# CartoDB Platform Documentation Documentation

Release 1.0.0

CartoDB, Inc.

Nov 20, 2020

# Contents

1	CAR	TO introduction 3									
	1.1	What can I do with CARTO?    4									
2	ponents 7										
	2.1	PostgreSQL									
	2.2	PostGIS									
	2.3	Redis									
	2.4	CARTO PostgreSOL extension									
	2.5	CARTO Builder									
	2.6	Maps API									
	2.7	SOL API									
3	Insta	llation 11									
	3.1	System requirements									
	3.2	PostgreSQL									
	3.3	GIS dependencies									
	3.4	PostGIS									
	3.5	Redis									
	3.6	Node.js									
	3.7	SQL API									
	3.8	MAPS API									
	3.9	Ruby 15									
	3.10	Builder									
	-										
4	Runr	ling CARTO									
	4.1	First run, setting up an user									
	4.2	Running all the processes									
5	Conf	Configuration 21									
•	5.1	Basemaps									
	5.2	Basemaps with a layer of labels									
	53	Domainless URLs									
	5.4	Configuration changes for Domainless URLs 23									
	5.5	Limitations									
	5.6	Common Data 23									
	5.0	Separate folders									
	5.1	Separate folders									

)	Oper	ations
	6.1	Creating users
	6.2	Creating organizations
	6.3	Changing Feature Flags
	6.4	Changing limits
	6.5	Exporting/Importing visualizations
	6.6	Exporting/Importing full visualizations
	6.7	Running Sync Tables
	6.8	HTTP Header Authentication
	6.9	Configuring Dataservices

35

#### 7 Indices and tables

ii

Contents:

# CARTO introduction

CARTO is an open source tool that allows for the storage and visualization of geospatial data on the web.

It was built to make it easier for people to tell their stories by providing them with flexible and intuitive ways to create maps and design geospatial applications. CARTO can be installed on your own server and we also offer a hosted service at carto.com.





# 1.1 What can I do with CARTO?

With CARTO, you can upload your geospatial data (Shapefiles, GeoJSON, etc) using a web form and then make it public or private.

After it is uploaded, you can visualize it in a table or on a map, search it using SQL, and apply map styles using CartoCSS. You can even access it using the CARTO API OVERVIEW and SQL API, or export it to a file.

In other words, with CARTO you can make awesome maps and build powerful geospatial applications! Definitely check out the CARTO Help Center for interactive examples and code.



← <b>6</b> table_50m_urban_area DATA VIEW MAP VIEW Control							options - VISUALIZE		
	cartodb_id - number	the_geom aco geometry	<b>featurecla</b> - string	scalerank - number	<b>created_at -</b> date	<b>updated_at -</b> date		Add	layer
	1	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z		Capit	e_oon_drban_area
	2	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z	SOL		Custom SQL query
	3	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z	alu		Ctrl + SPACE to autocomplete. CMD + S to apply your query.
	4	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z		1	SELECT * FROM table_50m_urban_area
	5	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z			
	6	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z			
	7	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z			
	8	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z			
	9	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z			
	10	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z			
	11	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z			
	12	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z	<b>4</b> <sup>2</sup>		
	13	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z	_		
	14	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z			
	15	Polygon	Urban Area	5	2014-08-28T07:27:33Z	2014-08-28T07:27:33Z	ιţi		Apply query

### Components

A common CARTO stack is powered by those components:

# 2.1 PostgreSQL

PostgreSQL is the open source database powering CARTO.

CARTO uses PostgreSQL for two purposes:

- Metadata storage. This is the metadata used by the CARTO Builder. Builder models like users information, visualizations, or other metadata is stored in this database. The name and connection information of this database is specified on the Rails app\_config.yml configuration file.
- User data storage. Each single user or organization in CARTO has a individual PostgreSQL database. This database is created during the user signup process. Its database name and connection info is generated on the fly by the CARTO application during this process. Both values are stored within the user info in the metadata database. Every user database name contains the user UUID.

Both metadata database and users databases can be hosted either in the same PostgreSQL cluster or different ones. Having both the in the same cluster is the recommended approach for small environments. Builder only knows how to connect to metadata database. However, within every request it checks the connection info of the user database which is stored in metadata database, as described before.

At this moment CARTO requires PostgreSQL 10 version.

## 2.2 PostGIS

PostGIS is the extension which adds spatial capabilities to PostgreSQL. It allows working with geospatial types or running geospatial functions in PostgreSQL. Is is required both on the user data clusters and the metadata cluster.

At this moment CARTO requires PostGIS version 2.4.

# 2.3 Redis

Redis is a key-value store engine used by most components of the CARTO application stack to store configuration and cache.

In contrast to the PostgreSQL metadata (which is used only by the CARTO Builder), the metadata stored in Redis is shared among Builder, the SQL API, and the Maps API.

**Important:** Even though also used as a cache, there is also persistent data stored in Redis. In some environments Redis is configured by default to act without persistency (see http://redis.io/topics/persistence).

You must ensure Redis is configured properly to keep its data between restarts.

