remove prefix that you want to update.

The Admin UI is located at http://localhost:8000/

URL:

- http://localhost:8000/admin/cdr_alert/alertremoveprefix/

To Add/Update alert-remove prefix

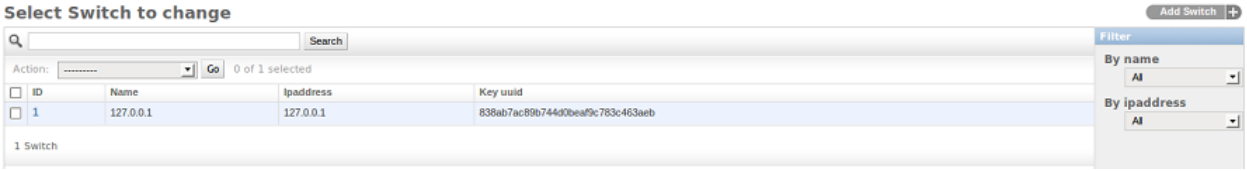
URL:

- http://localhost:8000/admin/cdr_alert/alertremoveprefix/add/
- http://localhost:8000/admin/cdr_alert/alertremoveprefix/1/

Switch

URL:

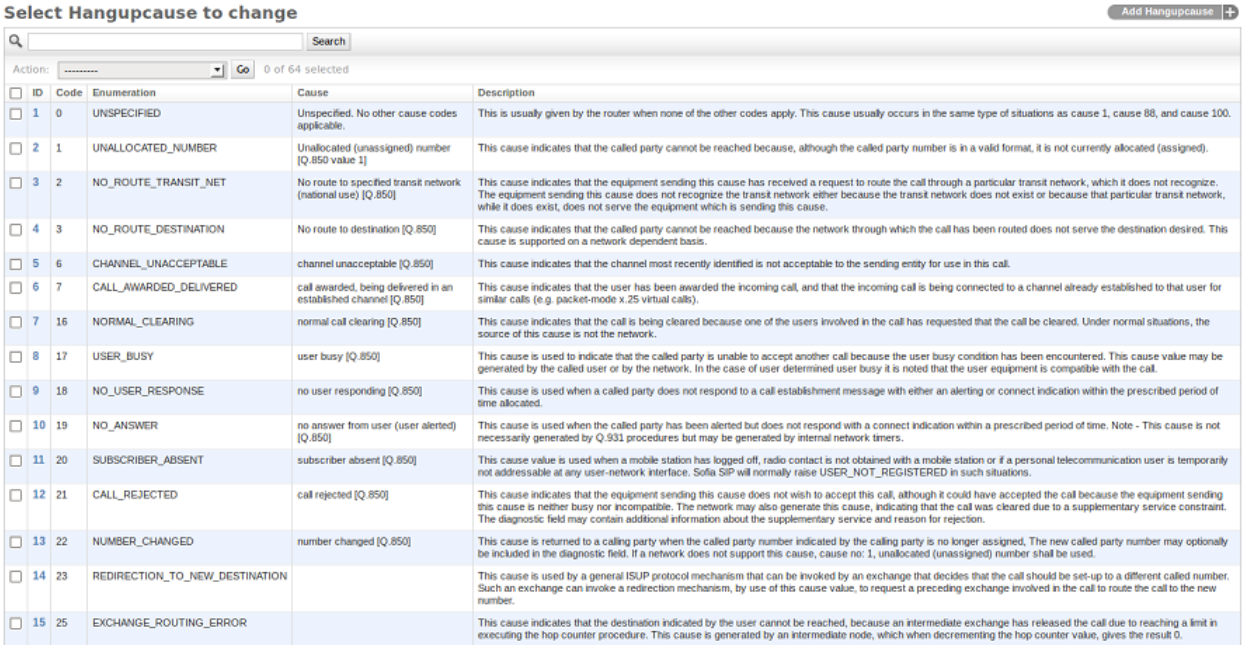
- http://localhost:8000/admin/cdr/switch/



HangupCause

URL:

- http://localhost:8000/admin/cdr/hangupcause/

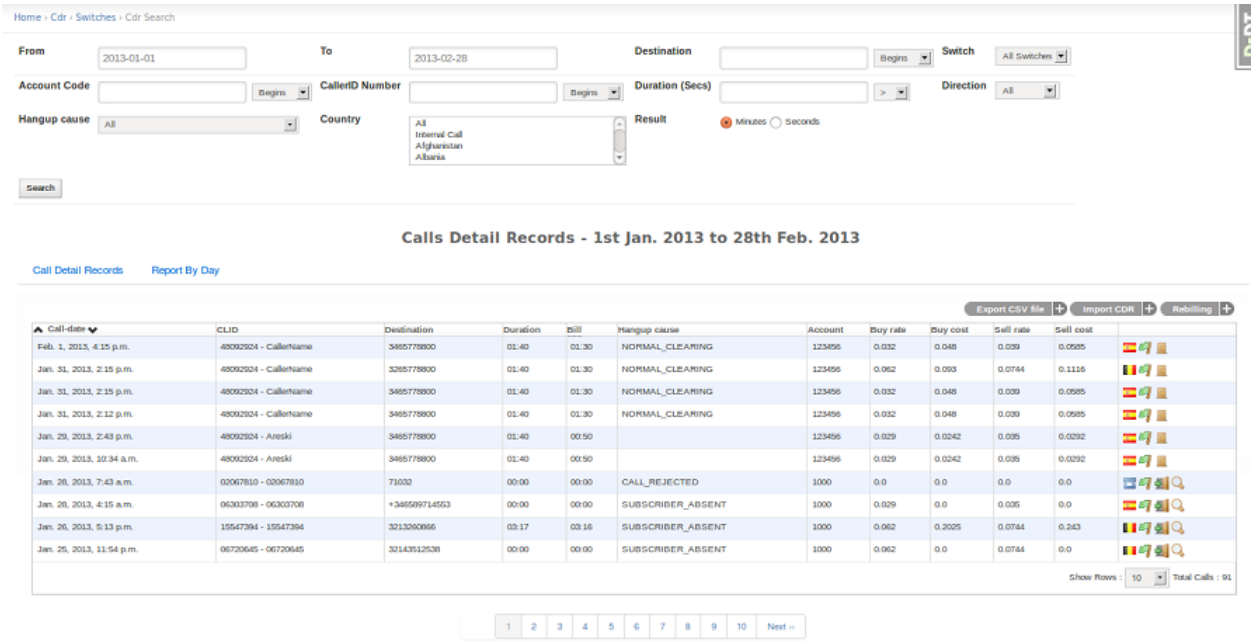


ID	Code	Enumeration	Cause	Description
1	0	UNSPECIFIED	Unspecified. No other cause codes applicable.	This is usually given by the router when none of the other codes apply. This cause usually occurs in the same type of situations as cause 1, cause 88, and cause 100.
2	1	UNALLOCATED_NUMBER	Unallocated (unassigned) number [Q.850 value 1]	This cause indicates that the called party cannot be reached because, although the called party number is in a valid format, it is not currently allocated (assigned).
3	2	NO_ROUTE_TRANSIT_NET	No route to specified transit network (national use) [Q.850]	This cause indicates that the equipment sending this cause has received a request to route the call through a particular transit network, which it does not recognize. The equipment sending this cause does not recognize the transit network either because the transit network does not exist or because that particular transit network, while it does exist, does not serve the equipment which is sending this cause.
4	3	NO_ROUTE_DESTINATION	No route to destination [Q.850]	This cause indicates that the called party cannot be reached because the network through which the call has been routed does not serve the destination desired. This cause is supported on a network dependent basis.
5	6	CHANNEL_UNACCEPTABLE	channel unacceptable [Q.850]	This cause indicates that the channel most recently identified is not acceptable to the sending entity for use in this call.
6	7	CALL_AWARDED_DELIVERED	call awarded, being delivered in an established channel [Q.850]	This cause indicates that the user has been awarded the incoming call, and that the incoming call is being connected to a channel already established to that user for similar calls (e.g. packet-mode x.25 virtual calls).
7	16	NORMAL_CLEARING	normal call clearing [Q.850]	This cause indicates that the call is being cleared because one of the users involved in the call has requested that the call be cleared. Under normal situations, the source of this cause is not the network.
8	17	USER_BUSY	user busy [Q.850]	This cause is used to indicate that the called party is unable to accept another call because the user busy condition has been encountered. This cause value may be generated by the called user or by the network. In the case of user determined user busy it is noted that the user equipment is compatible with the call.
9	18	NO_USER_RESPONSE	no user responding [Q.850]	This cause is used when a called party does not respond to a call establishment message with either an alerting or connect indication within the prescribed period of time allocated.
10	19	NO_ANSWER	no answer from user (user alerted) [Q.850]	This cause is used when the called party has been alerted but does not respond with a connect indication within a prescribed period of time. Note - This cause is not necessarily generated by Q.931 procedures but may be generated by internal network timers.
11	20	SUBSCRIBER_ABSENT	subscriber absent [Q.850]	This cause value is used when a mobile station has logged off, radio contact is not obtained with a mobile station or if a personal telecommunication user is temporarily not addressable at any user-network interface. Sofia SIP will normally raise USER_NOT_REGISTERED in such situations.
12	21	CALL_REJECTED	call rejected [Q.850]	This cause indicates that the equipment sending this cause does not wish to accept this call, although it could have accepted the call because the equipment sending this cause is neither busy nor incompatible. The network may also generate this cause, indicating that the call was cleared due to a supplementary service constraint. The diagnostic field may contain additional information about the supplementary service and reason for rejection.
13	22	NUMBER_CHANGED	number changed [Q.850]	This cause is returned to a calling party when the called party number indicated by the calling party is no longer assigned, The new called party number may optionally be included in the diagnostic field. If a network does not support this cause, cause no: 1, unallocated (unassigned) number shall be used.
14	23	REDIRECTION_TO_NEW_DESTINATION		This cause is used by a general ISUP protocol mechanism that can be invoked by an exchange that decides that the call should be set-up to a different called number. Such an exchange can invoke a redirection mechanism, by use of this cause value, to request a preceding exchange involved in the call to route the call to the new number.
15	25	EXCHANGE_ROUTING_ERROR		This cause indicates that the destination indicated by the user cannot be reached, because an intermediate exchange has released the call due to reaching a limit in executing the hop counter procedure. This cause is generated by an intermediate node, which when decrementing the hop counter value, gives the result 0.

CDR View

URL:

- http://localhost:8000/admin/cdr/switch/cdr_view/



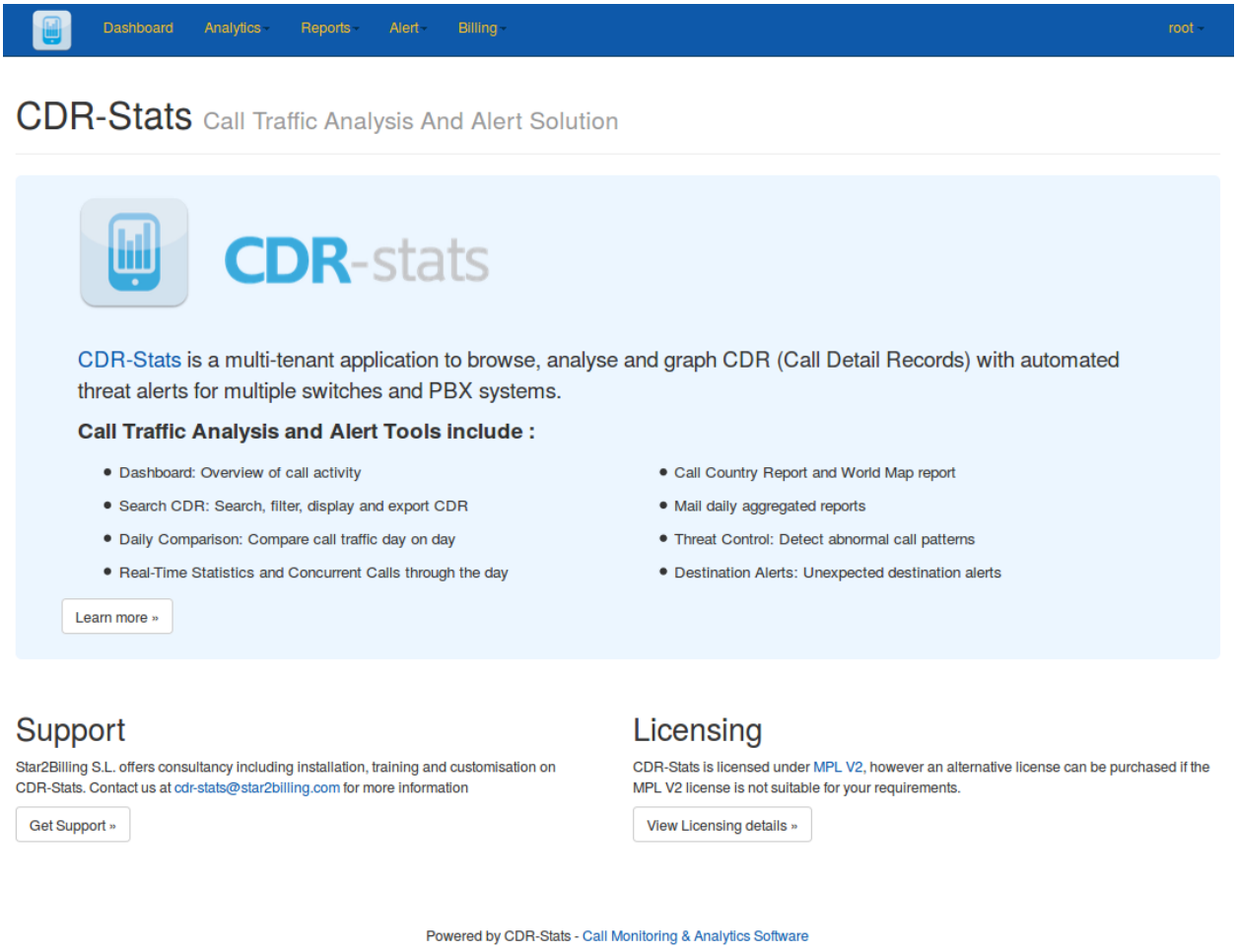
User Panel

The User Interface is the core part of CDR-Stats, this is the one that the users will use to get reporting and take advantage of CDR-Stats capabilities and features.

