provy Documentation

Release 0.7.0-dev

Bernardo Heynemann

June 08, 2015

Contents

Python provisioning made **easy**!

CHAPTER 1

What's provy and provisioning

According to Wikipedia, provisioning is "the process of preparing and equipping a network to allow it to provide (new) services to its users".

We'll draw from this concept the part of preparing and equipping.

provy is a infrastructure provisioning automation tool. Its main goal is making it easy to create highly-scalable architectures with simple scripts.

provy stands on the shoulder of giants! fabric for the networking part and jinja2 for templating capabilities.

A very simple, yet working example of a valid provyfile.py:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from provy.core import Role
from provy.more.debian import UserRole, AptitudeRole

class SimpleServer(Role):
    def provision(self):
        with self.using(UserRole) as role:
            role.ensure_user('my-user', identified_by='my-pass', is_admin=True)

        with self.using(AptitudeRole) as role:
            role.ensure_package_installed('vim')

servers = {
    'frontend': {
        'address': '33.33.33.33',
        'user': 'root',
        'roles': [
            SimpleServer
        ]
    }
}
```

CHAPTER 2

Changelog

These are the changes in the project, from the latest version on top to the earlier ones.

2.1 0.6.1

- Documentation and website completely revamped!
- Replaced M2Crypto with PyCrypto
- New methods:
  - `provy.core.roles.Role.execute_python_script()`
  - `provy.core.roles.Role.remote_list_directory()`
  - `provy.core.roles.Role.create_remote_temp_file()`
  - `provy.core.roles.Role.create_remote_temp_dir()`
  - `provy.more.debian.package.virtualenv.VirtualenvRole.get_base_directory()`
- Changed from personal project to community-managed project in GitHub
- Small changes in the API (we tried to change the least that we could, and reduce impact as much as possible)
- Code coverage raised to 75%, as measured at the time of the release
- Multiple bugfixes

CHAPTER 3

Installing provy

Before installing provy you will need to ensure you have swig installed, as m2crypto needs it. Here is how to install it in some platforms.

3.1 Install the OS dependencies

3.1.1 MacOSX

To install swig on a mac, the easiest way is to install using the homebrew package manager (which we will not cover here). After installing it, execute this command:

```
brew install https://raw.github.com/cobrateam/formulae/master/swig.rb
```

3.1.2 Ubuntu and Debian GNU/Linux

It is just an apt-get install away =)

```
$ sudo apt-get install swig
```

3.1.3 Arch Linux

Swig is in the extra repository and can be installed with:

```
$ sudo pacman -S swig
```

3.1.4 Other platforms

If your platform is not listed above, try searching in your package manager for *swig* and install it given the search results.

3.2 Install provy

Now that you have *swig*, installing *provy* is as easy as:

```
$ pip install provy
```

It can be easily installed from source as well, like this:

```
$ # make sure fabric, jinja2 and m2crypto are installed
$ pip install fabric
$ pip install jinja2
$ pip install m2crypto

$ # now actually installing it
$ git clone git@github.com:heynemann/provy.git
$ python setup.py install
$ provy --version
```

As can be seen above, after being installed a *provy* command becomes available.

provy is FOSS and you can find its source code at its github page.

CHAPTER 4

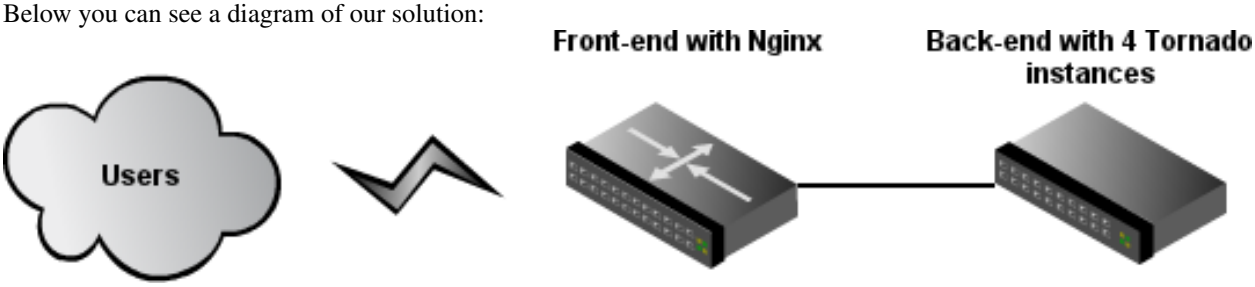
Getting started

Starting to provision servers with *provy* is extremely simple. We'll provision a fairly scalable infrastructure (yet simple) for a Tornado website.

Tornado is facebook's python web server. It features non-blocking I/O and is extremely fast.

4.1 The solution

Below you can see a diagram of our solution:



Our solution will feature a front-end server with one nginx instance doing the load-balancing among the tornado instances in our back-end server.

The back-end server will feature 4 tornado instances kept alive by supervisor (a process monitoring tool).

Create a file called *website.py* at some directory with this content:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import sys
import tornado.ioloop
import tornado.web

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("Hello, world")

application = tornado.web.Application([
    (r"/", MainHandler),
])

if __name__ == "__main__":
    port = int(sys.argv[1])
```

```
    application.listen(port, '0.0.0.0')
    print ">> Website running at http://0.0.0.0:%d" % port
    tornado.ioloop.IOLoop.instance().start()
```