#### Data is stored in separate databases inside this Redis:

- Database 0: Table and visualization metadata, including map styles and named maps.
- Database 3: OAuth credentials metadata.
- Database 5: Metadata about the users, including API keys and database\_hosts.
- Database 8: Rate limits.

At this moment CARTO requires Redis version 4.0 or newer with the redis-cell extension.

## 2.4 CARTO PostgreSQL extension

CARTO's PostgreSQL extension can be found at https://github.com/CartoDB/cartodb-postgresql.

This extensions is required by all the components of CARTO and it must be installed in the server where user databases are stored.

It provides functions and other helpers needed by Builder, Maps and SQL APIs like:

- CartoDBfying functions which convert raw PostgreSQL tables in tables recognized by CARTO by adding some additional columns and triggers
- · Multiuser schema handling functions
- · Quota helpers
- · Cache helpers
- etc..

The CARTO extension depends on PostGIS.

## 2.5 CARTO Builder

Builder is the web management component of CARTO. Within Builder you can find all the features available in CARTO. These are some of the most significant tasks you can do with Builder:

- User management. Credentials, authorization, personal info and billing.
- Connect datasets to your CARTO account either by importing your datasets or other ones publicly available.
- Create maps from your datasets

- Publising and permissions management of datasets and maps
- Synchronized tables management

InternallyBuilder is the operations core of CARTO. It manages PostgreSQL metadata database, keep some metadata in sync with Redis, manages new datasets import queues with resque, etc..

It is developed in Ruby on Rails and like the other components of CARTO is Open Source and you can find the source code at CartoDB/cartodb

You can find usage documentation at https://docs.carto.com/cartodb-editor.html

Although you can chechout any branch of the repository most of them are usually work in progress that is not guaranteed to work. In order to run a production ready Editor service you need to use **master** branch.

#### 2.5.1 Service modes

The code of CARTO Builder needs to run in two different modes. HTTP server mode and background jobs mode.

The **HTTP server** processes the http requests sent to the service and returns a response synchronously. Any ruby rack server can be used to start the Builder in this mode. Some examples of rack servers are mongrel, webrick, thin or unicorn.

The **background jobs** mode is started with resque. In this mode the service keep polling some redis keys in order to find pending background jobs. When it finds one, it processes it and change the state of the job in redis. CARTO uses this mode for different type of jobs like datasets imports or synchronized tables.

# 2.6 Maps API

The Maps API provides a node.js based API that allows you to generate maps based on data hosted in your CARTO account by applying custom SQL and CartoCSS to the data

Like the other components of CARTO is Open Source and you can find the source code at CartoDB/Windshaft-cartodb

You can find usage documentation at https://docs.carto.com/cartodb-platform/maps-api.html

Although you can chechout any branch of the repository most of them are usually work in progress that is not guaranteed to work. In order to run a production ready Maps API service you need to use **master** branch.

# 2.7 SQL API

The SQL API provides a node.js based API for running SQL queries against CARTO.

Like the other components of CARTO is Open Source and you can find the source code at CartoDB/CartoDB-SQL-API

You can find usage documentation at https://docs.carto.com/cartodb-platform/sql-api.html.

Although you can chechout any branch of the repository most of them are usually work in progress that is not guaranteed to work. In order to run a production ready SQL API service you need to use **master** branch.

## Installation

**Warning:** CARTO works with Ubuntu 16.04 x64. This documentation describes the process to install CartoDB in this specific OS version.

However this doesn't mean that it won't work with other Operating Systems or other Ubuntu. There are also many successful installations on Amazon EC2, Linode, dedicated instances and development machines running OS X and Ubuntu 12.04+.

You will find notes along this guide explaining some of the Ubuntu 16.04 specifics, and pointing to alternative solutions for other environments.

### 3.1 System requirements

Besides the OS version mentioned in the introduction, there are some system requirements needed before starting with the installation of the stack. Also this process assumes that you have enough permissions in the system to run successfully most part of the commands of this doc.

#### 3.1.1 System locales

Installations assume you use UTF8. You can set the locale by doing this:

```
sudo locale-gen en_US.UTF-8
sudo update-locale LANG=en_US.UTF-8 LC_ALL=en_US.UTF-8
```

#### 3.1.2 Build essentials

Although we try to maintain packaged versions of almost every part of the stack, there are some parts like gems or npm packages that need some development tools in the system in order to compile. You can install all the needed build tools by doing this:

sudo apt-get install make pkg-config

#### 3.1.3 GIT

You will need git commands in order to handle some repositories and install some dependencies:

```
sudo apt-get install git
```

## 3.2 PostgreSQL

**Note:** CARTO requires PostgreSQL 10+. The PPA packages also provide some additional patches, which are not needed but help improve the experience in production environments.

· Add PPA repository

sudo add-apt-repository ppa:cartodb/postgresql-10 && sudo apt-get update

· Install packages

PostgreSQL access authorization is managed through pg\_hba.conf configuration file, which is normally in /etc/ postgresql/10/main/pg\_hba.conf. Here it's defined how the users created in postgresql cluster can access the server. This involves several aspects like type of authentication (md5, no password, etc..) or source IP of the connection. In order to simplify the process of the installation we are going to allow connections with postgres user from localhost without authentication. Of course this can be configured in a different way at any moment but changes here should imply changes in database access configuration of CARTO apps.

