pgsql stub Documentation

Release 1.0

me

July 15, 2015

Contents

Contents:

CHAPTER 1

Requires: PostgreSQLClient

1.1 Example Usage

This is what a charm using this relation would look like:

```
from charmhelpers.core import hookenv
from charmhelpers.core.reactive import hook
from charmhelpers.core.reactive import when
from charmhelpers.core.reactive import when_file_changed
from charmhelpers.core.reactive import set_state
from charmhelpers.core.reactive import remove_state

@hook('db-relation-joined')
def request_db(pgsql):
    pgsql.change_database_name('mydb')
    pgsql.request_roles('myrole', 'otherrole')

@hook('config-changed')
def check_admin_pass():
    admin_pass = hookenv.config('admin-pass')
    if admin_pass:
        set_state('admin-pass')
    else:
        remove_state('admin-pass')

@when('db.database.available', 'admin-pass')
def render_config(pgsql):
    render_template('app-config.j2', '/etc/app.conf', {
        'db_conn': pgsql.connection_string(),
        'admin_pass': hookenv.config('admin-pass'),
    })

@when_file_changed('/etc/app.conf')
def restart_service():
    hookenv.service_restart('myapp')
```

1.2 Reference

class requires.**PostgreSQLClient**(*relation_name*, *conversations=None*)

change_database_name(*dbname*)
Tell the PostgreSQL server to provide us with a database with a specific name.

Parameters **dbname** (*str*) – New name for the database to use.

connection_string()
Get the connection string, if available, or None.

database()
Get the database, if available, or None.

host()
Get the host, if available, or None.

password()
Get the password, if available, or None.

port()
Get the port, if available, or None.

request_roles(**roles*)
Tell the PostgreSQL server to provide our user with a certain set of roles.

Parameters **roles** (*list*) – One or more role names to give to this service's user.

schema_password()
Get the schema_password, if available, or None.

schema_user()
Get the schema_user, if available, or None.

user()
Get the user, if available, or None.

CHAPTER 2

Provides: PostgreSQL

2.1 Example Usage

This is what a charm using this relation would look like:

```
# in the postgres charm:
from charmhelpers.core import hookenv  # noqa
from charmhelpers.core import unitdata
from charmhelpers.core.reactive import when
from common import (
    user_name,
    create_user,
    reset_user_roles,
    ensure_database,
    get_service_port,
)


@when('db.roles.requested')
def update_roles(pgsql):
    for service, roles in pgsql.requested_roles():
        user = user_name(pgsql.relation_name(), service)
        reset_user_roles(user, roles)
        pgsql.ack_roles(service, roles)


@when('db.database.requested')
def provide_database(pgsql):
    for service, database in pgsql.requested_databases():
        if not database:
            database = service
        roles = pgsql.requested_roles(service)

        user = user_name(pgsql.relation_name(), service)  # generate username
        password = create_user(user)  # get-or-create user
        schema_user = "{}_schema".format(user)
        schema_password = create_user(schema_user)

        reset_user_roles(user, roles)
        ensure_database(user, schema_user, database)

        pgsql.provide_database(
            service=service,
```

```
            host=hookenv.unit_private_ip(),
            port=get_service_port(),
            database=database,
            state=unitdata.kv().get('pgsql.state'),  # master, hot standby, standalone
            user=user,
            password=password,
            schema_user=schema_user,
            schema_password=schema_password,
        )
```

2.2 Reference

class provides.**PostgreSQL**(*relation_name*, *conversations=None*)

ack_roles(**args*, ***kwargs*)
Acknowledge that a set of roles have been given to a service's user.

Parameters **service** (*str*) – The service which requested the roles, as returned by *requested_roles()*.

joined_changed()
Handles the relation-joined and relation-changed hook.

Depending on the state of the conversation, this can trigger one of the following states:

- {relation_name}.database.requested This state will be activated if the remote service has requested a different database name than the one it has been provided. This state should be resolved by calling provide_database(). See also requested_databases().
- {relation_name}.roles.requested This state will be activated if the remote service has requested a specific set of roles for its user. This state should be resolved by calling ack_roles(). See also requrested_roles().

previous_database(*service*)
Return the roles previously requested, if different from the currently requested roles.

previous_roles(*service*)
Return the roles previously requested, if different from the currently requested roles.

provide_database(**args*, ***kwargs*)
Provide a database to a requesting service.

Parameters

- **service** (*str*) – The service which requested the database, as returned by *requested_databases()*.
- **host** (*str*) – The host where the database can be reached (e.g., the charm's private or public-address).
- **port** (*int*) – The port where the database can be reached.
- **database** (*str*) – The name of the database being provided.
- **user** (*str*) – The username to be used to access the database.
- **password** (*str*) – The password to be used to access the database.
- **schema_user** (*str*) – The username to be used to admin the database.

- **schema_password** (*str*) – The password to be used to admin the database.
- **state** (*str*) – I have no idea what this is for. TODO: Document this better

requested_database(*service*)

Return the database name requested by the given service. If the given service has not requested a specific database name, an empty string is returned, indicating that the database name should be generated.

requested_databases()

Return a list of tuples mapping a service name to the database name requested by that service. If a given service has not requested a specific database name, an empty string is returned, indicating that the database name should be generated.

Example usage:

```
for service, database in pgsql.requested_databases():
    database = database or generate_dbname(service)
    pgsql.provide_database(**create_database(database))
```

requested_roles(*service=None*)

Return the roles requested by all or a single given service.

Parameters **service** (*str*) – The name of a service requesting roles, as provided by either `requested_roles()` (with no args) or `requested_databases()`.

Returns If no service name is given, then a list of `(service, roles)` tuples are returned, mapping service names to their requested roles. If a service name is given, a list of the roles requested for that service is returned.

Example usage:

```
for service, roles in pgsql.requested_roles():
    set_roles(username_from_service(service), roles)
    pgsql.ack_roles(service, roles)
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

A

C

D

H

J

P

R

S

U