Yes, it is not a very involved example, but *Hello World* suffices for our purposes. This python application takes a port as command-line argument and can be run with:

```
$ python website.py 8000
>> Website running at http://0.0.0.0:8000
```

4.2 The servers

Ok, now that we have a functioning application, let's deploy it to production.

First, let's create a local "production" environment using Vagrant. Using Vagrant is beyond the scope of this tutorial.

First make sure you have the *base* box installed. If you don't, use:

```
$ vagrant box add base http://files.vagrantup.com/lucid32.box
```

In the same directory that we created the website.py file, type:

```
$ vagrant init
      create  Vagrantfile
```

This will create a file called VagrantFile. This is the file that configures our Vagrant instances. Open it up in your editor of choice and change it to read:

```
Vagrant::Config.run do |config|

    config.vm.define :frontend do |inner_config|
        inner_config.vm.box = "base"
        inner_config.vm.forward_port(80, 8080)
        inner_config.vm.network(:hostonly, "33.33.33.33")
    end

    config.vm.define :backend do |inner_config|
        inner_config.vm.box = "base"
        inner_config.vm.forward_port(80, 8081)
        inner_config.vm.network(:hostonly, "33.33.33.34")
    end

end
```

Ok, now when we run vagrant we'll have two servers up: 33.33.33.33 and 33.33.33.34. The first one will be our front-end server and the latter our back-end server.

4.3 Provisioning file

It's now time to start provisioning our servers. In the same directory that we created the *website.py* file, let's create a file called *provyfile.py*.

The first thing we'll do in this file is importing the *provy* classes we'll use. We'll also define *FrontEnd* and *BackEnd* roles and assign them to our two vagrant servers.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from provy.core import Role

class FrontEnd(Role):
    def provision(self):
        pass

class BackEnd(Role):
    def provision(self):
        pass

servers = {
    'test': {
        'frontend': {
            'address': '33.33.33.33',
            'user': 'vagrant',
            'roles': [
                FrontEnd
            ]
        },
        'backend': {
            'address': '33.33.33.34',
            'user': 'vagrant',
            'roles': [
                BackEnd
            ]
        }
    }
}
```

Even though our script does not actually provision anything yet, let's stop to see some interesting points of it.

You can see that our roles (*FrontEnd* and *BackEnd*) both inherit from *provy.Role*. This is needed so that these roles can inherit a lot of functionality needed for interacting with our servers.

Another thing to notice is the *servers* dictionary. This is where we tell *provy* how to connect to each server and what roles does it have.

We can run this script (even if it won't do anything) with:

```
$ # will provision both servers
$ provy -s test

$ # will provision only the frontend server
$ provy -s test.frontend

$ # will provision only the backend server
$ provy -s test.backend
```

4.4 Provisioning the back-end server

Let's start working in our back-end server, since our front-end server depends on it to run.

First we'll make sure we are running our app under our own user and not root:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from provy.core import Role
from provy.more.debian import UserRole

class FrontEnd(Role):
    def provision(self):
        pass

class BackEnd(Role):
    def provision(self):
        with self.using(UserRole) as role:
            role.ensure_user('backend', identified_by='pass', is_admin=True)

servers = {
    'test': {
        'frontend': {
            'address': '33.33.33.33',
            'user': 'vagrant',
            'roles': [
                FrontEnd
            ]
        },
        'backend': {
            'address': '33.33.33.34',
            'user': 'vagrant',
            'roles': [
                BackEnd
            ]
        }
    }
}
```

Then we'll need to copy the *website.py* file to the server. *provy* can easily copy files to the servers, as long as it can find them. Just move the *website.py* file to a directory named *files* in the same directory as *provyfile.py*.