The User UI is located at http://localhost:8000/

Index

Index page for the customer interface after successful login with user credentials

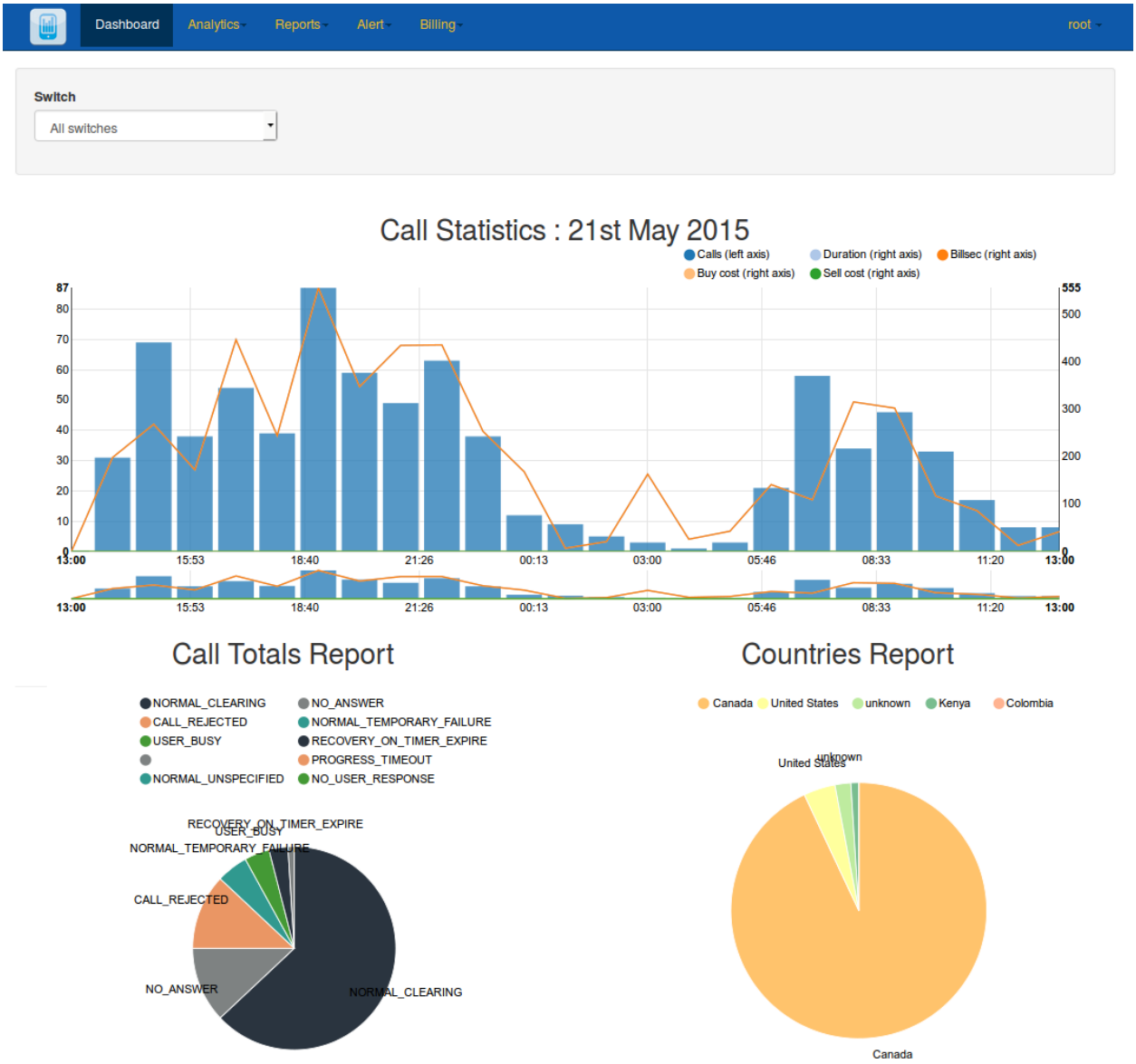


Dashboard

The dashboard displays a graphical representation of the last 24 hours calls, call status statistics and calls by country, either agregrated for all switches, or selectable by switch.

URL:

- http://localhost:8000/dashboard/



CDR-View

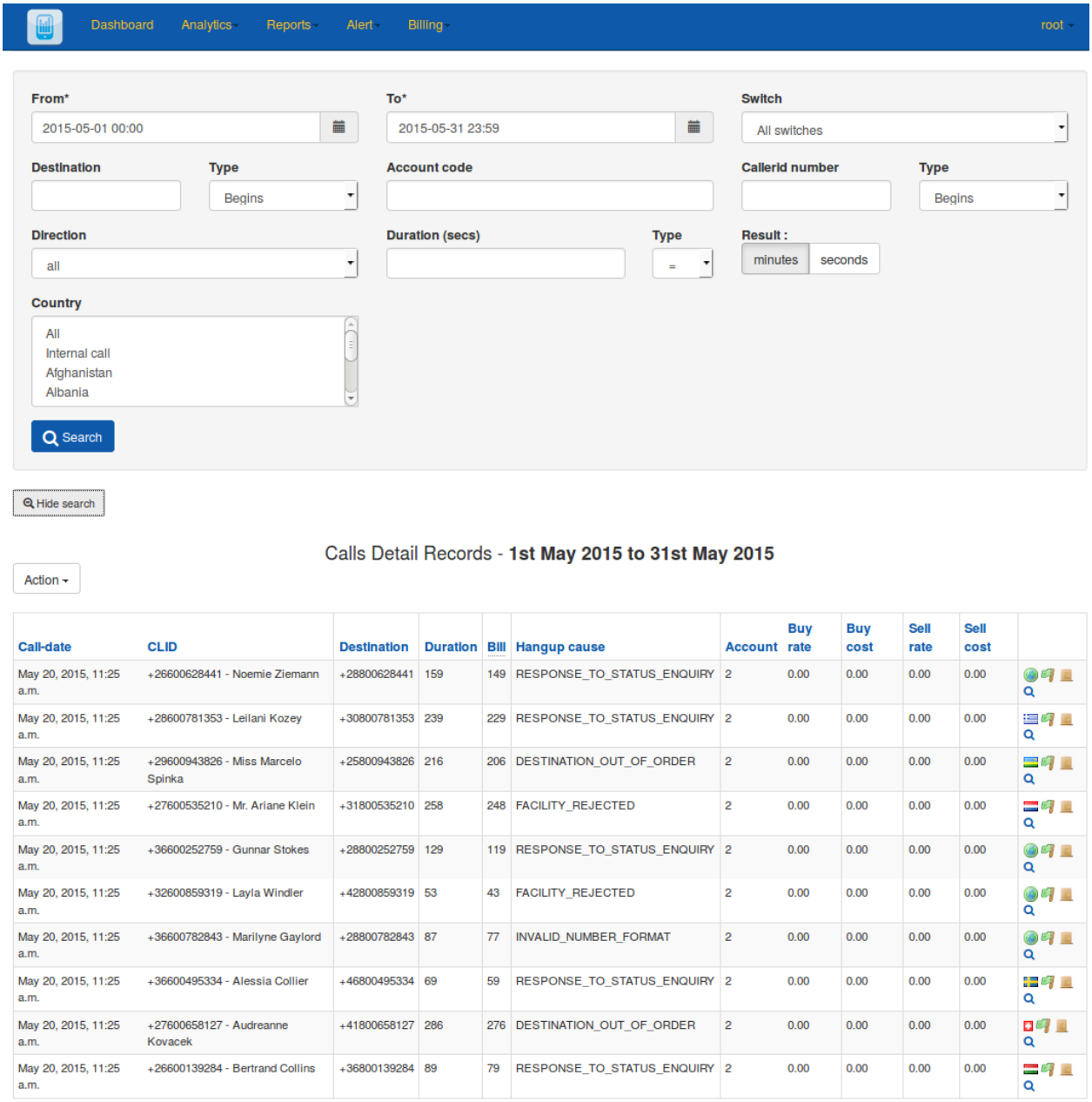
Call detail records listed in table format which can be exported to CSV file.

Advanced Search allows further filtering and searching on a range of criteria

The Report by Day shows a graphical illustration of the calls, minutes and average call time.

URL:

- http://localhost:8000/cdr_view/

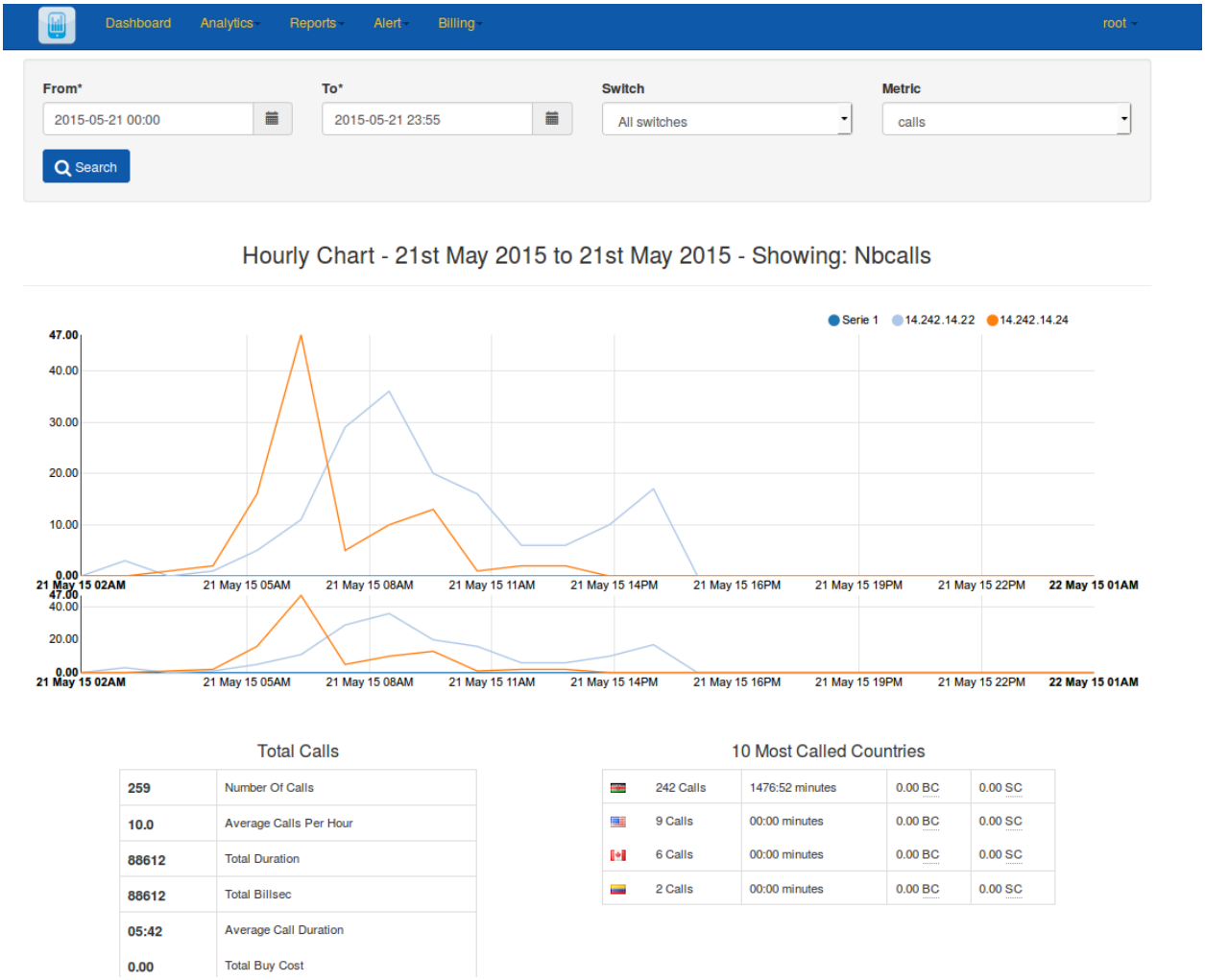


CDR-Overview

A pictorial view of calls with call-count or call-duration from any date or date-range

URL:

- http://localhost:8000/cdr_overview/

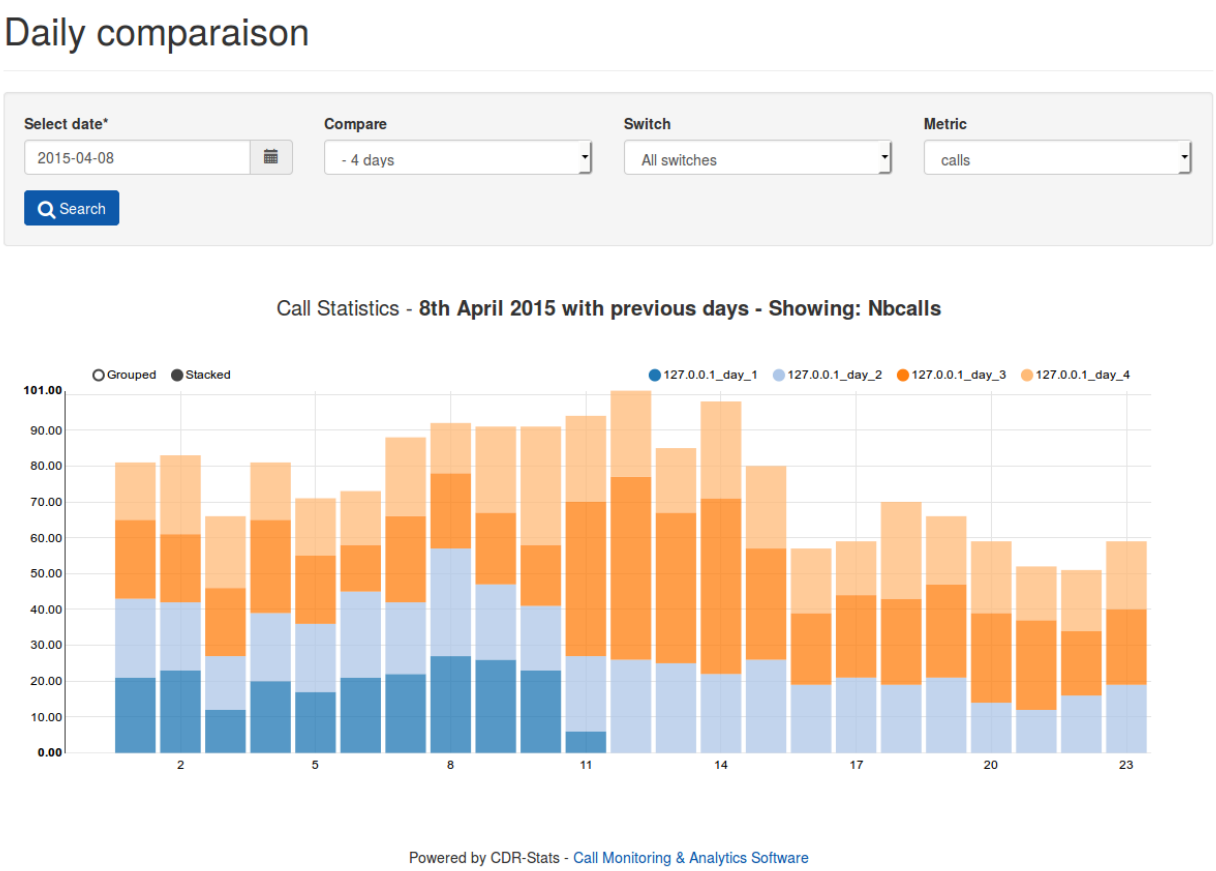


CDR-Hourly-Report

An hourly pictorial view of calls with call-count & call-duration. You can compare different dates

URL:

- http://localhost:8000/hourly_report/

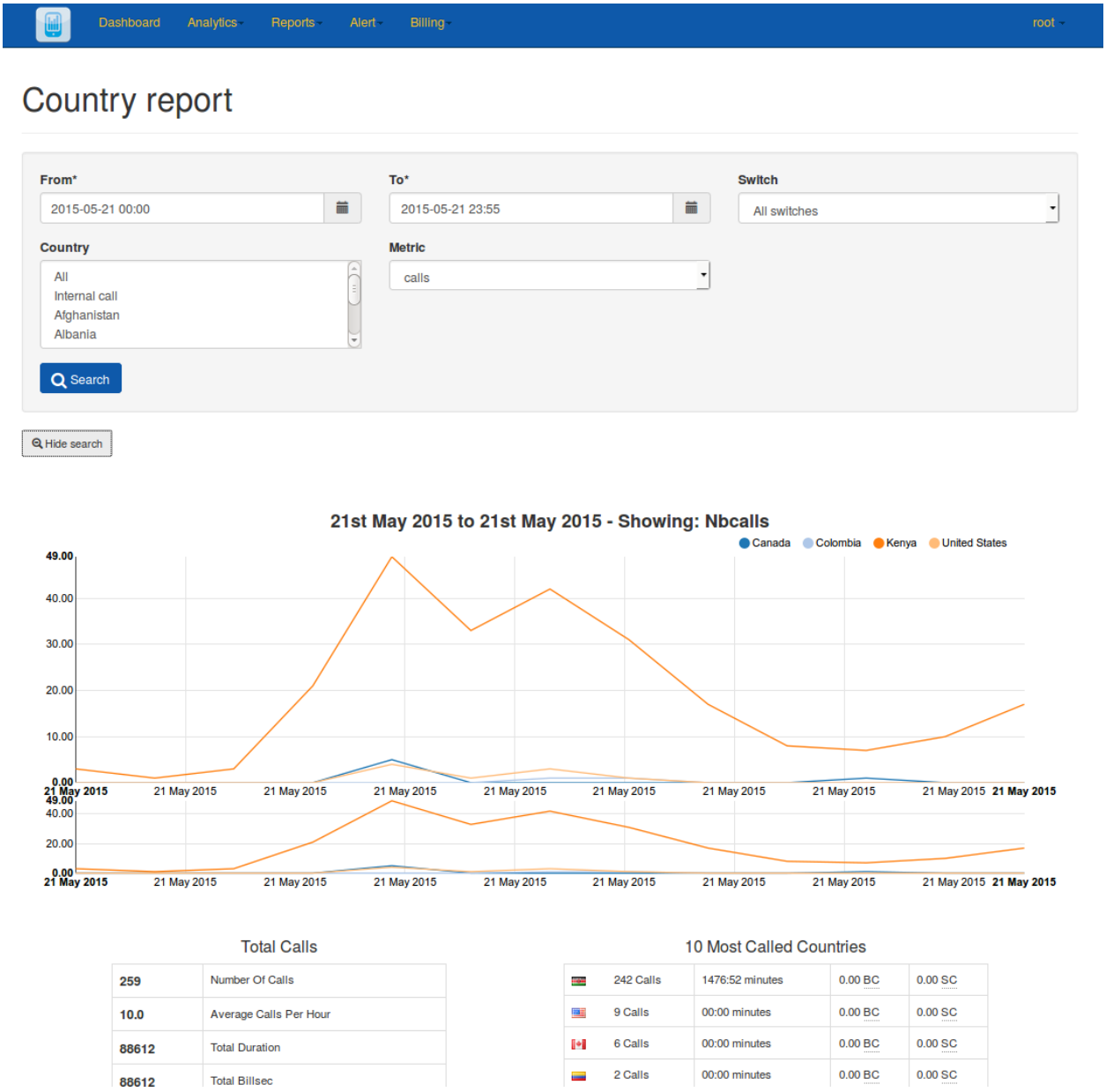


CDR-Country-Report

A pictorial view of all calls by country with the 10 most called countries in a pie chart.

URL:

- http://localhost:8000/country_report/



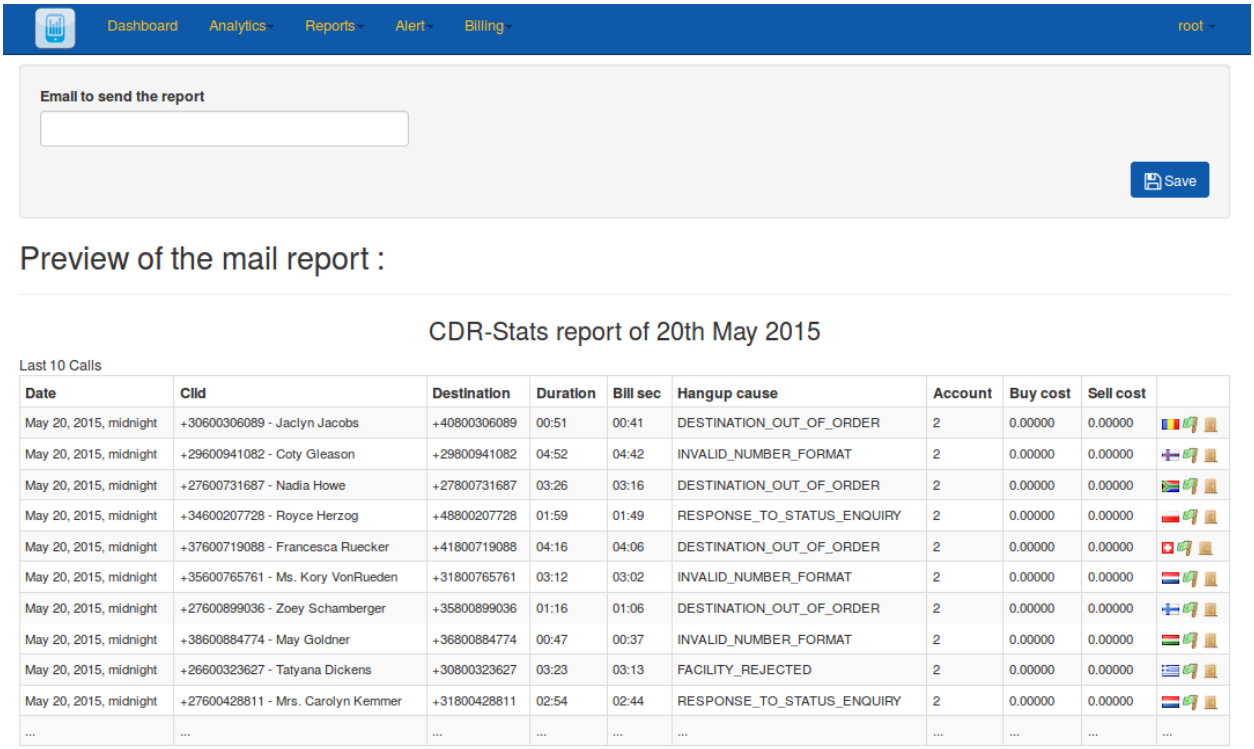
Mail-Report

A list of the last 10 calls of the previous day, along with total calls, a breakdown of the call status, and the top 5 countries called.

This report is emailed automatically, email recipients can be set up in the admin section or by adding an email address in the "Email to send a report" field in the Mail Report section.

URL:

- http://localhost:8000/mail_report/

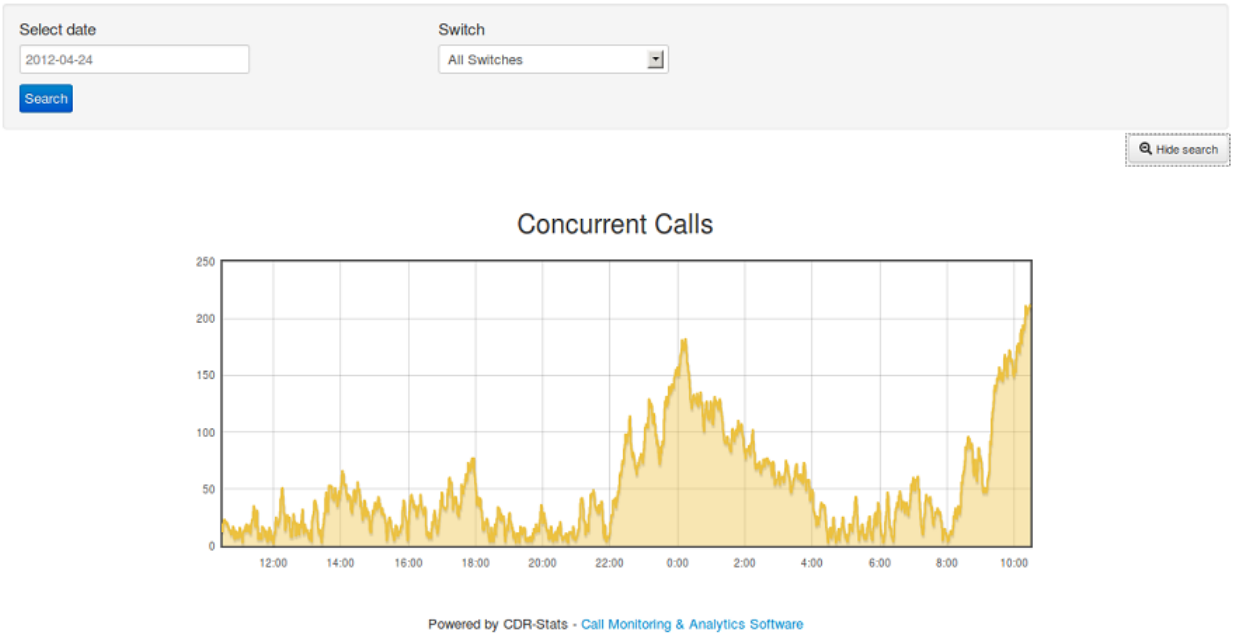


Concurrent-call-report

A report of concurrent calls. The statistics are collated from the realtime report, not from the CDR.