Edit /etc/postgresql/10/main/pg\_hba.conf, modifying the existing lines to use trust authentication (no password access from localhost):

local	all	postgres		trust
local	all	all		trust
host	all	all	127.0.0.1/32	trust

For these changes to take effect, you'll need to restart postgres:

sudo systemctl restart postgresql

· Create some users in PostgreSQL. These users are used by some CARTO apps internally

```
sudo createuser publicuser --no-createrole --no-createdb --no-superuser -U_

→postgres

sudo createuser tileuser --no-createrole --no-createdb --no-superuser -U postgres
```

• Install CartoDB postgresql extension. This extension contains functions that are used by different parts of the CartoDB platform, included Builder and the SQL and Maps API.

```
git clone https://github.com/CartoDB/cartodb-postgresql.git
cd cartodb-postgresql
git checkout <LATEST cartodb-postgresql tag>
sudo make all install
```

# 3.3 GIS dependencies

• Add GIS PPA

sudo add-apt-repository ppa:cartodb/gis && sudo apt-get update

• Install GDAL

```
sudo apt-get install gdal-bin libgdal-dev
```

## 3.4 PostGIS

Note: CARTO requires PostGIS 2.4. The PPA just packages this version for Ubuntu 16.04.

• Install PostGIS

```
sudo apt-get install postgis
```

• Initialize template postgis database. We create a template database in postgresql that will contain the postgis extension. This way, every time CartoDB creates a new user database it just clones this template database

• (Optional) Run an installcheck to verify the database has been installed properly

sudo PGUSER=postgres make installcheck # to run tests

Check https://github.com/cartodb/cartodb-postgresql for further reference

### 3.5 Redis

**Note:** CARTO requires Redis 4+. You can also optionally install redis-cell for rate limiting, which is not described by this guide.

• Add redis PPA

sudo add-apt-repository ppa:cartodb/redis-next && sudo apt-get update

• Install redis

sudo apt-get install redis

**Warning:** By default redis server is configured to only have periodic snapshotting to disk. If stopped or restarted some data stored in redis since the last snahpshot can be lost. In CARTO redis is not just a simple cache storage. It stores information that need to be persisted.

For data safety, make sure to have proper values of *save*, *appendonly* and *appendfsync* config attributes. For more information check *http://redis.io/topics/persistence* 

#### 3.6 Node.js

Note: CARTO requires Node.js 10+ and npm 6+.

Node.js is required by different parts of the stack. The more significant are the Maps and SQL APIs. It's also used to install and execute some dependencies of Builder.

· Install Node.js

```
curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Note this should install both Node.js 10.x and npm 6.x. You can verify the installation went as expected with:

node -v npm -v

We will also install some development libraries that will be necessary to build some Node.js modules:

```
sudo apt-get install libpixman-1-0 libpixman-1-dev
sudo apt-get install libcairo2-dev libjpeg-dev libgif-dev libpango1.0-dev
```

## 3.7 SQL API

· Download API

```
git clone git://github.com/CartoDB/CartoDB-SQL-API.git
cd CartoDB-SQL-API
```

• Install npm dependencies

npm install

• Create configuration. The name of the filename of the configuration must be the same than the environment you are going to use to start the service. Let's assume it's development.

cp config/environments/development.js.example config/environments/development.js

• Start the service. The second parameter is always the environment if the service. Remember to use the same you used in the configuration.

```
node app.js development
```

### 3.8 MAPS API

Download API

```
git clone git://github.com/CartoDB/Windshaft-cartodb.git
cd Windshaft-cartodb
```

• Install yarn dependencies

npm install

• Create configuration. The name of the filename of the configuration must be the same than the environment you are going to use to start the service. Let's assume it's development.

```
cp config/environments/development.js.example config/environments/development.js
mkdir logs
```

• Start the service. The second parameter is always the environment of the service. Remember to use the same you used in the configuration.

node app.js development

## 3.9 Ruby

Note: CARTO requires exactly Ruby 2.4.x. Older or newer versions won't work.

· Add brightbox ruby repositories

sudo apt-add-repository ppa:brightbox/ruby-ng && sudo apt-get update

• Install ruby 2.4

```
sudo apt-get install ruby2.4 ruby2.4-dev
```

• Install bundler. Bundler is an app used to manage ruby dependencies. It is needed by CARTO Builder

```
sudo apt-get install ruby-bundler
```

· Install compass. It will be needed later on by CARTO's Builder

sudo gem install compass

### 3.10 Builder

Note: CARTO users Python 2.7+. Python 3 will not work correctly.

· Download Builder's code

```
git clone --recursive https://github.com/CartoDB/cartodb.git
cd cartodb
```

#### • Install pip

sudo apt-get install python-pip

#### · Install ruby dependencies

```
sudo apt-get install imagemagick unp zip libicu-dev
RAILS_ENV=development bundle install
```

#### · Install python dependencies