Now we can easily copy it to the */home/frontend* directory:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from provy.core import Role
from provy.more.debian import UserRole

class FrontEnd(Role):
    def provision(self):
        pass

class BackEnd(Role):
    def provision(self):
        with self.using(UserRole) as role:
            role.ensure_user('backend', identified_by='pass', is_admin=True)

        self.update_file('website.py', '/home/backend/website.py', owner='backend', sudo=True)

servers = {
    'test': {
        'frontend': {
```

```
            'address': '33.33.33.33',
            'user': 'vagrant',
            'roles': [
                FrontEnd
            ]
        },
        'backend': {
            'address': '33.33.33.34',
            'user': 'vagrant',
            'roles': [
                BackEnd
            ]
        }
    }
}
```

The *update_file* method tells *provy* to compare the source and target files and if they are different update the target file. For more information check the documentation.

Next we must make sure Tornado is installed. *provy* already comes with a role that does that:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from provy.core import Role
from provy.more.debian import UserRole, TornadoRole

class FrontEnd(Role):
    def provision(self):
        pass

class BackEnd(Role):
    def provision(self):
        with self.using(UserRole) as role:
            role.ensure_user('backend', identified_by='pass', is_admin=True)

        self.update_file('website.py', '/home/backend/website.py', owner='backend', sudo=True)

        self.provision_role(TornadoRole)

servers = {
    'test': {
        'frontend': {
            'address': '33.33.33.33',
            'user': 'vagrant',
            'roles': [
                FrontEnd
            ]
        },
        'backend': {
            'address': '33.33.33.34',
            'user': 'vagrant',
            'roles': [
                BackEnd
            ]
        }
    }
}
```

Now all we have to do is instruct supervisor to run four instances of our app:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from provoy.core import Role
from provoy.more.debian import UserRole, TornadoRole, SupervisorRole

class FrontEnd(Role):
    def provision(self):
        pass

class BackEnd(Role):
    def provision(self):
        with self.using(UserRole) as role:
            role.ensure_user('backend', identified_by='pass', is_admin=True)

        self.update_file('website.py', '/home/backend/website.py', owner='backend', sudo=True)

        self.provision_role(TornadoRole)

        # make sure we have a folder to store our logs
        self.ensure_dir('/home/backend/logs', owner='backend')

        with self.using(SupervisorRole) as role:
            role.config(
                config_file_directory='/home/backend',
                log_folder='/home/backend/logs/',
                user='backend'
            )

            with role.with_program('website') as program:
                program.directory = '/home/backend'
                program.command = 'python website.py 800%(process_num)s'
                program.number_of_processes = 4

                program.log_folder = '/home/backend/logs'

servers = {
    'test': {
        'frontend': {
            'address': '33.33.33.33',
            'user': 'vagrant',
            'roles': [
                FrontEnd
            ]
        },
        'backend': {
            'address': '33.33.33.34',
            'user': 'vagrant',
            'roles': [
                BackEnd
            ]
        }
    }
}
```

4.5 Provisioning the front-end server

Ok, now let's get our front-end up and running. *provy* comes with an nginx module, so it is pretty easy configuring it.

We have to provide template files for both *nginx.conf* and our website's site. Following what Tornado's documentation instructs, these are good templates:

```
user {{ user }};
worker_processes 1;

error_log /home/frontend/error.log;
pid /home/frontend/nginx.pid;

events {
    worker_connections 1024;
    use epoll;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    access_log /home/frontend/nginx.access.log;

    keepalive_timeout 65;
    proxy_read_timeout 200;
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    gzip on;
    gzip_min_length 1000;
    gzip_proxied any;
    gzip_types text/plain text/css text/xml
               application/x-javascript application/xml
               application/atom+xml text/javascript;

    proxy_next_upstream error;

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

```
upstream frontends {
    server 33.33.33.34:8000;
    server 33.33.33.34:8001;
    server 33.33.33.34:8002;
    server 33.33.33.34:8003;
}

server {
    listen 8888;
    server_name  localhost 33.33.33.33;

    access_log  /home/frontend/website.access.log;

    location / {
        proxy_pass_header Server;
        proxy_set_header Host $http_host;
```

```
        proxy_redirect off;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Scheme $scheme;
        proxy_pass http://frontends;
    }
}
```

Save them as *files/nginx.conf* and *files/website*, respectively.