URL:

- http://localhost:8000/cdr_concurrent_calls/

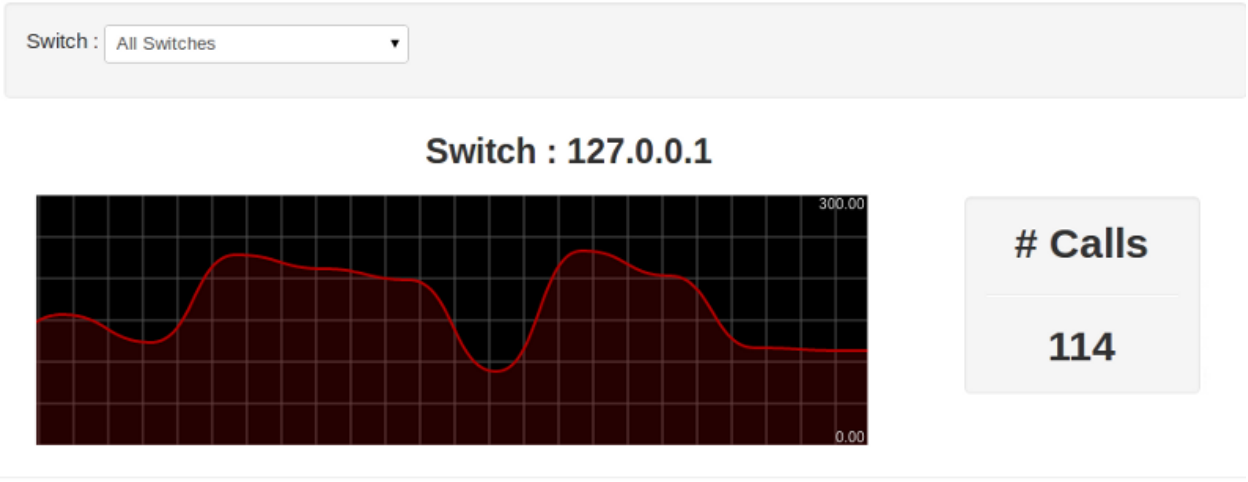


Realtime-Report

Realtime monitoring of the traffic on the connected telecoms servers, Freeswitch and Asterisk are supported.

URL:

- http://localhost:8000/cdr_realtime/

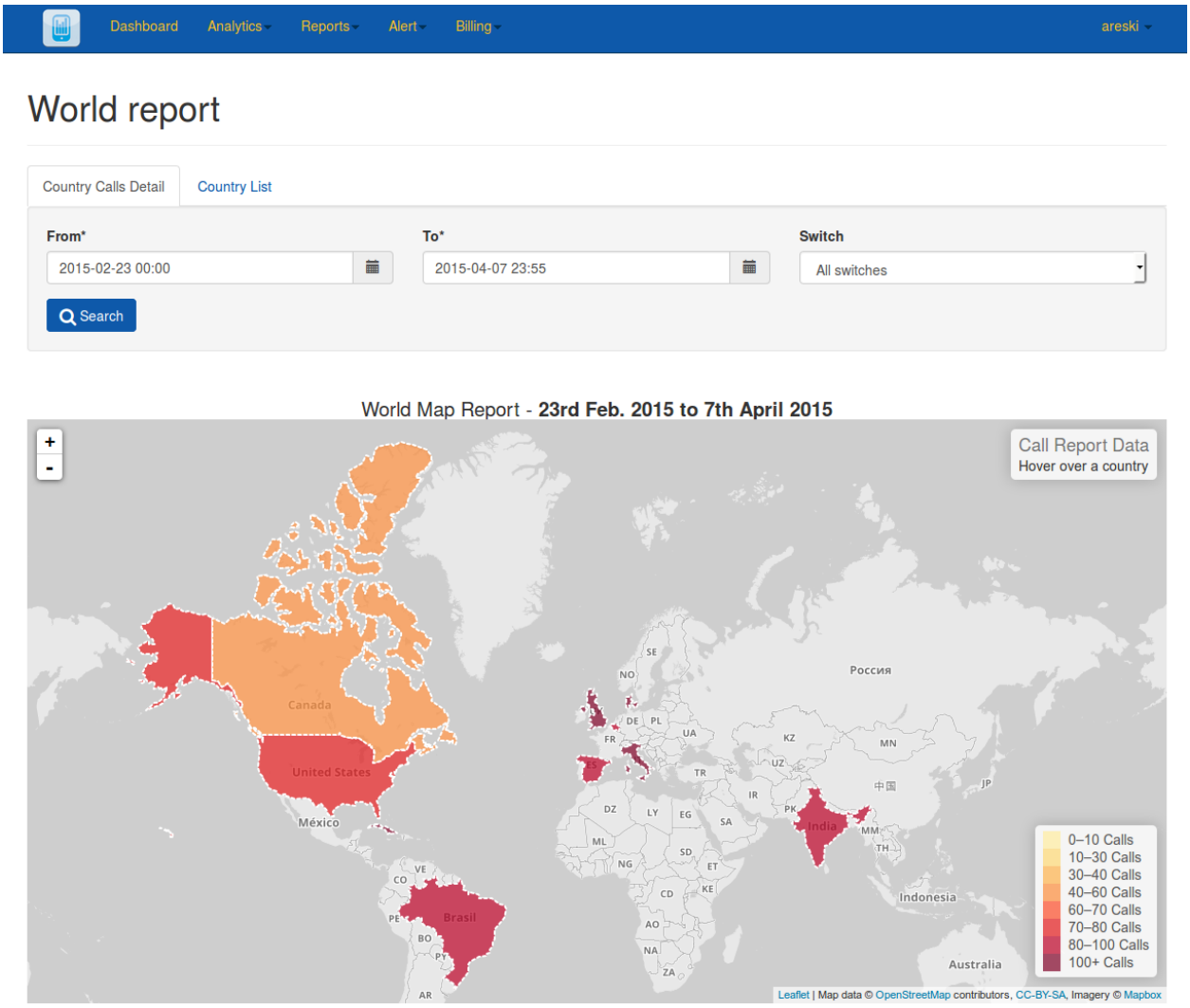


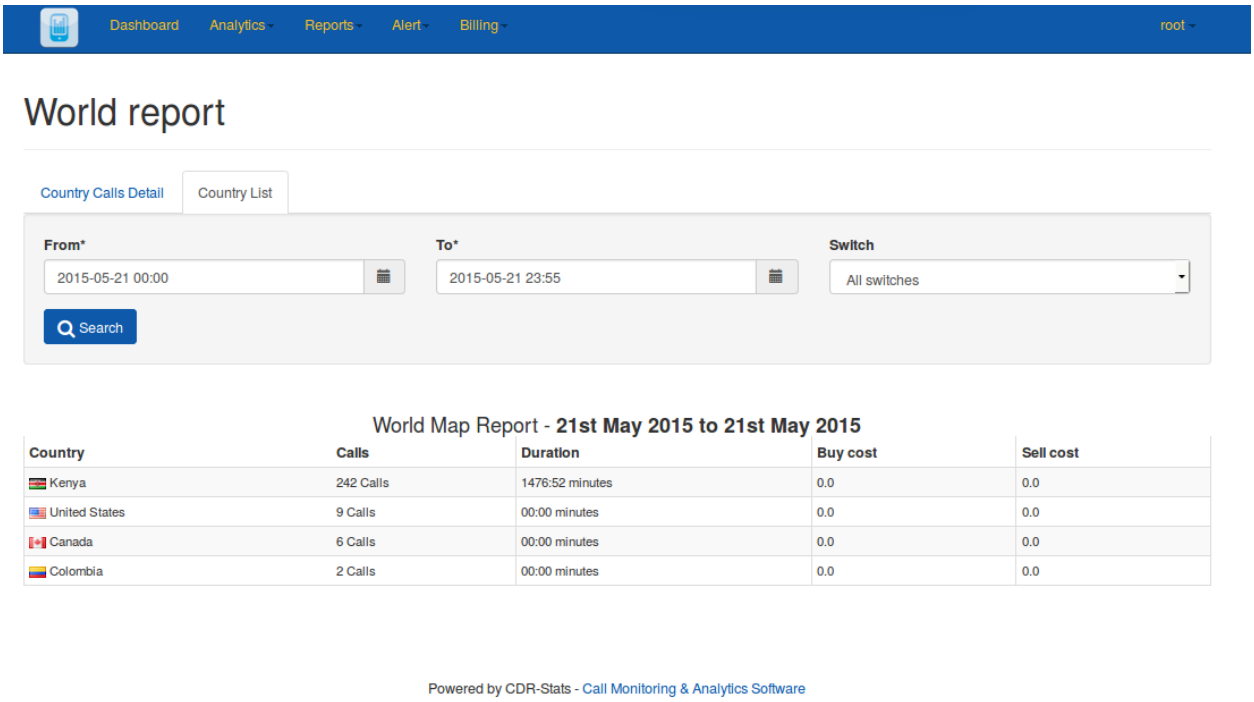
World Map Report

A distriibution map of all calls / durations by country. You can select date criteria and on mouse over on the world map you can get information about each country.

URL:

- http://localhost:8000/world_map/

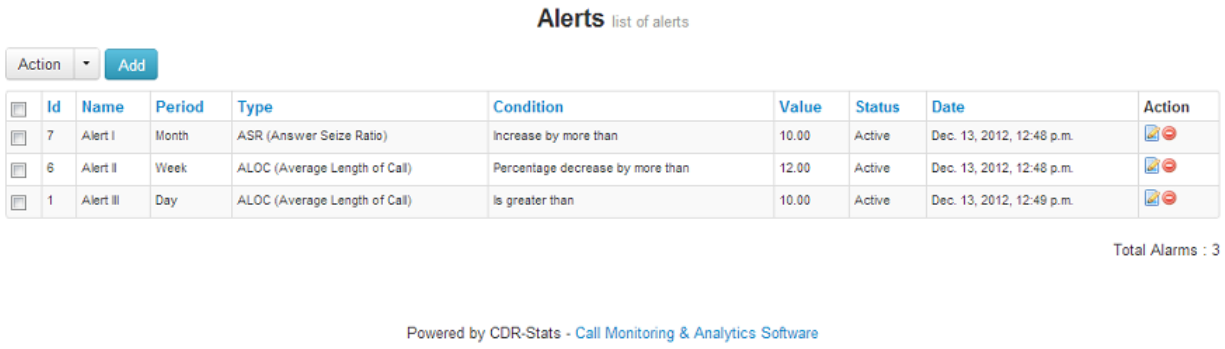

Dashboard
Analytics
Reports
Alert
Billing
areski
World report
Country Calls Detail
Country List
From*
2015-02-23 00:00
To*
2015-04-07 23:55
Switch
All switches
Search
World Map Report - 23rd Feb. 2015 to 7th April 2015
Call Report Data
Hover over a country
0–10 Calls
10–30 Calls
30–40 Calls
40–60 Calls
60–70 Calls
70–80 Calls
80–100 Calls
100+ Calls
Canada
United States
México
Brasil
Россия
India
Indonesia
Australia
Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA, Imagery © Mapbox



Alert Settings

URL:

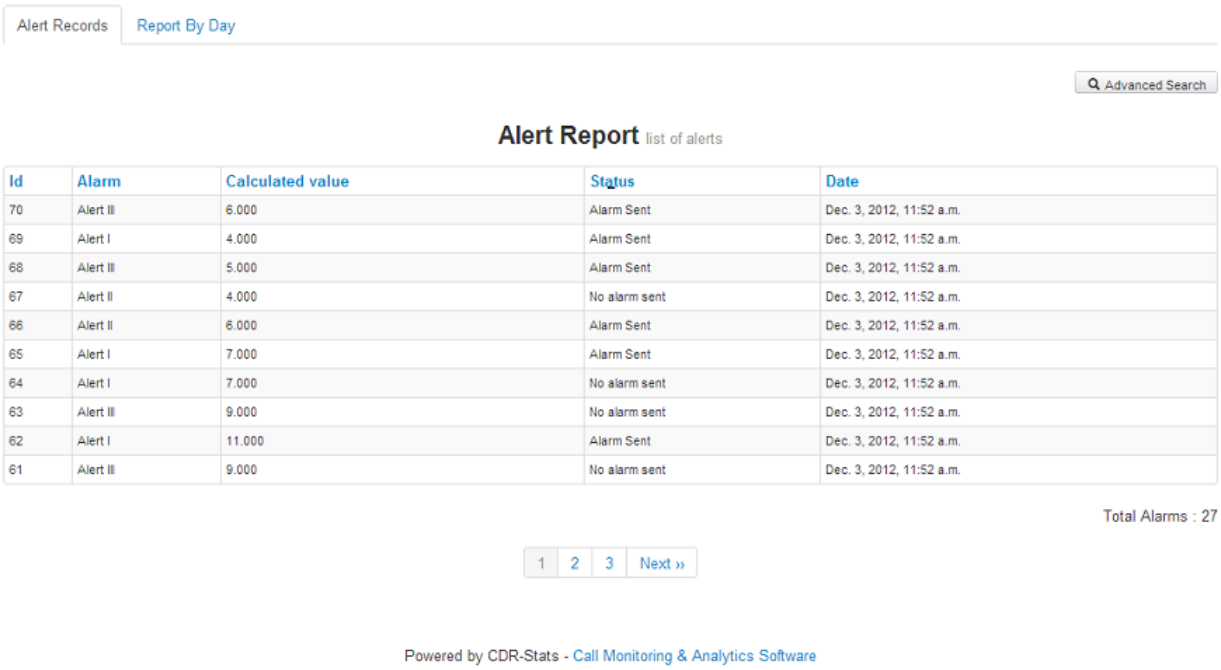
- http://localhost:8000/alert/



Alert Report

URL:

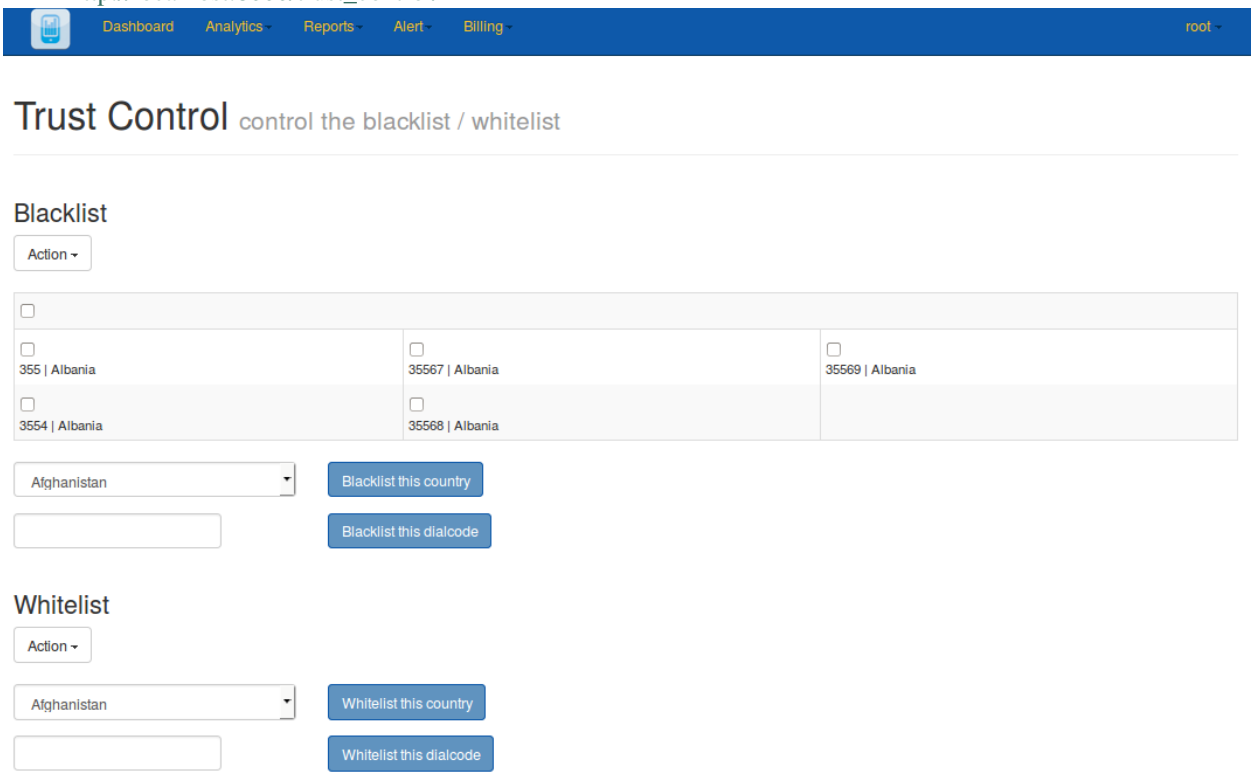
- http://localhost:8000/alert_report/



Destination Control

URL:

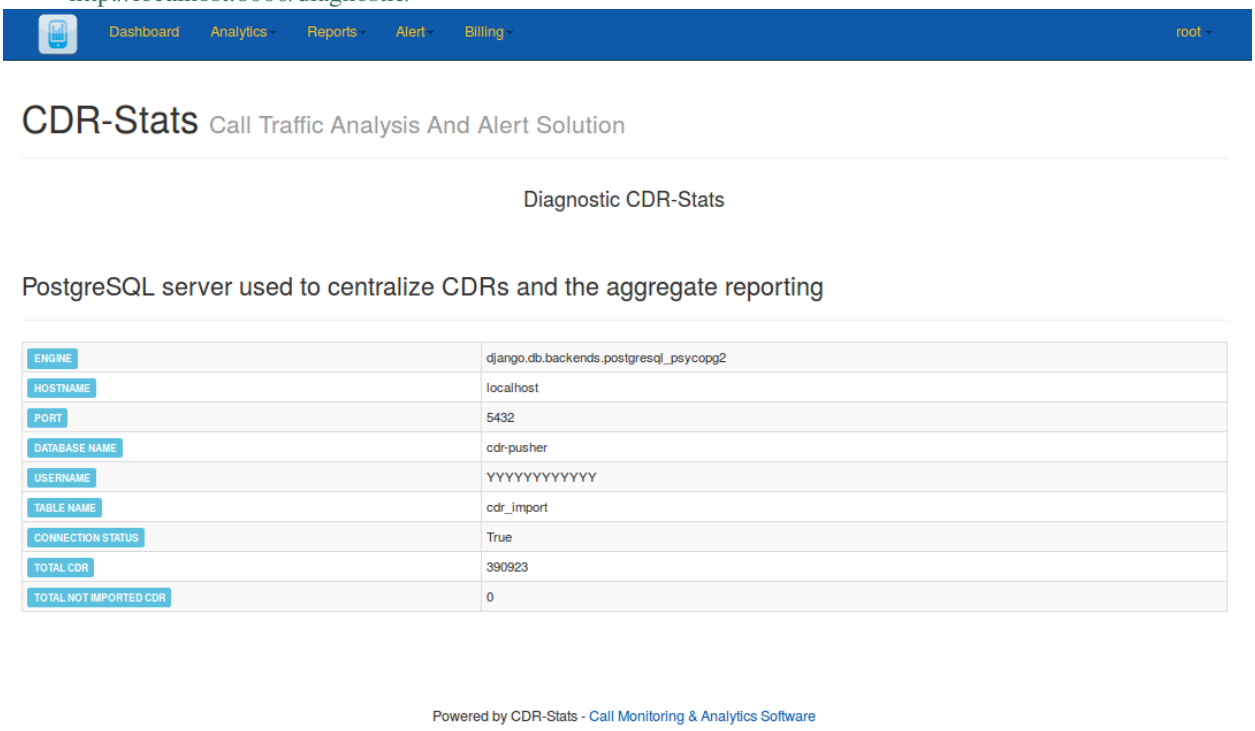
- http://localhost:8000/trust_control/



Diagnostic CDR-Stats

URL:

- http://localhost:8000/diagnostic/

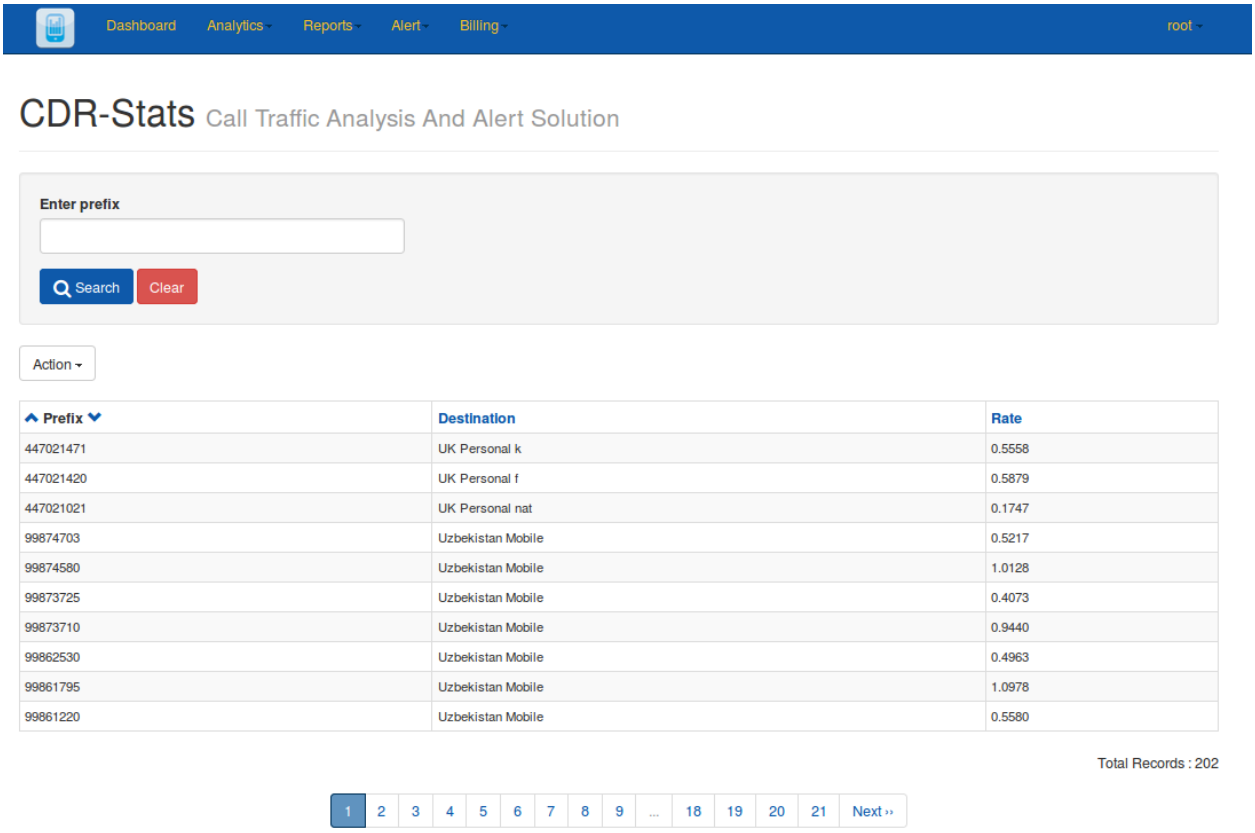


Rates

voip call rates.

URL:

- http://localhost:8000/rates/

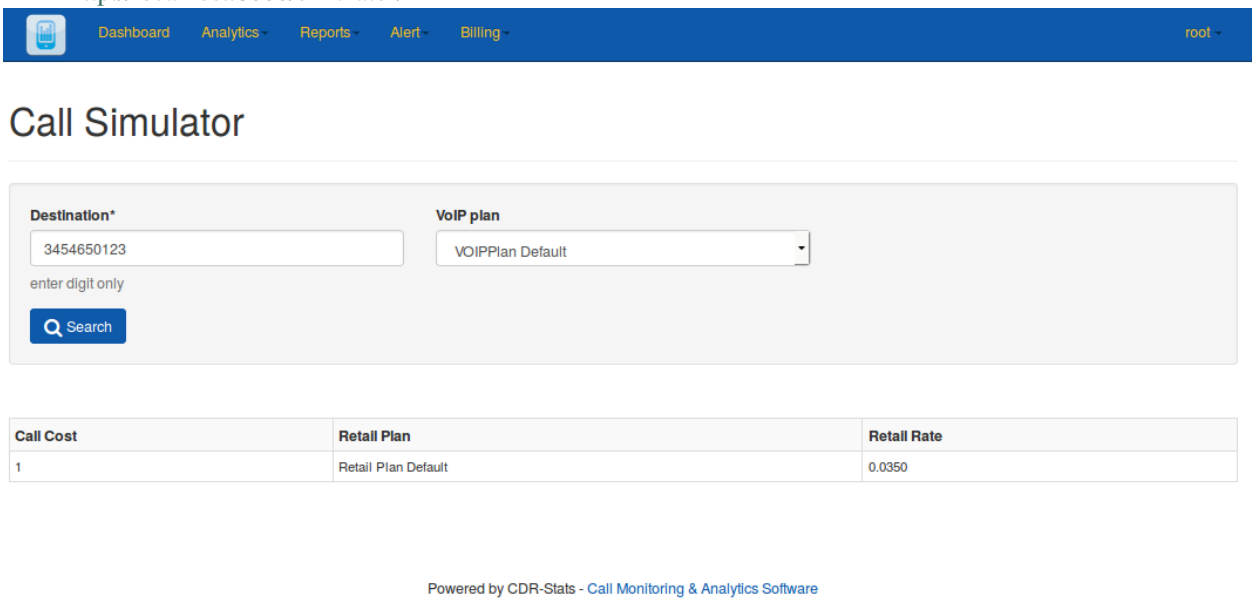


Call Simulator

voip call simulator.

URL:

- http://localhost:8000/simulator/

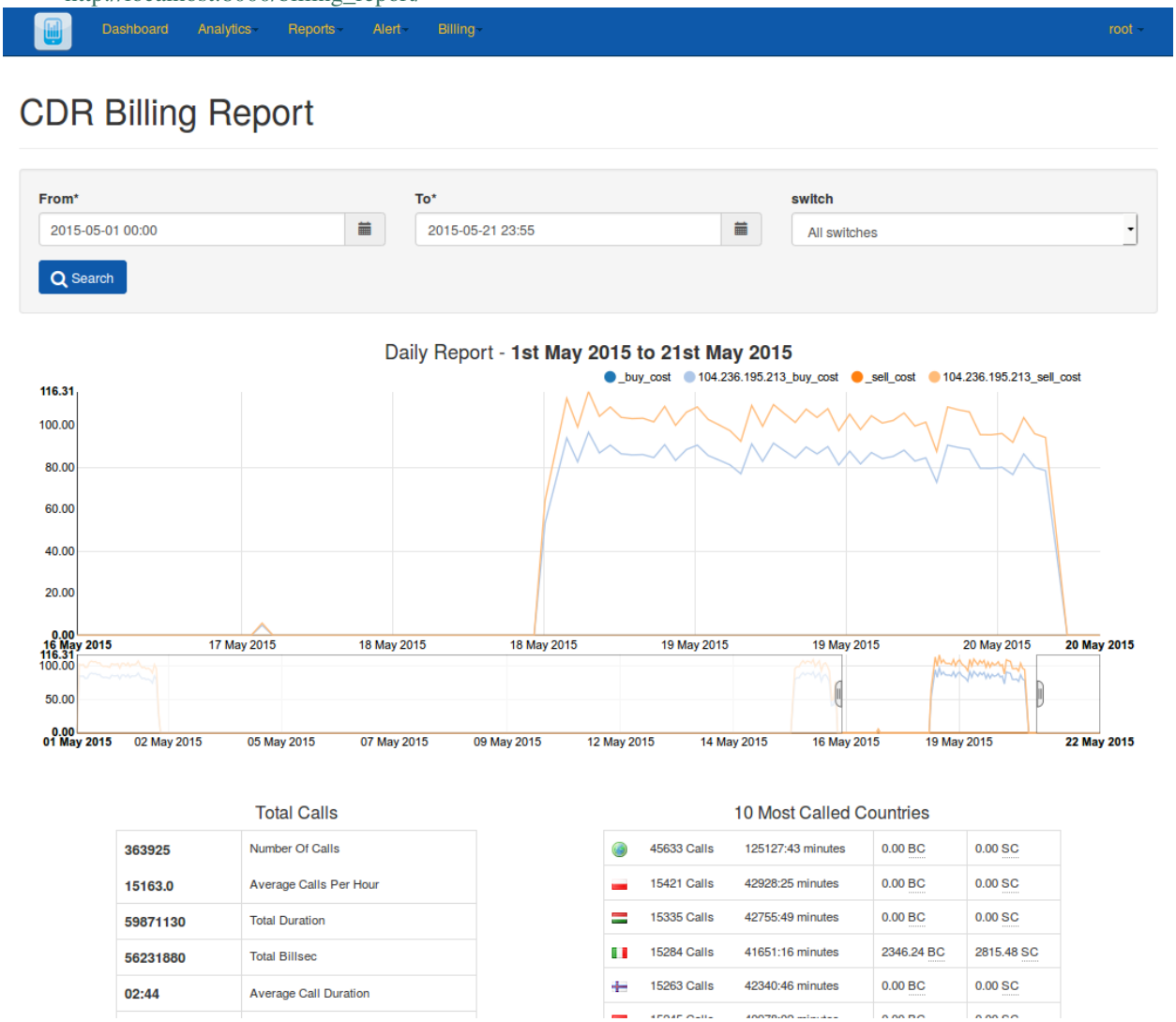


Daily report of Billed call

Daily report of Billed call.