```
sudo pip install --no-use-wheel -r python_requirements.txt
```

**Warning:** If this fails due to the installation of the gdal package not finding Python.h or any other header file, you'll need to do this:

```
export CPLUS_INCLUDE_PATH=/usr/include/gdal
export C_INCLUDE_PATH=/usr/include/gdal
export PATH=$PATH:/usr/include/gdal
```

After this, re-run the pip install command. Variables can be passed to sudo if exporting them and re-running pip install doesn't work:

```
sudo CPLUS_INCLUDE_PATH=/usr/include/gdal C_INCLUDE_PATH=/usr/include/gdal PATH=
```

If gdal keeps failing, see more information here: http://gis.stackexchange.com/questions/28966/ python-gdal-package-missing-header-file-when-installing-via-pip

#### • Install Node.js dependencies

npm install

• Compile static assets

```
npm run carto-node && npm run build:static
```

• (Optional) Precompile assets. Needed if you don't want to use CARTO's CDN for assets.

```
export PATH=$PATH:$PWD/node_modules/grunt-cli/bin
bundle exec grunt --environment=development
```

• Create configuration files

```
cp config/app_config.yml.sample config/app_config.yml
cp config/database.yml.sample config/database.yml
```

#### • Start the redis-server that allows access to the SQL and Maps APIs:

sudo systemctl start redis-server

• Initialize the metadata database

```
RAILS_ENV=development bundle exec rake db:create
RAILS_ENV=development bundle exec rake db:migrate
```

• Start Builder's HTTP server

```
RAILS_ENV=development bundle exec rails server
```

#### • In a different process/console start the resque process

```
RAILS_ENV=development bundle exec ./script/resque
```

## **Running CARTO**

#### 4.1 First run, setting up an user

First run, setting up first time to run your development version of CARTO. Let's suppose that we are going to create a development env and that our user/subdomain is going to be 'development'

```
cd cartodb
export SUBDOMAIN=development
# Add entries to /etc/hosts needed in development
echo "127.0.0.1 ${SUBDOMAIN}.localhost.lan" | sudo tee -a /etc/hosts
# Create a development user
sh script/create_dev_user
```

## 4.2 Running all the processes

Start the resque daemon (needed for import jobs):

```
bundle exec script/resque
```

Finally, start the CARTO development server on port 3000:

bundle exec thin start --threaded -p 3000 --threadpool-size 5

Node apps

```
cd cartodb-sql-api && node app.js
cd windshaft-cartodb && node app.js
```

You should now be able to access 'http://<mysubdomain>.localhost.lan:3000' in your browser and login with the password specified above.

Enjoy

## Configuration

In this section you can find some helpful configuration examples related with **Basemaps**, **Domainles Urls** and **Common-data**.

#### 5.1 Basemaps

The way to add/change the basemaps available in CARTO is chaging the config/app\_config.yml. Basically you need to add a new entry called basemaps, that entry can have different sections and each section one or more basemaps.

Each section corresponds to row in CARTO basemap dialog. If the basemaps entry is not present a set of default basemaps will be used (CARTO and Stamen ones, check the default basemaps file https://github.com/CartoDB/ cartodb/blob/master/lib/assets/javascripts/cartodb/table/default\_layers.js)

Also, it's always necessary to have a default basemap among all the configured ones in the app\_config.yml. The way to set a basemap as default a "default" attribute needs to be added to the basemap. There can be several basemaps in the config with the attribute default set, however, only the first one found in the same order than in the app\_config will be used as default.

Here is an example config.yml:

```
basemaps:
CARTO:
positron_rainbow:
    default: true # Ident with spaces not with tab
    url: 'http://{s}.basemaps.cartocdn.com/light_all/{z}/{x}/{y}.png'
    subdomains: 'abcd'
    minZoom: '0'
    maxZoom: '18'
    name: 'Positron'
    className: 'positron_rainbow'
    attribution: '© <a href="http://www.openstreetmap.org/copyright">OpenStreetMap
    </a> contributors © <a href="http://www.openstreetmap.org/copyright">OpenStreetMap
    </a> contributors © <a href="http://carto.com/about-carto/">CARTO</a>'
    dark_matter_rainbow:
    url: 'http://{s}.basemaps.cartocdn.com/dark_all/{z}/{x}/{y}.png'
```

(continues on next page)

(continued from previous page)