Now all that's left is making sure that *provy* configures our front-end server:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from provy.core import Role
from provy.more.debian import UserRole, TornadoRole, SupervisorRole, NginxRole

class FrontEnd(Role):
    def provision(self):
        with self.using(UserRole) as role:
            role.ensure_user('frontend', identified_by='pass', is_admin=True)

        with self.using(NginxRole) as role:
            role.ensure_conf(conf_template='nginx.conf', options={'user': 'frontend'})
            role.ensure_site_disabled('default')
            role.create_site(site='website', template='website')
            role.ensure_site_enabled('website')

class BackEnd(Role):
    def provision(self):
        with self.using(UserRole) as role:
            role.ensure_user('backend', identified_by='pass', is_admin=True)

        self.update_file('website.py', '/home/backend/website.py', owner='backend', sudo=True)

        self.provision_role(TornadoRole)

        # make sure we have a folder to store our logs
        self.ensure_dir('/home/backend/logs', owner='backend')

        with self.using(SupervisorRole) as role:
            role.config(
                config_file_directory='/home/backend',
                log_folder='/home/backend/logs/',
                user='backend'
            )

            with role.with_program('website') as program:
                program.directory = '/home/backend'
                program.command = 'python website.py 800%(process_num)s'
                program.number_of_processes = 4

                program.log_folder = '/home/backend/logs'

servers = {
    'test': {
        'frontend': {
            'address': '33.33.33.33',
            'user': 'vagrant',
```

```
            'roles': [
                FrontEnd
            ]
        },
        'backend': {
            'address': '33.33.33.34',
            'user': 'vagrant',
            'roles': [
                BackEnd
            ]
        }
    }
}
```

See how we passed the user name as an option to the *nginx.conf* template? *provy* allows this kind of template interaction in many places. For more information, check the documentation.

4.6 Running and verifying it works

We can now fire our brand new infrastructure and check that the website is working:

```
$ vagrant up
$ provy -s test
$ curl http://33.33.33.33
```

After these 3 commands finished running (it might take a long time depending on your connection speed), you should see *Hello World* as the result of the curl command.

CHAPTER 5

provyfile and Runtime Arguments

provy uses a python module called *provyfile.py* in order to retrieve the definitions of your roles, servers and the relationships between them.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from provy.core import Role, AskFor
from provy.more.debian import NginxRole, TornadoRole, UserRole, SSHRole
from provy.more.debian import PipRole, VarnishRole, AptitudeRole, GitRole
from provy.more.debian import SupervisorRole
```

Role imports

```
class FrontEnd(Role):
    def provision(self):
        user = self.context['front-end-user']
        with self.using(UserRole) as role:
            role.ensure_user(user, identified_by='pass', is_admin=True)

        with self.using(VarnishRole) as role:
            role.ensure_vcl('default.vcl', owner=user)
            role.ensure_conf('default_varnish', owner=user)

        with self.using(NginxRole) as role:
            role.ensure_conf(conf_template='test-conf.conf')
            role.ensure_site_disabled('default')
            role.create_site(site='frontend', template='test-site')
            role.ensure_site_enabled('frontend')
```

Role Definitions

```
servers = {
    'test': {
        'frontend': {
            'address': '33.33.33.33',
            'user': 'vagrant',
            'roles': [
                FrontEnd
            ],
            'options': {
                'front-end-user': AskFor('front-end-user', 'Please enter the name of the nginx user')
            }
        },
}
```

Server Definitions

This is the overall structure of a *provy* provisioning file. The first imports will be very similar among most your provyfile.py files. You'll always import Role and most of the time will use one of the other built-in roles.

After the imports, come your *Role* Definitions. This is where you'll specify how you want your servers to be built. You can find more about how to build roles in the What are roles? and Using Roles in my Roles sections.

