We have followed the steps provided in below link for installation procedure:

https://github.com/wrouesnel/postgres_exporter

PostgreSQL Server Exporter

Prometheus exporter for PostgreSQL server metrics.

CI Tested PostgreSQL versions: `9.4`, `9.5`, `9.6`, `10`, `11`

Quick Start

This package is available for Docker:

```
# Start an example database
docker run --net=host -it --rm -e POSTGRES_PASSWORD=password postgres
# Connect to it
docker run --net=host -e
DATA_SOURCE_NAME="postgresql://postgres:password@localhost:5432/postgres?sslmode=disa
ble" wrouesnel/postgres_exporter
```

Building and running

The build system is based on Mage

The default make file behavior is to build the binary:

```
$ go get github.com/wrouesnel/postgres_exporter
$ cd ${GOPATH-$HOME/go}/src/github.com/wrouesnel/postgres_exporter
$ go run mage.go binary
$ export DATA_SOURCE_NAME="postgresql://login:password@hostname:port/dbname"
$ ./postgres_exporter <flags>
```

To build the dockerfile, run `go run mage.go docker`.
This will build the docker image as `wrouesnel/postgres_exporter:latest`. This is a minimal docker image containing *just*postgres_exporter. By default no SSL certificates are included, if you need to use SSL you should either bind-mount`/etc/ssl/certs/ca-certificates.crt` or derive a new image containing them.

Flags

- `web.listen-address` Address to listen on for web interface and telemetry. Default is `:9187`.
- `web.telemetry-path` Path under which to expose metrics. Default is `/metrics`.
- `disable-default-metrics` Use only metrics supplied from `queries.yaml` via `--extend.query-path`.
- `disable-settings-metrics` Use the flag if you don't want to scrape `pg_settings`.
- `extend.query-path` Path to a YAML file containing custom queries to run. Check out `queries.yaml` for examples of the format.
- `dumpmaps` Do not run - print the internal representation of the metric maps. Useful when debugging a custom queries file.
- `log.level` Set logging level: one of `debug`, `info`, `warn`, `error`, `fatal`
- `log.format` Set the log output target and format. e.g. `logger:syslog?appname=bob&local=7` or `logger:stdout?json=true`Defaults to `logger:stderr`.
- `constantLabels` Labels to set in all metrics. A list of `label=value` pairs, separated by commas.

Environment Variables

The following environment variables configure the exporter:

- `DATA_SOURCE_NAME` the default legacy format. Accepts URI form and key=value form arguments. The URI may contain the username and password to connect with.
- `DATA_SOURCE_URI` an alternative to `DATA_SOURCE_NAME` which exclusively accepts the raw URI without a username and password component.
- `DATA_SOURCE_USER` When using `DATA_SOURCE_URI`, this environment variable is used to specify the username.
- `DATA_SOURCE_USER_FILE` The same, but reads the username from a file.
- `DATA_SOURCE_PASS` When using `DATA_SOURCE_URI`, this environment variable is used to specify the password to connect with.
- `DATA_SOURCE_PASS_FILE` The same as above but reads the password from a file.
- `PG_EXPORTER_WEB_LISTEN_ADDRESS` Address to listen on for web interface and telemetry. Default is `:9187`.
- `PG_EXPORTER_WEB_TELEMETRY_PATH` Path under which to expose metrics. Default is `/metrics`.
- `PG_EXPORTER_DISABLE_DEFAULT_METRICS` Use only metrics supplied from `queries.yaml`. Value can be `true` or `false`. Default is `false`.
- `PG_EXPORTER_DISABLE_SETTINGS_METRICS` Use the flag if you don't want to scrape `pg_settings`. Value can be `true` or `false`. Defauls is `false`.

- `PG_EXPORTER_EXTEND_QUERY_PATH` Path to a YAML file containing custom queries to run. Check out `queries.yaml` for examples of the format.
- `PG_EXPORTER_CONSTANT_LABELS` Labels to set in all metrics. A list of `label=value` pairs, separated by commas.

Settings set by environment variables starting with `PG_` will be overwritten by the corresponding CLI flag if given.

Setting the Postgres server's data source name

The PostgreSQL server's data source name must be set via the `DATA_SOURCE_NAME` environment variable.
For running it locally on a default Debian/Ubuntu install, this will work (transpose to init script as appropriate):

```
sudo -u postgres DATA_SOURCE_NAME="user=postgres host=/var/run/postgresql/
sslmode=disable" postgres_exporter
```

Also, you can set a list of sources to scrape different instances from the one exporter setup. Just define a comma separated string.

```
sudo -u postgres DATA_SOURCE_NAME="port=5432,port=6432" postgres_exporter
```

See the github.com/lib/pq module for other ways to format the connection string.

Running as non-superuser

To be able to collect metrics from `pg_stat_activity` and `pg_stat_replication` as non-superuser you have to create functions and views as a superuser, and assign permissions separately to those.
In PostgreSQL, views run with the permissions of the user that created them so they can act as security barriers. Functions need to be created to share this data with the non-superuser. Only creating the views will leave out the most important bits of data.

```
-- To use IF statements, hence to be able to check if the user exists before
-- attempting creation, we need to switch to procedural SQL (PL/pgSQL)
-- instead of standard SQL.
-- More: https://www.postgresql.org/docs/9.3/plpgsql-overview.html
-- To preserve compatibility with <9.0, DO blocks are not used; instead,
-- a function is created and dropped.
CREATE OR REPLACE FUNCTION __tmp_create_user() returns void as $$
BEGIN
  IF NOT EXISTS (
          SELECT                       -- SELECT list can stay empty for this
          FROM   pg_catalog.pg_user
          WHERE  usename = 'postgres_exporter') THEN
    CREATE USER postgres_exporter;
  END IF;
```

```sql
END;
$$ language plpgsql;

SELECT __tmp_create_user();
DROP FUNCTION __tmp_create_user();

ALTER USER postgres_exporter WITH PASSWORD 'password';
ALTER USER postgres_exporter SET SEARCH_PATH TO postgres_exporter,pg_catalog;

-- If deploying as non-superuser (for example in AWS RDS), uncomment the GRANT
-- line below and replace <MASTER_USER> with your root user.
-- GRANT postgres_exporter TO <MASTER_USER>;
CREATE SCHEMA IF NOT EXISTS postgres_exporter;
GRANT USAGE ON SCHEMA postgres_exporter TO postgres_exporter;

CREATE OR REPLACE FUNCTION get_pg_stat_activity() RETURNS SETOF pg_stat_activity AS
$$ SELECT * FROM pg_catalog.pg_stat_activity; $$
LANGUAGE sql
VOLATILE
SECURITY DEFINER;

CREATE OR REPLACE VIEW postgres_exporter.pg_stat_activity
AS
  SELECT * from get_pg_stat_activity();

GRANT SELECT ON postgres_exporter.pg_stat_activity TO postgres_exporter;

CREATE OR REPLACE FUNCTION get_pg_stat_replication() RETURNS SETOF
pg_stat_replication AS
$$ SELECT * FROM pg_catalog.pg_stat_replication; $$
LANGUAGE sql
VOLATILE
SECURITY DEFINER;

CREATE OR REPLACE VIEW postgres_exporter.pg_stat_replication
AS
  SELECT * FROM get_pg_stat_replication();

GRANT SELECT ON postgres_exporter.pg_stat_replication TO postgres_exporter;
```