```
subdomains: 'abcd'
       minZoom: '0'
       maxZoom: '18'
       name: 'Dark matter'
       className: 'dark_matter_rainbow'
       attribution: '@ <a href="http://www.openstreetmap.org/copyright">OpenStreetMap
↔</a> contributors © <a href="https://carto.com/about-carto/">CARTO</a>'
     positron lite rainbow:
       url: 'http://{s}.basemaps.cartocdn.com/light_nolabels/{z}/{x}/{y}.png'
       subdomains: 'abcd'
       minZoom: '0'
       maxZoom: '18'
       name: 'Positron (lite)'
       className: 'positron_lite_rainbow'
       attribution: '<sup>©</sup> <a href="http://www.openstreetmap.org/copyright">OpenStreetMap
→</a> contributors © <a href="https://carto.com/about-carto/">CARTO</a>'
   stamen:
     toner stamen:
       url: 'https://stamen-tiles-{s}.a.ssl.fastly.net/toner/{z}/{x}/{y}.png'
       subdomains: 'abcd'
       minZoom: '0'
       maxZoom: '18'
       name: 'Toner'
       className: 'toner_stamen'
       attribution: 'Map tiles by <a href="http://stamen.com">Stamen Design</a>,...
→ <a href="http://openstreetmap.org">OpenStreetMap</a>, under <a href="http://www.
→openstreetmap.org/copyright">ODbL</a>.'
```

# 5.2 Basemaps with a layer of labels

Basemaps can optionally add a layer with labels on top of other layers. To do so, you should add the labels key to the basemap config, as follows:

# 5.3 Domainless URLs

Historically, CARTO URLs were based on a username.carto.com/PATH schema. When Multiuser accounts were introduced, an alternate schema organizationname.carto.com/u/username/PATH was built along-

side the "classic" one. Both schemas introduce some problems for opensource and/or custom installs of the platform, as they require DNS changes each time a new user or organization is added.

Subdomainless urls are the answer to this problems. Modifying some configuration settings, any CARTO installation can be setup to work with a new schema, carto.com/user/username/PATH.

The following sections details the steps to make it work and the limitations it has.

#### 5.4 Configuration changes for Domainless URLs

• For a default installation, app\_config.yml contains this relevant values:

```
session_domain: '.localhost.lan'
subdomainless_urls: false
```

• To activate subdomainless urls, change to (notice the removed starting dot from session\_domain:

```
session_domain: 'localhost.lan'
subdomainless_urls: true
```

• Non-default HTTP and HTTPs ports can also be configured here for REST API calls, with the following **app\_config.yml** attributes:

```
# nil/integer. HTTP port to use when building urls.
# Leave empty to use default (80)
http_port:
# nil/integer. HTTPS port to use when building urls.
# Leave empty to use default (443)
https_port:
```

Remember that as with other configuration changes, Rails application must be restarted to apply them.

### 5.5 Limitations

If you leave the dot at session\_domain having subdomainless urls, you will be forced to always have a subdomain. Any will do, but must be present. If you remove the dot it will work as intended without any subdomain.

When subdomainless urls are used, organizations will be ignored from the urls. In fact, typing whatever.carto. com/user/user1 and carto.com/user/user1 is the same. The platform will replicate the sent subdomain fragment to avoid CORS errors but no existing organization checks will be performed. You should be able to use them, assign quota to the organization users, etc.

#### 5.6 Common Data

This service uses the visualizations API to retrieve all the public datasets from a defined user and serve them as importable datasets to all the users of the platform through the data library options.

All can be configured through the common\_data settings section. If the base\_url option is set, this will be the base url the service is going to use to build the URL to retrieve datasets. For example:

```
common_data:
    protocol: 'https'
    username: 'common-data'
    base_url: 'https://common-data.carto.com'
    format: 'shp'
```

Use https://common-data.carto.com as the base url to retrieve all the public datasets from that user.

This is the default behaviour in CARTO, but if you want to use your own system and user for this purpose you have to define the username property pointing to the user that will provide the datasets in your own instance. The URL in this case is going to be built using your instance base url. For example if your instance base url is http://www.example.com and the config is:

```
common_data:
    protocol: 'https'
    username: 'common-data-user'
    format: 'shp'
```

the system populates the data library with the public datasets from http://common-data-user.example. com...

The format option is used to define the format of the file generated when you are importing one datasets from the data library. When you import a dataset it uses a stored URL to download that dataset as a file, in the format defined in the config, and import as your own dataset.

# 5.7 Separate folders

Default installation keeps logs, configuration files and assets under the standard Rails folder structure: /log, / config and /public at Rails root (your installation directory). Some installations might be interested in moving those directories outside Rails root in order to separate code and data. You can accomplish that with symbolic links. Nevertheless, there are three environment variables that you can use instead:

- RAILS\_LOG\_BASE\_PATH: for example, setting it to /var/carto will use that as a base folder for log files, which will be stored at /var/carto/log. Defaults to Rails.root.
- RAILS\_CONFIG\_BASE\_PATH: for example, setting it to /etc/carto will make Rails open the application and database configuration files at /etc/carto/conf/app\_config.yml and /etc/carto/conf/ database.yml. Defaults to Rails.root.
- RAILS\_PUBLIC\_UPLOADS\_PATH: sets assets base path, both static and dynamic. For example, setting it to /var/carto/assets will upload files (markers, avatars and so on) to /var/carto/assets/ uploads, but it also makes Rails server to load public assets (CSSs, JS...) from there. Defaults to app\_config[:importer]["uploads\_path"] or Rails.root if it's not present (due to backwards compatibility). If you use this variable you'll need to do one onf the following:
  - Use nginx to load the assets (recommended): making /public the nginx default root will make nginx use the proper folders for assets, without requesting them to the Rails server: root /opt/carto/ builder/embedded/cartodb/public;.
  - Copy or link assets (from /<RAILS ROOT>/public) to public upload path folder.

## Operations

Certain operations in the CARTO platform can be done using rake tasks or scripts that bundle several of them together. In this section you will instructions to carry out various common operations.

Common operations in CARTO include:

### 6.1 Creating users

Creating users in CARTO is simple thanks to the create\_dev\_user script located in scripts/ create\_dev\_user. To execute this script, be sure to be located at the cartodb repository root directory and simply run:

\$ ./script/create\_dev\_user

You will be prompted to input two parameters:

- **subdomain** is the same as the user's user name. This is what you will enter in the browser to access the user's dashboard: https://<user\_name>.carto.com. Set it to whatever you want the user's user name to be.
- password this is the password the new user will use to login into their account.

Upon script completion, the new user will have been created.

### 6.2 Creating organizations

To create a new organization, a rake task called create\_new\_organization\_with\_owner is used. For this task to work properly, a user **created beforehand** must be provided as an owner (if you're not sure how to create a user, refer to "Creating Users").

In order to create the organization, 4 parameters must be set:

• ORGANIZATION\_NAME is a short nickname for the organization, that may only contain letters, numbers and dash (-) characters. For example, 'cartodb' would be OK.

- ORGANIZATION\_DISPLAY\_NAME is a longer, more beautiful name for the organization. It may contain any characters needed. For example, 'CartoDB Inc.'.
- ORGANIZATION\_SEATS is the number of users that will be able to be created under the organization. For example, 5 seats will mean that a maximum of 5 users can belong to the organization.
- ORGANIZATION\_QUOTA is the space quota in **bytes** that the organization is assigned. For example, 1024 \* 1024 \* 1024 is 1GB of quota.
- USERNAME is the user name of the owner of the organization. In our example, let's assume that our user name is 'manolo'.

This task is executed like:

and an example execution for creating an organization owned by 'manolo', named 'CartoDB Inc.', referred to as 'cartodb', with 5 seats and a 1GB quota, would be:

#### 6.2.1 Seats

You can change the viewer seats:

## 6.3 Changing Feature Flags

CARTO uses feature flags, so different users can have access to different features of CARTO. If you would like to enable or disable feature flags to one or all users or to a given organization, you can use the rake tasks described in this section. Feature flag creation and deletion are also covered.

#### 6.3.1 Enabling a feature for all users

Enabling a feature for all users is done with a rake task called enable\_feature\_for\_all\_users and it takes one parameter.

• feature\_flag\_name is the name of the feature flag to be enabled. For example: 'special\_dashboard'.

This task is executed like:

```
$ bundle exec rake cartodb:features:enable_feature_for_all_users[<feature_flag_name>]
```

And an example to enable the 'special\_dashboard' feature could be:

\$ bundle exec rake cartodb:features:enable\_feature\_for\_all\_users["special\_dashboard"]

#### 6.3.2 Enabling a feature for a given user

Enabling a feature for a given user is done with a rake task called enable\_feature\_for\_user and it takes two parameters.

- feature\_flag\_name is the name of the feature flag to be enabled. For example: 'special\_dashboard'.
- user\_name is the user name of the user to whom the feature flag is to be enabled. For example: 'manolo'.

This task is executed like:

Warning: Please be very careful NOT to leave a space between parameters, as it will cause rake to spit a don't know how to build task type error.

And an example to enable the 'special\_dashboard' feature for user with user name 'manolo' could be:

#### 6.3.3 Enabling a feature for a given organization

Enabling a feature for a given organization is done with a rake task called enable\_feature\_for\_organization and it takes two parameters.

- feature\_flag\_name is the name of the feature flag to be enabled. For example: 'special\_dashboard'.
- organization\_name is the internal name ('cartodb' vs 'CartoDB Inc.') to which the feature flag is to be enabled. For example: 'cartodb'.

This task is executed like:

Warning: Please be very careful NOT to leave a space between parameters, as it will cause rake to spit a don't know how to build task type error.

And an example to enable the 'special\_dashboard' feature for organization 'cartodb' could be:

#### 6.3.4 Disabling a feature for all users

Disabling a feature for all users is done with a rake task called disable\_feature\_for\_all\_users and it takes one parameter.

• feature\_flag\_name is the name of the feature flag to be disabled. For example: 'special\_dashboard'.

This task is executed like:

\$ bundle exec rake cartodb:features:disable\_feature\_for\_all\_users[<feature\_flag\_name>]

And an example to disable the 'special\_dashboard' feature could be:

\$ bundle exec rake cartodb:features:disable\_feature\_for\_all\_users["special\_dashboard"]

#### 6.3.5 Disabling a feature for a given user

Disabling a feature for a given user is done with a rake task called disable\_feature\_for\_user and it takes two parameters.

- feature\_flag\_name is the name of the feature flag to be disabled. For example: 'special\_dashboard'.
- user\_name is the user name of the user to whom the feature flag is to be disabled. For example: 'manolo'.

This task is executed like:

Warning: Please be very careful NOT to leave a space between parameters, as it will cause rake to spit a don't know how to build task type error.

And an example to disable the 'special\_dashboard' feature for user with user name 'manolo' could be:

#### 6.3.6 Disabling a feature for a given organization

Disabling a feature for a given organization is done with a rake task called disable\_feature\_for\_organization and it takes two parameters.

- feature\_flag\_name is the name of the feature flag to be disabled. For example: 'special\_dashboard'.
- organization\_name is the internal name ('cartodb' vs 'CartoDB Inc.') to which the feature flag is to be disabled. For example: 'cartodb'.

This task is executed like:

Warning: Please be very careful NOT to leave a space between parameters, as it will cause rake to spit a don't know how to build task type error.

And an example to disable the 'special\_dashboard' feature for organization 'cartodb' could be:

#### 6.3.7 Adding a feature flag

Adding feature flags should be done using the rake task called add\_feature\_flag. This rake task only takes one argument:

• feature\_flag\_name is the name of the feature flag to be created.

This task is executed like:

\$ bundle exec rake cartodb:features:add\_feature\_flag[<feature\_flag\_name>]

And an example to create a feature flag named "special\_dashboard" could be:

```
$ bundle exec rake cartodb:features:add_feature_flag["special_dashboard"]
```

#### 6.3.8 Removing a feature flag

Removing feature flags should be done using the rake task called remove\_feature\_flag. This rake task only takes one argument:

• feature\_flag\_name is the name of the feature flag to be removed.

This task is executed like:

\$ bundle exec rake cartodb:features:remove\_feature\_flag[<feature\_flag\_name>]

And an example to remove a feature flag named "special\_dashboard" could be:

```
$ bundle exec rake cartodb:features:remove_feature_flag["special_dashboard"]
```

#### 6.3.9 Listing all feature flags

All existing feature flags can be listed using the rake task called list\_all\_features.

This task is executed like:

```
$ bundle exec rake cartodb:features:list_all_features
```

### 6.4 Changing limits

This section explains how to use rake tasks to change several limits for users.

#### 6.4.1 Change user import limits

Using the rake task set\_custom\_limits\_for\_user, you can change import limits for a given user. The parameters this task takes are:

- user\_name is the user name of the user for whom these limits will be changed.
- import\_file\_size is the maximum size in bytes for a file to be imported.
- table\_row\_count is the maximum number of rows for a file to be imported.
- concurrent\_imports is the maximum number of concurrent imports that can be handled at once.

This task is executed like:

```
$ bundle exec rake cartodb:set_custom_limits_for_user[<user_name>,<import_file_size>,

$ <table_row_count>,<concurrent_imports>]
```

and an example execution could be:

\$ bundle exec rake cartodb:set\_custom\_limits\_for\_user["manolo","1048576","50000","5"]

#### 6.4.2 Increasing Twitter imports limit

Increasing the Twitter imports limit should done using rake task increase\_limits\_for\_twitter\_import\_users. This rake task takes no parameters. Upon execution, all users with Twitter imports enabled will have 1500MB of file size quota and a 5M row quota limit.

## 6.5 Exporting/Importing visualizations

You might be interested in exporting and importing visualizations because of several reasons:

- · Backup purposes.
- Moving visualizations between different hosts.
- Moving visualizations between users.
- Using the same visualization with different data.

With cartodb:vizs:export\_user\_visualization\_json task you can export a visualization to JSON, and with cartodb:vizs:import\_user\_visualization\_json you can import it. First outputs to stdout and second reads stdin.

This example exports c54710aa-ad8f-11e5-8046-080027880ca6 visualization.

```
$ bundle exec rake cartodb:vizs:export_user_visualization_json['c54710aa-ad8f-11e5-

$$046-080027880ca6'] > c54710aa-ad8f-11e5-8046-080027880ca6.json
```

and this imports it into 6950b745-5524-4d8d-9478-98a8a04d84ba user, who is in another server.

\$ cat c54710aa-ad8f-11e5-8046-080027880ca6.json | bundle exec rake\_ →cartodb:vizs:import\_user\_visualization\_json['6950b745-5524-4d8d-9478-98a8a04d84ba']

Please keep in mind the following:

- Exporting has **backup purposes**, so it keeps ids. If you want to use this to replicate a visualization in the same server you can edit the JSON and change the ids. Any valid, distinct UUID will work.
- It **does export neither the tables nor its data**. Destination user should have tables with the same name than the original one for the visualization to work. You can change the table names in the JSON file if names are different.

## 6.6 Exporting/Importing full visualizations

Disclaimer: this feature is still in beta

You can export a complete visualization (data, metadata and map) with this command: bundle exec rake cartodb:vizs:export\_full\_visualization['5478433b-b791-419c-91d9-d934c56f2053']

That will generate a .carto file that you can import in any CARTO installation just dropping the file as usual.

### 6.7 Running Sync Tables

If you are working with the **Sync Tables** feature, you must run a rake task to trigger the synchronization of the dataset. This rake retrieves all sync tables that should get synchronized, and puts the synchronization tasks at Resque:

```
bundle exec rake cartodb:sync_tables[true]
```

You might want to set up a cron so that this task is executed periodically in an automated way.

### 6.8 HTTP Header Authentication

With web servers such as NGINX or others you can perform SSO by making the web server add a trusted, safe header to every request sent to CARTO. Example:

User browser - GET http://myorg.mycompany.lan/dashboard -> NGINX (adds 'sso-user-email': 'alice@myorg.com' header) -> CARTO server

You can enable HTTP Header Authentication at CARTO by adding the following to app\_conf.yml (taken from app\_conf.yml.sample):

```
http_header_authentication:
    header: # name of the trusted, safe header that your server adds to the request
    field: # 'email' / 'username' / 'id' / 'auto' (autodetection)
    autocreation: # true / false (true requires field to be email)
```

Configuration for the previous example:

```
http_header_authentication:
    header: 'sso-user-email'
    field: 'email'
    autocreation: false
```

#### 6.8.1 Autocreation

Even more, if you want not only *authentication* (authenticating existing users) but also *user creation* you can turn autocreation on by setting autocreation: true. If you do so, when a user with the trusted header performs their first request the user will be created automatically. This feature requires that field is set to email, since the new user will be created with it:

- email: value of the header (alice@myorg.com).
- username: user of the email (alice).
- password: random. they can change it in their account page.
- organization: taken from the subdomain (myorg).

# 6.9 Configuring Dataservices

The services provided by the Dataservices SQL extension can be manually configured for users or organizations through the following rake tasks.

The service configuration is stored in the users and organization metadata tables and reflected in the REDIS configuration database.

#### 6.9.1 Service provider

For each basic service class (geocoder, routing, isolines) a provider can be assigned with the tasks:

- cartodb:services:set\_user\_provider[username, service, provider]
- cartodb:services:set\_organization\_provider[orgname,service,provider]

Valid providers are:

- mapzen for Mapzen-based third party services (Mapzen Search/Mapzen Mobility)
- here for HERE maps third party services (HERE Geocoder/Routing APIs)
- google for Google Maps services

#### **Examples:**

Set geocoder provider for user user-name to Mapzen:

rake cartodb:services:set\_user\_provider['user-name',geocoder,mapzen]

Set geocoder provider for organization org-name to Mapzen:

rake cartodb:services:set\_organization\_provider['org-name',geocoder,mapzen]

#### 6.9.2 Quotas

Service limits can be established for individual users or organizations in the form of quotas (maximum number of requests per billing period). The next tasks manage the configuration of the quotas.

- cartodb:services:set\_user\_quota[username,service,quota]
- cartodb:services:set\_org\_quota[orgname, service, quota]

Quota values must be non-negative integers.

Valid services are:

- geocoding for street-level (hi-res) geocoding services.
- here\_isolines for general isolines/isochrones zoning services (despite the name, this does not only applies to Here-provided services).
- obs\_snapshot for Data Observatory snapshot services.
- obs\_general for Data Observatory general services.
- mapzen\_routing for general routing services (again, not necessarily provided by Mapzen).

#### **Examples:**

Set geocoding quota for user user-name to 1000 monthly requests:

```
rake cartodb:services:set_user_quota['user-name',geocoder,1000]
```

Set geocoder quota for organization org-name to 1000 monthly requests:

```
rake cartodb:services:set_org_quota['org-name',geocoder,1000]
```

#### 6.9.3 Soft limits

The service limits for a user can be configured to be *soft*, meaning that the user can exceed the limits (possibly incurring in additional charges). So, when soft limits is set to *false* (the default) and the limits are exceeded, service requests will fail, while if the soft limits are set to *true* requests will succeed (and the user will be charged for the excess).

• cartodb:services:set\_user\_soft\_limit[username,service,soft\_limit\_status]

The possible values for the soft limits status is either true or false.

Valid services are, as for the quota configuration:

- geocoding for street-level (hi-res) geocoding services.
- here\_isolines for general isolines/isochrones zoning services (despite the name, this does not only applies to Here-provided services).
- obs\_snapshot for Data Observatory snapshot services.
- obs\_general for Data Observatory general services.
- mapzen\_routing for general routing services (again, not necessarily provided by Mapzen).

#### **Examples:**

Activate soft geocoding limits for user user-name:

rake cartodb:services:set\_user\_soft\_limit['user-name',geocoder,true]

Disable soft geocoding limits for user user-name:

rake cartodb:services:set\_user\_soft\_limit['user-name',geocoder,false]

#### 6.9.4 User database configuration (from Builder)

Add the following entry to the geocoder entry of the cartodb/config/app\_config.yml file:

```
api:
    host: 'localhost'
    port: '5432'
    user: 'dataservices_user'
    dbname: 'dataservices_db'
```

In the *cartodb/config/app\_config.yml* file, enable the desired dataservices:

```
enabled:
    geocoder_internal: false
    hires_geocoder: false
    isolines: false
    routing: false
    data_observatory: true
```

Execute the rake tasks to update all the users and organizations:

bundle exec rake cartodb:db:configure\_geocoder\_extension\_for\_organizations['',
true]

bundle exec rake cartodb:db:configure\_geocoder\_extension\_for\_non\_org\_users['',
true]

Indices and tables

- genindex
- modindex
- search