
Ground Control

Ground Control is a Go based daemon that runs on your Pi and lets you manage and monitor it with ease.

See a screenshot of the management UI and my Pi's temperature on [Librato](#).

See [FAQ](#) for some common question that got asked on the Hacker News thread.

Update: I just pushed groundcontrol-ui if you want to hack on the UI part of groundcontrol.

Usage

Download the Ground Control package, or build from source (see [Development](#)).

Then, transfer it to your Pi.

```
1 $ scp groundcontrol-v0.0.1.tar.gz pi-user@PI_HOST:
```

or download it directly on your Pi

```
1 $ wget http://jondot.github.io/groundcontrol/groundcontrol-0.0.1.tar.gz
```

On the Pi, extract and change directories.

```
1 $ tar zxvf groundcontrol-0.0.1.tar.gz
2 $ cd groundcontrol-0.0.1/
```

Run Ground Control with a simple command (you should have [config.json.sample](#) there for a quick start).

```
1 $ ./groundcontrol -config myconfig.json
```

You can access the UI from your browser on port 4571.

```
1 http://PI_HOST:4571/
```

For configuration, use [groundcontrol.json.sample](#) as a basis and make sure to customize these fields:

- [librato](#) - You can make a free account at Librato and then drop the key and user there.
- [tempodb](#) - You can make a free account at TempoDB and then drop the key and user there.
- [Graphite](#) or [hostedgraphite](#) - You can use your own standard Graphite server, or you can make a 14-day trial account at Hosted Graphite and then drop the key as a prefix. **Important:** for [prefix](#) specify a trailing dot . and [postfix](#) a leading dot ., if you want them.

Here's a typical graphite config:

```
1  "graphite" : {
2    "prefix" : "prefix-or-key.",
3    "postfix" : ".ip-pi",
4    "linerec": "localhost:2003"
5  },
```

Make sure to go over the plans (paid and free) and see what fits you best.

Both Librato and TempoDB were included because they have different retention and resolution and features for the free plans.

Next up, set up your “controls”. This is where you input a label, and an “on”, “off”, “once” commands to automatically build a GUI around it.

Here is an example of having an `xbmc` control. It allows for shutting down and turning on XBMC.

```
1  "controls" : {
2    "xbmc": {
3      "on" : "/etc/init.d/xbmc start",
4      "off" : "/etc/init.d/xbmc stop"
5    }
6  }
```

init.d

You might want to use an `init.d` or `upstart` script to keep `groundcontrol` up at all times.

An default `init.d` script is included here [support/init.d/groundcontrol](#).

Place your `groundcontrol` folder at:

```
1 /opt/groundcontrol/
```

And configuration should be at:

```
1 /etc/groundcontrol.json
```

You can edit your `support/init.d/groundcontrol` if you want to modify these paths.

After you've verified `/etc/init.d/groundcontrol start` to be working, to set up the default run order you can use:

```
1 $ update-rc.d groundcontrol defaults
```

FAQ

Q: Is this specific to the RaspberryPi?

A: Nope. Mechanically, it was built to work on any Unix like environment - just in case. However, the fact that Go makes such a slim resource profile, and a cross-compilation toolkit that works well makes it perfect for it (takes very little resource).

Q: Why was this made?

A: So here we go:

- For fun (as said here)
- Scratching my own itch - I needed a way to remotely run commands though a nice UI, and a way to see how my Pi is doing when I'm not at home.
- For lack of better tooling - every thing I evaluated needed a combination of things, no other tool gave me all-in-one. This made the resources bloated. With GC, you get a few megabytes of memory usage.
- To prove to myself that Go can be as great for development on the Pi as Python (which many people use there) I also like the idea of Internet of Things http://en.wikipedia.org/wiki/Internet_of_Things

Q: Does it need root?

A: Not necessarily. Since it runs shell commands for you exposed through REST, it boils down to whether your commands require root (example for these is starting/stopping services)

Q: Does it work on Windows?

A: No. For health collection it uses the production grade library sigar which support unixy environments. Thankfully, there was a go port of it.

Q: Can you do the same thing with other tools

A: Yes. I would opt for Collectd with a good set of plugins and which is C based. You'll have to make sure there's a plugin for your choice of metrics database. Then write some kind of Web endpoint in Python to execute shell commands. However as I mentioned before, sum up the resources of those, and you'll get a bigger consumption.

More Details

There's plenty more to Ground Control under the hood, let's list out a few things.

Temperature Monitoring

Ground control will read a file containing a temperature reading to update its own health records.

This is made so the mechanism is flexible – you can use Ground Control on any machine (not just a Pi) as long as it will expose a temperature reading in a file-like device.

Controls

You can add and remove controls (commands that your Ground Control can run) by editing or specifying them with your configuration file..

Here's a full description of the format:

```
1 {
2   "control_name": {
3     "on" : "cmd for on, normally a 'start' for a service",
4     "off" : "cmd for off, normally a 'stop' for a service",
5     "status" : "A command that returns the status of the process",
6     "once" : "a one time command, a cleanup, a shutdown etc."
7   }
8 }
```

By convention `control_name` is snake_case and we turn it into “Control Name” on the UI.

The name should be nice for using in a REST API, in this case the commands turn into:

```
1 POST controls/control_name/on
2 POST controls/control_name/off
3 POST controls/control_name/once
4 GET controls/control_name/status
```

The `on`, `off`, and `once` commands are async, and we return a 202 OK for success. The `status` command are async, and we return a 200 OK for success.

And you can easily build an app (mobile?) yourself that makes use of those.

Development

Here's a short guide if you want to experiment with Ground Control yourself.

In each case, start off by taking a Ground Control repo clone.

Compiling

Set up dependencies and build:

```
1 $ go get github.com/jondot/gosigar
2 & go build
```

Cross Compiling

You probably want to build on your own (much more powerful) system rather than on the Pi itself to save time.

In my case I'll be compiling a Go binary on a Mac (OSX, x64), for a Raspberry Pi (Linux, ARMv5/6).

Here's how to do it on a Mac and `brew`:

```
1 $ brew install go --devel --cross-compile-all # I usually take --
    devel with Go, drop if you don't.
```

If you've got ZShell, or a nice alias-supporting shell, this is a nice alias:

```
1 alias go-pi='GOARCH=arm GOARM=5 GOOS=linux go'
```

and then I just

```
1 $ go-pi build
```

If you don't want to use an alias then this is the command to cross-compile for the Pi:

```
1 $ GOARCH=arm GOARM=5 GOOS=linux go build
```

Implementation Details

Ground control surrounds around several concepts:

- Reporter - an entity that takes a Health, and reports it to somewhere.
- Health - the entity that's responsible to gather all of the important health metrics.
- Control - a switchboard-like entity that runs commands on request.

They are sorted by the level of fun/hackability you can get from it, but YMMV :).

Note, that `go-metrics` for example, could have replaced the entire reporter stack here using its various pluggable reporters, however, it only supports integers out of the box.

At the worst case, `go-metrics` itself can be implemented as a reporter (in fact it will be an aggregate reporter of reporters :).

Contributing

Fork, implement, add tests, pull request, get my everlasting thanks and a respectable place here :).

Copyright

Copyright (c) 2013 Dotan Nahum @jondot. See MIT-LICENSE for further details.