URL:

- http://localhost:8000/billing_report/



CHAPTER 7

PostgreSQL

Web http://www.postgresql.org/

–

PostgreSQL is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards-compliance.

PostgreSQL provides few interesting features that make it a perfect choice for CDR-Stats:

- Materialized view (http://www.postgresql.org/docs/9.4/static/rules-materializedviews.html), these views contains the results of queries, it's ideal for aggregation views, they also can be refreshed since PG 9.4 without locking.
- Json Types (http://www.postgresql.org/docs/9.4/static/datatype-json.html), are for storing JSON (JavaScript Object Notation) data, this field is ideal to store non-structured data. CDR-Stats aggregate data from several types of telco switches where the type of data received can vary.

7.1 Materialized views

We created 2 Materialized views to help on our reporting, here is the schema structure of those 2 views:

```
-- Materialized View
CREATE MATERIALIZED VIEW matv_voip_cdr_aggr_hour AS
    SELECT
        date_trunc('hour', starting_date) as starting_date,
        country_id,
        switch_id,
        cdr_source_type,
        hangup_cause_id,
        user_id,
        count(*) AS nbcalls,
        sum(duration) AS duration,
        sum(billsec) AS billsec,
        sum(buy_cost) AS buy_cost,
        sum(sell_cost) AS sell_cost
    FROM
        voip_cdr
    GROUP BY
```

```
        date_trunc('hour', starting_date), country_id, switch_id, cdr_source_type, hangup_cause_id, u

-- Create index on Materialized view
CREATE UNIQUE INDEX matv_voip_cdr_aggr_hour_date
  ON matv_voip_cdr_aggr_hour (starting_date, country_id, switch_id, cdr_source_type, hangup_cause_id)

-- Materialized View
CREATE MATERIALIZED VIEW matv_voip_cdr_aggr_min AS
    SELECT
        date_trunc('minute', starting_date) as starting_date,
        country_id,
        switch_id,
        cdr_source_type,
        hangup_cause_id,
        user_id,
        count(*) AS nbcalls,
        sum(duration) AS duration,
        sum(billsec) AS billsec,
        sum(buy_cost) AS buy_cost,
        sum(sell_cost) AS sell_cost
    FROM
        voip_cdr
    GROUP BY
        date_trunc('minute', starting_date), country_id, switch_id, cdr_source_type, hangup_cause_id,

-- Create index on Materialized view
CREATE UNIQUE INDEX matv_voip_cdr_aggr_min_date
  ON matv_voip_cdr_aggr_min (starting_date, country_id, switch_id, cdr_source_type, hangup_cause_id);
```

You can drop those views with:

```
-- Drop Materialized View
DROP MATERIALIZED VIEW matv_voip_cdr_aggr_hour;

-- Drop Materialized View
DROP MATERIALIZED VIEW matv_voip_cdr_aggr_min;
```

You can refresh the view as follows, using "CONCURRENTLY" to ensure we do not lock the view:

```
# Refresh without lock
REFRESH MATERIALIZED VIEW CONCURRENTLY matv_voip_cdr_aggr_hour;

# Refresh without lock
REFRESH MATERIALIZED VIEW CONCURRENTLY matv_voip_cdr_aggr_min;
```

The update of the Materialized view is done periodically by a celery task using the above commands "REFRESH MATERIALIZED VIEW".

CHAPTER 8

Developer doc

Contents:

8.1 Prerequisites

To fully understand this project, developers will need to have a advanced knowledge of:

- Django : http://www.djangoproject.com/
- Celery : http://www.celeryproject.org/
- Python : http://www.python.org/
- Freeswitch : http://www.freeswitch.org/
- Asterisk : http://www.asterisk.org/

8.2 Coding Style & Structure

8.2.1 Style

Coding follows the PEP 8 Style Guide for Python Code.

8.2.2 Structure

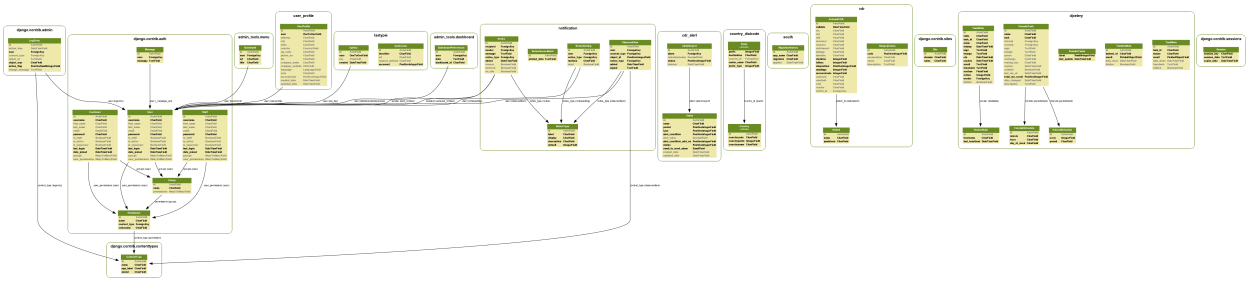
The CDR-Stats directory:

```
|-- api                - The code for APIs
|   `-- api_playground
|-- cdr                - The code for CDR
|   |-- management
|   |-- templatetags
|   `-- fixtures
|-- cdr_alert          - The code for alarm, blacklist, whitelist
|   |-- management
|   `-- fixtures
|-- frontend           - The code for login, logout user
|-- user_profile       - The code for user detail of system
|-- static
```

```
|    |-- cdr
|    |     |-- css
|    |     |-- js
|    |     |-- icons
|    |     `-- images
|-- resources           - This area is used to hold media files
`-- templates           - This area is used to override templates
    |-- admin
    |-- admin_tools
    |-- api_browser
    `-- frontend
```

8.3 Database Design

The current database schema is shown below:



Follow this link for more details : https://github.com/cdr-stats/cdr-stats/raw/master/docs/source/_static/images/model_cdr-stats.png

8.4 Objects Description

8.4.1 Switch

class cdr.models.**Switch**(**args*, ***kwargs*)

This defines the Switch

Attributes:

- name - Name of switch.
- ipaddress - ipaddress

Name of DB table: voip_switch

8.4.2 HangupCause

class cdr.models.**HangupCause**(**args*, ***kwargs*)

This defines the HangupCause

Attributes:

- code - ITU-T Q.850 Code.
- enumeration - Enumeration

- cause - cause
- description - cause description

Name of DB table: hangup_cause

8.4.3 UserProfile

class user_profile.models.**UserProfile**(**args*, ***kwargs*)

This defines extra features for the user

Attributes:

- accountcode - Account name.
- address -
- city -
- state -
- address -
- country -
- zip_code -
- phone_no -
- fax -
- company_name -
- company_website -
- language -
- note -

Relationships:

- user - Foreign key relationship to the User model.
- userprofile_gateway - ManyToMany
- userprofile_voipservergroup - ManyToMany
- dialersetting - Foreign key relationship to the DialerSetting model.

Name of DB table: user_profile

8.4.4 Alarm

8.4.5 AlertRemovePrefix

8.4.6 AlarmReport

8.4.7 Blacklist

8.4.8 Whitelist

8.4.9 VoIPPlan

class voip_billing.models.**VoIPPlan**(**args*, ***kwargs*)
VoIPPlans are associated to your clients, this defines the rate at which the VoIP calls are sold to your clients. A VoIPPlan is a collection of VoIPRetailPlans, you can have 1 or more VoIPRetailPlans associated to the VoIPPlan

A client has a single VoIPPlan, VoIPPlan has many VoIPRetailPlans. VoIPRetailPlan has VoIPRetailRates

The LCR system will route the VoIP via the lowest cost carrier.

8.4.10 BanPlan

class voip_billing.models.**BanPlan**(**args*, ***kwargs*)
List of Ban Plan which are linked to VoIP Plan

8.4.11 VoIPPlan_BanPlan

class voip_billing.models.**VoIPPlan_BanPlan**(**args*, ***kwargs*)
OnetoMany relationship between VoIPPlan & BanPlan

8.4.12 BanPrefix

class voip_billing.models.**BanPrefix**(**args*, ***kwargs*)
Ban prefixes are linked to Ban plan & VoIP with these prefix will not be authorized to send.

prefix_with_name()
Return prefix with name on Ban Prefix Listing (changelist_view)

8.4.13 VoIPRetailPlan

class voip_billing.models.**VoIPRetailPlan**(**args*, ***kwargs*)
This contains the VoIPRetailRates to retail to the customer. these plans are associated to the VoIPPlan with a ManyToMany relation. It defines the costs at which we sell the VoIP calls to clients.

VoIPRetailPlan will then contain a set of VoIPRetailRates which will define the cost of sending a VoIP call to each destination. The system can have several VoIPRetailPlans, but only the ones associated to the VoIPplan will be used by the client.

8.4.14 VoIPPlan_VoIPRetailPlan

class voip_billing.models.**VoIPPlan_VoIPRetailPlan**(**args*, ***kwargs*)
ManytoMany relationship between VoIPPlan & VoIPRetailPlan

8.4.15 VoIPRetailRate

class voip_billing.models.**VoIPRetailRate**(**args*, ***kwargs*)
A single VoIPRetailRate consists of a retail rate and prefix at which you want to use to sell a VoIP Call to a particular destination. VoIPRetailRates are grouped by VoIPRetailPlan, which will be then in turn be associated to a VoIPPlan

prefix_with_name()
Return prefix with name on Retail Rate listing (changelist_view)

voip_retail_plan_name()
Return Retail Plan name on Retail Rate listing (changelist_view)

8.4.16 VoIPCarrierPlan

class voip_billing.models.**VoIPCarrierPlan**(**args*, ***kwargs*)
Once the retail price is defined by the VoIPPlan, VoIPRetailPlans and VoIPRetailRates, we also need to know which is the best route to send the VoIP how much it will cost, and which VoIP Gateway to use.

VoIPCarrierPlan is linked to the VoIP Plan, so once we found how to sell the service to the client, we need to look at which carrier (Provider) we want to use, The VoIPCarrierPlan defines this.

The system can have several VoIPCarrierPlans, but only the one associated to the VoIPRetailPlan-VoIPPlan will be used to connect the VoIP of the client.

8.4.17 VoIPCarrierRate

class voip_billing.models.**VoIPCarrierRate**(**args*, ***kwargs*)
The VoIPCarrierRates are a set of all the carrier rate and prefix that will be used to purchase the VoIP from your carrier, VoIPCarrierRates are grouped by VoIPCarrierPlan, which will be then associated to a VoIPRetailPlan

prefix_with_name()
Return prefix with name on Carrier Rate listing (changelist_view)

voip_carrier_plan_name()
Return Carrier Plan name on Carrier Rate listing (changelist_view)

8.4.18 VoIPPlan_VoIPCarrierPlan

class voip_billing.models.**VoIPPlan_VoIPCarrierPlan**(**args*, ***kwargs*)
ManytoMany relationship between VoIPPlan & VoIPCarrierPlan

8.5 Objects used by the VoIP Billing module

8.5.1 Prefix

These are the prefixes and destinations. For instance, 44 ; United Kingdom

8.5.2 Provider

This defines the VoIP Provider you want to use to deliver your VoIP calls. Each provider will be associated to a Gateway which will link to the Service Provider.

8.5.3 VoIPPlan

VoIPPlans are associated to your clients, this defines the rate at which the VoIP calls are sold to your clients. A VoIPPlan is a collection of VoIPRetailPlans, you can have 1 or more VoIPRetailPlans associated to the VoIPPlan.

- A client has a single VoIPPlan
- A VoIPPlan has many VoIPRetailPlans
- A VoIPRetailPlan has VoIPRetailRates

LCR rules will bill the call based on the lowest cost carrier.

8.5.4 VoIPRetailPlan

This contains the *VoIPRetailRates*, the list of rates to retail to the customer. These plans are associated to the VoIPPlan with a ManyToMany relation.

It defines the costs at which we sell the VoIP calls to the clients. VoIPRetailPlan will then contain a set of VoIPRetail-Rates which will define the cost of sending a VoIP to each destination.

