pgctl Documentation

Release 0.1.1

Buck Evan

August 24, 2015

Contents

Issues | Github | PyPI

Release v0.1. (*Installation*)

CHAPTER 1

Introduction

`pgctl` is an MIT Licensed tool to manage developer "playgrounds".

Often projects have various processes that should run in the backround (*services*) during development. These services amount to a miniature staging environment that we term *playground*. Each service must have a well-defined state at all times (it should be starting, up, stopping, or down), and should be independantly restartable and debuggable.

`pgctl` aims to solve this problem in a unified, language-agnostic framework (although the tool happens to be written in Python).

As a simple example, let's say that we want a *date* service in our playground, that ensures our *now.date* file always has the current date.

```
$ cat playground/date/run
date > now.date

$ pgctl-2015 start
$ pgctl-2015 status
date -- up (0 seconds)

$ cat now.date
Fri Jun 26 15:21:26 PDT 2015

$ pgctl-2015 stop
$ pgctl-2015 status
date -- down (0 seconds)
```

CHAPTER 2

Feature Support

- User-friendly Command Line Interface
- Simple Configuration
- Python 2.6—3.4

CHAPTER 3

User Guide

This part of the documentation covers the step-by-step instructions and usage of `pgctl` for getting started quickly.

3.1 Installation

This part of the documentation covers the installation of pgctl. The first step to using any software package is getting it properly installed.

3.1.1 Distribute & Pip

Installing pgctl is simple with pip, just run this in your terminal:

```
$ pip install pgctl
```

3.1.2 Get the Code

pgctl is actively developed on GitHub, where the code is always available.

You can either clone the public repository:

```
$ git clone git://github.com/yelp/pgctl.git
```

Download the tarball:

```
$ curl -OL https://github.com/yelp/pgctl/tarball/master
```

Or, download the zipball:

```
$ curl -OL https://github.com/yelp/pgctl/zipball/master
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages easily:

```
$ python setup.py install
```

3.2 User Guide

3.2.1 Usage

`pgctl` has eight basic commands: `start`, `stop`, `restart`, `debug`, `status`, `log`, `reload`, `config`

Note: With no arguments, `pgctl <cmd>` is equivalent to `pgctl <cmd> default`. By default, default maps to all services. See *Aliases*.

---

start

```
$ pgctl start <service=default>
```

Starts a specific service, group of services, or all services. This command is blocking until all services have successfully reached the up state. `start` is idempotent.

stop

```
$ pgctl stop <service=default>
```

Stops a specific service, group of services, or all services. This command is blocking until all services have successfully reached the down stated. `stop` is idempotent.

restart

```
$ pgctl restart <service=default>
```

Stops and starts specific service, group of services, or all services. This command is blocking until all services have successfully reached the down stated.

debug

```
$ pgctl debug <service=default>
```

Runs a specific service in the foreground.

status

```
$ pgctl status <service=default>
<service> (pid <PID>) -- up (0 seconds)
```

Retrieves the state, PID, and time in that state of a specific service, group of services, or all services.

log

```
$ pgctl log <service=default>
```

Retrieves the stdout and stderr for a specific service, group of services, or all services.

reload

```
$ pgctl reload <service=default>
```

Reloads the configuration for a specific service, group of services, or all services.

config

```
$ pgctl config <service=default>
```

Prints out a configuration for a specific service, group of services, or all services.

3.3 Quickstart

This page attempts to be a quick-and-dirty guide to getting started with pgctl.

3.3.1 Setting up

The minimal setup for pgctl is a `playground` directory containing the services you want to run. A service consists of a directory with a `run` script. The script should run in the foreground.

```
$ cat playground/date/run
date > now.date
```

Once this is in place, you can start your playground and see it run.

```
$ pgctl start
$ pgctl logs
[webapp] Serving HTTP on 0.0.0.0 port 36474 ...

$ curl
```

3.3.2 Aliases

With no arguments, `pgctl start` is equivalent to `pgctl start default`. By default, `default` maps to a list of all services. You can configure what `default` means via `playground/config.yaml`:

```
aliases:
    default:
        - service1
        - service2
```

You can also add other aliases this way. When you name an alias, it simply expands to the list of configured services, so that `pgctl start A-and-B` would be entirely equivalent to `pgctl start A B`.

CHAPTER 4

Developer Guide

This part of the documentation gives an internal look at the design decisions for pgctl.

4.1 Developers

4.1.1 Directory Structure

```
$ pwd
/home/<user>/<project>

$ tree playground/
playground/
-- service1
|   -- down
|   -- run
|   -- stderr.log
|   -- stdout.log
|   -- supervise -> ~/.run/pgctl/home/<user>/<project>/playground/service1/supervise
-- service2
|   -- down
|   -- run
|   -- stderr.log
|   -- stdout.log
|   -- supervise -> ~/.run/pgctl/home/<user>/<project>/playground/service2/supervise
-- service3
    -- down
    -- run
    -- stderr.log
    -- stdout.log
    -- supervise -> ~/.run/pgctl/home/<user>/<project>/playground/service3/supervise
```

There are a few points to note: logging, services, state, symlinking.

logging

stdin and stdout will be captured from the supervised process and written to log files under the service directory. The user will be able to use the `pgctl logs` command to aggregate these logs in a readable form.

services

All services are located under the playground directory.

state

We are using daemontools for process management and call the daemontools `supervise` command directly. It was a design decision to not use `svscan` to automatically supervise all services. This was due to inflexability with logging (by default stdout is only logged). To ensure that every service is in a consistent state, a down file is added to each service directory (man supervise) if it does not already exist.

symlinking

Currently `pip install .` calls shutil.copy to copy all files in the current project when in the project's base directory. Having pipes present in the projects main directory attempts to copy the pipe and deadlocks. To remedy this situation, we have symlinked the supervise directory to the user's home directory to prevent any pip issues.

4.1.2 Design Decisions

Design of debug

Unsupervise all things when down

CHAPTER 5

API Documentation

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

5.1 pgctl package

5.1.1 Submodules

5.1.2 pgctl.cli module

5.1.3 Module contents

CHAPTER 6

Contributor Guide

If you want to contribute to the project, this part of the documentation is for you.