The system can have several VoIPRetailPlans, but only the ones associated to the VoIPplan will be used by the client.

8.5.5 VoIPPlan_VoIPRetailPlan

Help to setup the *ManytoMany* relationship between VoIPPlan & VoIPRetailPlan.

8.5.6 VoIPRetailRate

A single VoIPRetailRate consists of a retail rate and prefix at which you want to use to sell a VoIP to a particular destination. VoIPRetailRates are grouped by VoIPRetailPlan, which will be then in turn be associated to a VoIPPlan.

8.5.7 VoIPCarrierPlan

Once the retail price is defined by the VoIPRetailPlan, we also need to know which is the best route to send the call, what will be our cost, and which Gateway/Provider will be used.

VoIPCarrierPlan is linked to the VoIPRetailPlan, so once we have determined the destination, we need to look at which carrier (Provider) we want to use. The VoIPCarrierPlan defines exactly this.

The system can have several VoIPCarrierPlans, but only the one associated to the VoIPRetailPlan-VoIPPlan will be used to connect the VoIP of the client.

8.5.8 VoIPCarrierRate

The VoIPCarrierRates are a set of all the carrier rate and prefix that will be used to purchase the VoIP from your carrier, VoIPCarrierRates are grouped by VoIPCarrierPlan, which will be then associated to a VoIPRetailPlan.

8.5.9 VoIPPlan_VoIPCarrierPlan

Help to setup the *ManytoMany* relationship between VoIPPlan & VoIPCarrierPlan.

8.5.10 VoIP Call Report

This gives information of all the call delivered with the carrier charges and revenue of each message.

8.6 CDR-Stats Views

8.6.1 cdr_view

8.6.2 cdr_detail

8.6.3 cdr_dashboard

8.6.4 cdr_overview

8.6.5 cdr_realtime

8.6.6 cdr_daily_comparison

8.6.7 cdr_concurrent_calls

8.6.8 world_map_view

8.6.9 mail_report

8.6.10 customer_detail_change

8.6.11 alarm_list

8.6.12 alarm_add

8.6.13 alarm_del

8.6.14 alarm_change

8.6.15 alarm_test

8.6.16 alert_report

8.6.17 trust_control

8.6.18 index

8.6.19 diagnostic

8.6.20 login_view

8.6.21 logout_view

8.6.22 pleaselog

8.6.23 voip_rates

8.6.24 export_rate

8.6.25 simulator

8.6.26 billing_report

8.6.27 cust_password_reset

mod_registration.views.**cust_password_reset**(*request*)
Use django.contrib.auth.views.password_reset view method for forgotten password on the Customer UI

This method sends an e-mail to the user's email-id which is entered in password_reset_form

8.6.28 cust_password_reset_done

mod_registration.views.**cust_password_reset_done**(*request*)
Use django.contrib.auth.views.password_reset_done view method for forgotten password on the Customer UI

This will show a message to the user who is seeking to reset their password.

8.6.29 cust_password_reset_confirm

mod_registration.views.**cust_password_reset_confirm**(*request*, *uidb64=None*, *token=None*)

Use `django.contrib.auth.views.password_reset_confirm` view method for forgotten password on the Customer UI

This will allow a user to reset their password.

8.6.30 cust_password_reset_complete

mod_registration.views.**cust_password_reset_complete**(*request*)

Use `django.contrib.auth.views.password_reset_complete` view method for forgotten password on theCustomer UI

This shows an acknowledgement to the user after successfully resetting their password for the system.

8.7 CDR-Stats Tasks

8.7.1 sync_cdr_pending

8.7.2 chk_alarm

8.7.3 blacklist_whitelist_notification

8.7.4 send_cdr_report

8.7.5 RebillingTask

class voip_billing.tasks.**RebillingTask**

Re-billing for VoIPCall

Usage:

RebillingTask.delay(calls_kwargs, voipplan_id)

8.7.6 ReaggregateTask

class voip_billing.tasks.**RebillingTask**

Re-billing for VoIPCall

Usage:

RebillingTask.delay(calls_kwargs, voipplan_id)

8.8 Test Case Descriptions

8.8.1 Requirement

Run/Start Celery:

```
$ /etc/init.d/celery start
```

or:

```
$ python manage.py celeryd -l info
```

Run/Start Redis:

```
$ /etc/init.d/redis-server start
```

8.8.2 How to Run Tests

1. Run Full Test Suit:

```
$ python manage.py test --verbosity=2
```

3. Run CDRStatsAdminInterfaceTestCase:

```
$ python manage.py test cdr.CDRStatsAdminInterfaceTestCase --verbosity=2
```

4. Run CDRStatsCustomerInterfaceTestCase:

```
$ python manage.py test cdr.CDRStatsCustomerInterfaceTestCase --verbosity=2
```

8.9 Javascript Files

- jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. jQuery is designed to change the way that you write JavaScript.
- NVD3 is an attempt to build re-usable charts and chart components for d3.js without taking away the power that d3.js gives you. This is a very young collection of components, with the goal of keeping these components very customizeable, staying away from your standard cookie cutter solutions.
- Bootstrap is sleek, intuitive, and powerful front-end framework for faster and easier web development.
- Bootbox is a small JavaScript library which allows you to create programmatic dialog boxes using Twitter's Bootstrap modals, without having to worry about creating, managing or removing any of the required DOM elements or JS event handlers.
- Bootstrap-datepicker Datepicker for Bootstrap

CHAPTER 9

API Reference

Contents:

9.1 SwitchSerializer

class apirest.switch_serializers.**SwitchSerializer**(*instance=None*, *data=None*, *files=None*, *context=None*, *partial=False*, *many=False*, *allow_add_remove=False*, ***kwargs*)

Read:

CURL Usage:

```
curl -u username:password -H 'Accept: application/json' http://localhost:8000/rest-api/switc

curl -u username:password -H 'Accept: application/json' http://localhost:8000/rest-api/switc
```

Response:

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "url": "http://127.0.0.1:8000/rest-api/switch/1/",
            "name": "localhost",
            "ipaddress": "127.0.0.1",
            "key_uuid": "25116b72-b477-11e1-964f-000c296bd875"
        }
    ]
}
```

9.2 VoIPRateList

class apirest.view_voip_rate.**VoIPRateList**(***kwargs*)

List all voip rate

Read:

CURL Usage:

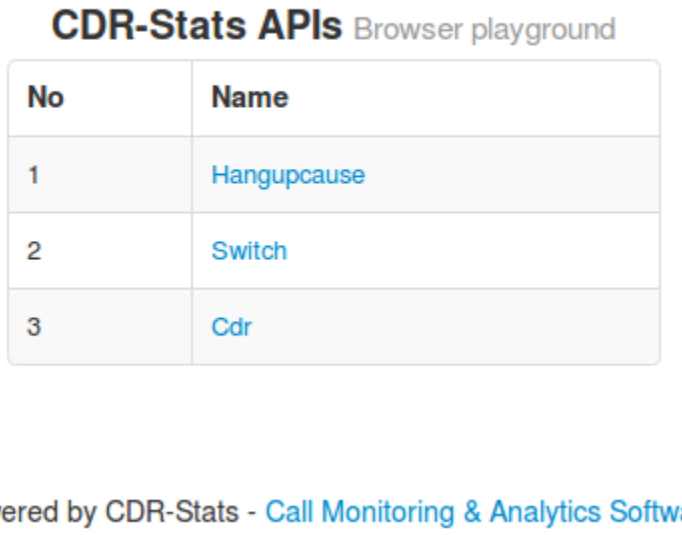
```
curl -u username:password -H 'Accept: application/json'
http://localhost:8000/rest-api/voip-rate/?recipient_phone_no=4323432&sort_field=prefix&order

curl -u username:password -H 'Accept: application/json'
http://localhost:8000/rest-api/voip-rate/?dialcode=4323432&sort_field=prefix&order=desc
```

9.3 VoipCallResource

Testing console of APIs:

URL : http://127.0.0.1:8000/api-explorer/



To test individual api, click on one api from the api list and you will get a similar screen as follows:

URL : http://127.0.0.1:8000/api-explorer/switch/


Switch API Playground
/switch/
This resource allows you to manage switchs.
GET /api/v1/switch/ Returns all switchs
GET
Request
GET /api/v1/switch/
Content-Type: application/json; charset=utf-8
Response Status
OK (200)
Response Headers
Date: Fri, 19 Oct 2012 10:23:46 GMT
Server: WSGIServer/0.1 Python/2.7.3
Vary: Accept-Language, Cookie
Content-Type: application/json; charset=utf-8
Content-Language: en
Cache-Control: no-cache
Response Body
{"meta": {"limit": 20, "next": null, "offset": 0, "previous": null, "total_count": 4},
"objects": [{"id": "1", "ipaddress": "127.0.0.1", "key_uuid": "c80445f8-183f-11e2-964f-
000c2925d15f", "name": "127.0.0.1", "resource_uri": "/api/v1/switch/1/"}]}
Give feedback about this response
GET /api/v1/switch/{switch-id}/ Returns a specific switch
URL Parameters
switch-id:
GET
POST /api/v1/switch/ Creates new switch
Data Parameters
name:
localhost
ipaddress:
192.168.1.4
POST
PUT /api/v1/switch/{switch-id}/ Update switch
URL Parameters
switch-id:
Data Parameters
name:
localhost
ipaddress:
192.168.1.4
DELETE /api/v1/switch/{switch-id}/ Delete switch

CHAPTER 10

Contributing

This document is highly inspired from the Celery documentation.

Welcome to CDR-Stats!

This document is fairly extensive and you are not really expected to study this in detail for small contributions;

> The most important rule is that contributing must be easy and that the community is friendly and not nitpicking on details such as coding style.

If you're reporting a bug you should read the Reporting bugs section below to ensure that your bug report contains enough information to successfully diagnose the issue, and if you're contributing code you should try to mimic the conventions you see surrounding the code you are working on, but in the end all patches will be cleaned up by the person merging the changes so don't worry too much.

10.1 Community Code of Conduct

The goal is to maintain a diverse community that is pleasant for everyone. That is why we would greatly appreciate it if everyone contributing to and interacting with the community also followed this Code of Conduct.

The Code of Conduct covers our behavior as members of the community, in any forum, mailing list, wiki, website, Internet relay chat (IRC), public meeting or private correspondence.

The Code of Conduct is heavily based on the Ubuntu Code of Conduct, Celery Code of Conduct, and the Pylons Code of Conduct.

10.1.1 Be considerate.

Your work will be used by other people, and you in turn will depend on the work of others. Any decision you take will affect users and colleagues, and we expect you to take those consequences into account when making decisions. Even if it's not obvious at the time, our contributions to CDR-Stats will impact the work of others. For example, changes to code, infrastructure, policy, documentation and translations during a release may negatively impact others work.

10.1.2 Be respectful.

The CDR-Stats community and its members treat one another with respect. Everyone can make a valuable contribution to CDR-Stats. We may not always agree, but disagreement is no excuse for poor behavior and poor manners. We might all experience some frustration now and then, but we cannot allow that frustration to turn into a personal attack. It's important to remember that a community where people feel uncomfortable or threatened is not a productive one. We expect members of the CDR-Stats community to be respectful when dealing with other contributors as well as with people outside the CDR-Stats project and with users of CDR-Stats.

10.1.3 Be collaborative.

Collaboration is central to CDR-Stats and to the larger free software community. We should always be open to collaboration. Your work should be done transparently and patches from CDR-Stats should be given back to the community when they are made, not just when the distribution releases. If you wish to work on new code for existing upstream projects, at least keep those projects informed of your ideas and progress. It many not be possible to get consensus from upstream, or even from your colleagues about the correct implementation for an idea, so don't feel obliged to have that agreement before you begin, but at least keep the outside world informed of your work, and publish your work in a way that allows outsiders to test, discuss and contribute to your efforts.

10.1.4 When you disagree, consult others.

Disagreements, both political and technical, happen all the time and the CDR-Stats community is no exception. It is important that we resolve disagreements and differing views constructively and with the help of the community and community process. If you really want to go a different way, then we encourage you to make a derivative distribution or alternate set of packages that still build on the work we've done to utilize as common of a core as possible.

10.1.5 When you are unsure, ask for help.

Nobody knows everything, and nobody is expected to be perfect. Asking questions avoids many problems down the road, and so questions are encouraged. Those who are asked questions should be responsive and helpful. However, when asking a question, care must be taken to do so in an appropriate forum.

10.1.6 Step down considerately.

Developers on every project come and go and CDR-Stats is no different. When you leave or disengage from the project, in whole or in part, we ask that you do so in a way that minimizes disruption to the project. This means you should tell people you are leaving and take the proper steps to ensure that others can pick up where you leave off.

10.2 Reporting Bugs

10.2.1 Bugs

Bugs can always be described to the *Mailing list*, but the best way to report an issue and to ensure a timely response is to use the issue tracker.

1. **Create a GitHub account.**

You need to create a GitHub account to be able to create new issues and participate in the discussion.

2. **Determine if your bug is really a bug.**

You should not file a bug if you are requesting support. For that you can use the *Mailing list*, or *IRC*.

3. **Make sure your bug hasn't already been reported.**

Search through the appropriate Issue tracker. If a bug like yours was found, check if you have new information that could be reported to help the developers fix the bug.

4. **Check if you're using the latest version.**

A bug could be fixed by some other improvements and fixes - it might not have an existing report in the bug tracker. Make sure you're using the latest version.

5. **Collect information about the bug.**

To have the best chance of having a bug fixed, we need to be able to easily reproduce the conditions that caused it. Most of the time this information will be from a Python traceback message, though some bugs might be in design, spelling or other errors on the website/docs/code.

1. If the error is from a Python traceback, include it in the bug report.
2. We also need to know what platform you're running (Windows, OS X, Linux, etc.), the version of your Python interpreter, and the version of related packages that you were running when the bug occurred.

6. **Submit the bug.**

By default GitHub will email you to let you know when new comments have been made on your bug. In the event you've turned this feature off, you should check back on occasion to ensure you don't miss any questions a developer trying to fix the bug might ask.

10.2.2 Issue Trackers

Bugs for a package in the CDR-Stats ecosystem should be reported to the relevant issue tracker.

- CDR-Stats Core: https://github.com/cdr-stats/cdr-stats/issues/
- Python-Acapela: https://github.com/cdr-stats/python-acapela/issues
- Lua-Acapela: https://github.com/cdr-stats/lua-acapela/issues
- Python-NVD3: https://github.com/areski/python-nvd3/issues

If you are unsure of the origin of the bug you can ask the *Mailing list*, or just use the CDR-Stats issue tracker.

10.3 Versions

Version numbers consists of a major version, minor version and a release number. We use the versioning semantics described by semver: http://semver.org.

Stable releases are published at PyPI while development releases are only available in the GitHub git repository as tags. All version tags starts with "v", so version 0.8.0 is the tag v0.8.0.

10.4 Branches

Current active version branches:

- master (http://github.com/cdr-stats/cdr-stats/tree/master)
- 2.19.10 (http://github.com/cdr-stats/cdr-stats/tree/v2.19.10)

You can see the state of any branch by looking at the Changelog:

https://github.com/cdr-stats/cdr-stats/blob/master/Changelog

10.4.1 Feature branches

Major new features are worked on in dedicated branches. There is no strict naming requirement for these branches.

Feature branches are removed once they have been merged into a release branch.

10.5 Tags

Tags are used exclusively for tagging releases. A release tag is named with the format `vX.Y.Z`, e.g. `v2.3.1`. Experimental releases contain an additional identifier `vX.Y.Z-id`, e.g. `v3.0.0-rc1`. Experimental tags may be removed after the official release.

10.6 Working on Features & Patches

Note: Contributing to CDR-Stats should be as simple as possible, so none of these steps should be considered mandatory.

You can even send in patches by email if that is your preferred work method. We won't like you any less, any contribution you make is always appreciated!

However following these steps may make maintainers life easier, and may mean that your changes will be accepted sooner.

10.6.1 Forking and setting up the repository

First you need to fork the repository, a good introduction to this is in the Github Guide: Fork a Repo.

After you have cloned the repository you should checkout your copy to a directory on your machine:

```
$ git clone git@github.com:username/cdr-stats.git
```

When the repository is cloned enter the directory to set up easy access to upstream changes:

```
$ cd cdr-stats
$ git remote add upstream git://github.com/cdr-stats/cdr-stats.git
$ git fetch upstream
```

If you need to pull in new changes from upstream you should always use the `--rebase` option to `git pull`:

```
$ git pull --rebase upstream master
```

With this option you don't clutter the history with merging commit notes. See Rebasing merge commits in git. If you want to learn more about rebasing see the Rebase section in the Github guides.

If you need to work on a different branch than `master` you can fetch and checkout a remote branch like this:

```
$ git checkout --track -b 3.0-devel origin/3.0-devel
```

10.6.2 Running the unit test suite

To run the CDR-Stats test suite you need to install a few dependencies. A complete list of the dependencies needed are located in `requirements/test.txt`.

Installing the test requirements:

```
$ pip install -U -r requirements/test.txt
```

When installation of dependencies is complete you can execute the test suite by calling `py.test`:

```
$ py.test
```

Some useful options to **`py.test`** are:

- `-x`

  Exit instantly on first error or failed test.

- `--ipdb`

  Starts the interactive IPython debugger on errors.

- `-k EXPRESSION`

  Only run tests which match the given substring expression.

- `-v`

  Increase verbose.

If you want to run the tests for a single test file only you can do so like this:

```
$ py.test appointment./tests.py
```

10.6.3 Creating pull requests

When your feature/bugfix is complete you may want to submit a pull requests so that it can be reviewed by the maintainers.

Creating pull requests is easy, and also let you track the progress of your contribution. Read the Pull Requests section in the Github Guide to learn how this is done.

You can also attach pull requests to existing issues by following the steps outlined here: http://bit.ly/koJoso

Calculating test coverage

To calculate test coverage you must first install the `coverage` module.

Installing the `coverage` module:

```
$ pip install -U coverage
```

Code coverage in HTML:

```
$ nosetests --with-coverage --cover-html
```

The coverage output will then be located at `cdr-stats/tests/cover/index.html`.

Code coverage in XML (Cobertura-style):

```
$ nosetests --with-coverage --cover-xml --cover-xml-file=coverage.xml
```

The coverage XML output will then be located at `coverage.xml`

Running the tests on all supported Python versions

There is a `tox` configuration file in the top directory of the distribution.

To run the tests for all supported Python versions simply execute:

```
$ tox
```

If you only want to test specific Python versions use the `-e` option:

```
$ tox -e py27
```

10.6.4 Building the documentation

To build the documentation you need to install the dependencies listed in `requirements/docs.txt`:

```
$ pip install -U -r requirements/docs.txt
```

After these dependencies are installed you should be able to build the docs by running:

```
$ cd docs
$ rm -rf .build
$ make html
```

Make sure there are no errors or warnings in the build output. After building succeeds the documentation is available at `.build/html`.

10.6.5 Verifying your contribution

To use these tools you need to install a few dependencies. These dependencies can be found in `requirements/pkgutils.txt`.

Installing the dependencies:

```
$ pip install -U -r requirements/pkgutils.txt
```

pyflakes & PEP8

To ensure that your changes conform to PEP8 and to run pyflakes execute:

```
$ flake8 cdr_stats
```

10.7 Coding Style

You should probably be able to pick up the coding style from surrounding code, but it is a good idea to be aware of the following conventions.

- All Python code must follow the PEP-8 guidelines.

pep8.py is an utility you can use to verify that your code is following the conventions.

- Docstrings must follow the PEP-257 conventions, and use the following style.

  Do this:

```
def method(self, arg):
    """Short description.

    More details.

    """
```

  or:

```
def method(self, arg):
    """Short description."""
```

  but not this:

```
def method(self, arg):
    """
    Short description.
    """
```

- Lines should not exceed 78 columns.

  You can enforce this in **vim** by setting the `textwidth` option:

```
set textwidth=78
```

  If adhering to this limit makes the code less readable, you have one more character to go on, which means 78 is a soft limit, and 79 is the hard limit :)

- Import order
  - Python standard library (*import xxx*)
  - Python standard library ('from xxx import')
  - Third party packages.
  - Other modules from the current package.

  or in case of code using Django:

  - Python standard library (*import xxx*)
  - Python standard library ('from xxx import')
  - Third party packages.
  - Django packages.
  - Other modules from the current package.

  Within these sections the imports should be sorted by module name.

  Example:

```
import threading
import time

from collections import deque
from Queue import Queue, Empty
```

```
        from .datastructures import TokenBucket
        from .five import zip_longest, items, range
        from .utils import timeutils
```

- Wildcard imports must not be used (*from xxx import* *).
- For distributions where Python 2.5 is the oldest support version additional rules apply:
  - Absolute imports must be enabled at the top of every module:

```
        from __future__ import absolute_import
```

  - If the module uses the with statement and must be compatible with Python 2.5 then it must also enable that:

```
        from __future__ import with_statement
```

  - Every future import must be on its own line, as older Python 2.5 releases did not support importing multiple features on the same future import line:

```
        # Good
        from __future__ import absolute_import
        from __future__ import with_statement

        # Bad
        from __future__ import absolute_import, with_statement
```

    (Note that this rule does not apply if the package does not include support for Python 2.5)

- Note that we use "new-style' relative imports when the distribution does not support Python versions below 2.5

  This requires Python 2.5 or later:

```
        from . import submodule
```

10.8 Contacts

This is a list of people that can be contacted for questions regarding the official git repositories, PyPI packages Read the Docs pages.

If the issue is not an emergency then it is better to *report an issue*.

10.8.1 Committers

Areski Belaid

github https://github.com/areski

twitter http://twitter.com/#!/areskib

10.8.2 Website

The CDR-Stats Project is run and maintained by

Star2Billing

website http://star2billing.com/

twitter https://twitter.com/#!/star2billing

10.9 Release Procedure

10.9.1 Updating the version number

The version number must be updated one place:

- `cdr_stats/cdr_stats/__init__.py`

After you have changed these files you must render the `README` files. There is a script to convert sphinx syntax to generic reStructured Text syntax, and the make target *readme* does this for you:

```
$ make readme
```

Now commit the changes:

```
$ git commit -a -m "Bumps version to X.Y.Z"
```

and make a new version tag:

```
$ git tag vX.Y.Z
$ git push --tags
```

10.9.2 Releasing

Commands to make a new public stable release:

```
$ make distcheck  # checks pep8, autodoc index, runs tests and more
$ make dist  # NOTE: Runs git clean -xdf and removes files not in the repo.
$ python setup.py sdist bdist_wheel upload  # Upload package to PyPI
```

If this is a new release series then you also need to do the following:

- **Go to the Read The Docs management interface at:** http://readthedocs.org/projects/cdr-stats/?fromdocs=cdr-stats
- Enter "Edit project"

  Change default branch to the branch of this series, e.g. `2.4` for series 2.4.
- Also add the previous version under the "versions" tab.

CHAPTER 11

Resources

11.1 Getting Help

11.1.1 Mailing list

For discussions about the usage, development, and future of CDR-Stats, please join the CDR-Stats mailing list.

11.1.2 IRC

Come chat with us on IRC. The **#cdr-stats** channel is located at the Freenode network.

11.2 Bug tracker

If you have any suggestions, bug reports or annoyances please report them to our issue tracker at https://github.com/cdr-stats/cdr-stats/issues/

11.3 Documentation

The latest documentation with user guides, tutorials and API references is hosted on CDR-Stats website : http://www.cdr-stats.org/documentation/

Beginner's Guide : http://www.cdr-stats.org/documentation/beginners-guide/

11.4 Support

Star2Billing S.L. offers consultancy including installation, training and customisation

Website : http://www.star2billing.com

Email : cdr-stats@star2billing.com

11.5 License

CHAPTER 12

Frequently Asked Questions

12.1 General

12.1.1 What is CDR-Stats?

Answer: CDR-Stats is a free and open source web based Call Detail Record analysis application with the ability to display reports and graphs.

12.1.2 Why should I use CDR-Stats?

Answer: We foresee two main areas where CDR-Stats would be useful. For telecoms companies who wish to mediate and rate call data records, ultimately to create invoices for their customers, as well as do carrier reconciliation, and for organisations that wish to analyse call patterns. For instance: if you have call detail records from an office PBX, telecoms switch(s), or carrier CDR to analyse then CDR-Stats is a useful tool to analyse the data and look for patterns in the traffic that may indicate problems or potential fraud. Furthermore, CDR-Stats can be configured to send email alerts on detection of unusual activity, as well as send daily reports on traffic.

12.2 CDR Import

12.2.1 How to start over and relaunch the import?

Answer: First stop celery by stopping supervisor:

```
$ /etc/init.d/supervisor stop
```

Then remove the aggregate data, connect on postgresql and enter the following:

```
DROP MATERIALIZED VIEW matv_voip_cdr_aggr_hour;
DROP MATERIALIZED VIEW matv_voip_cdr_aggr_min;
```

Recreate the Materialized View as follow:

```
CREATE MATERIALIZED VIEW matv_voip_cdr_aggr_hour AS
    SELECT
        date_trunc('hour', starting_date) as starting_date,
        country_id,
        switch_id,
        cdr_source_type,
        hangup_cause_id,
        user_id,
        count(*) AS nbcalls,
        sum(duration) AS duration,
        sum(billsec) AS billsec,
        sum(buy_cost) AS buy_cost,
        sum(sell_cost) AS sell_cost
    FROM
        voip_cdr
    GROUP BY
        date_trunc('hour', starting_date), country_id, switch_id, cdr_source_type, hangup_cause_id, u

CREATE MATERIALIZED VIEW matv_voip_cdr_aggr_min AS
    SELECT
        date_trunc('minute', starting_date) as starting_date,
        country_id,
        switch_id,
        cdr_source_type,
        hangup_cause_id,
        user_id,
        count(*) AS nbcalls,
        sum(duration) AS duration,
        sum(billsec) AS billsec,
        sum(buy_cost) AS buy_cost,
        sum(sell_cost) AS sell_cost
    FROM
        voip_cdr
    GROUP BY
        date_trunc('minute', starting_date), country_id, switch_id, cdr_source_type, hangup_cause_id,
```

Then, update all your CDRs from 'import_cdr' PostgreSQL database to be reimported as we flag them after import:

```
UPDATE cdr_import SET imported=FALSE;
```

Restart Celery:

```
$ /etc/init.d/supervisor stop
```

Finally check in the logs file that the CDRs are being imported:

```
tail -f /var/log/cdr-stats/djcelery_error.log
```

12.3 Debugging

12.3.1 How to debug mail connectivity?

Answer: Use mail_debug to test the mail connectivity:

```
$ workon cdr-stats
$ cd /usr/share/cdrstats
$ python manage.py mail_debug
```

12.3.2 What should I do if I have problems?

Answer:

- Review the installation script, and check that services are running.
- Read the documentation contained in the CDR-Stats website: http://docs.cdr-stats.org/en/latest/
- Ask a question on the mailing list: http://www.cdr-stats.org/community/
- Get professional support from the CDR-Stats team (Star2Billing): http://www.cdr-stats.org/support/

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

a

c

m

u

v