Last but not least, you describe your servers and how they relate to roles. This brings us to the *AskFor* parameter (*provy.core.AskFor*). This class allows you to specify that a given option for a given server should be filled at runtime, either by passing in the command line or by asking the user doing the provisioning.

AskFor takes two arguments: *key* and *question*. The *key* argument is needed to allow passing the argument when running *provy* (more on that in the next section). The *question* is used when *provy* asks the user running it for the parameter.

AskFor is really useful for sensitive data such as passwords. You don't want to expose this data in your provyfile in plain text. You just use an *AskFor* parameter for it and supply the information at runtime. Let's look at a sample of *AskFor* usage.

```
servers = {
    'frontend': {
        'address': '33.33.33.33',
        'user': 'vagrant',
        'roles': [
            FrontEnd
        ],
        'options': {
            'mysql-db-password':
                        AskFor('mysql-db-password',
                               'Please enter the password for the app database')
        }
    }
}
```

This parameter can be supplied twofold: if you don't specify it in the console when calling *provy*, you will be asked for it. If you need to specify it in the console, just use its key like this:

```
provy -s server -p password mysql-db-password=somepass
```

All arguments must take this form of key=value, with no spaces. The key must be exactly the same, case-sensitive.

CHAPTER 6

Running provy

provy comes with a console runner. It's the same one we used in the Getting started tutorial.

The console runner supports some options. For more information you can use the –help command. You should see output like the following:

```
$ provy --help
Usage: console.py [options]

Options:
  -h, --help            show this help message and exit
  -s SERVER, --server=SERVER
                        Servers to provision with the specified role. This is
                        a recursive     option.
  -p PASSWORD, --password=PASSWORD
                        Password to use for authentication with servers.
                        If passwords differ from server to server this does
                        not work.
```

The option you are most likely to use is the *server* option. It tells *provy* what servers you want provisioned.

As we saw in the provyfile and Runtime Arguments section, we can also supply *AskFor* arguments when running *provy*.

All arguments must take the form of key=value, with no spaces. The key must be exactly the same as the one in the *AskFor* definition, case-sensitive.

CHAPTER 7

What are roles?

Roles are the most important concept in *provy*. A role specifies one server capability, like user management or a package provider.

provy comes with many bundled roles, but you can very easily create your own roles. As a matter of fact, if you followed the Getting started tutorial, you have already created two custom roles.

Creating new roles is as easy as creating a class that inherits from *provy.Role* and implements a *provision* method.

This method is the one *provy* will call when this role is being provisioned into a given server.

CHAPTER 8

Using Roles in my Roles

As you may have noticed, *provy* provides a special syntax for using other roles in your role. Say we need to use the *AptitudeRole* in our role. This is how we'd do it:

```
class MyRole(Role):
    def provision(self):
        with self.using(AptitudeRole) as role:
            # do something with role
            role.ensure_package_installed('some-package')
```

The *using* method of the Role class is a special way of using other roles. The reason for using it is that it maintains context and the *provy* lifecycle (more on both later).

If you just want to provision another role in your role, you can use:

```
class MyRole(Role):
    def provision(self):
        self.provision_role(TornadoRole)
```

The *provision_role* method does exactly the same as the *using* method, except it does not enter a with block. This should be used when you don't wnat to call anything in the role, instead just have it provisioned.

CHAPTER 9

Custom Files and Templating

Some methods provided by provy (including *update_file*) support passing in options that may be used in templates.

provy uses jinja2 for templating, thus supporting if/else statements, loops and much more. It's advised that you take a look at jinja2 docs.

jinja2 will look for files in two different places. The first one and probably the one you'll use the most, is a directory called *files* in the same path as *provyfile.py*.

Any files you place inside this directory may be used as templates to be uploaded to the server being provisioned. Since *provy* is built on top of fabric, you can use its *put* method as well to put any file or folder to the server. It's advised to use the bundled methods that come with provy, though, as those are idempotent.

The other place you can put files is in a *templates* directory inside Role apps. The supervisor role uses this approach, if you want to take a look at an example. If you do place files in the *templates* directory, do not forget to call the *register_template_loader* method passing in the full namespace of your app (more details in the provy.more section below).

We used custom files in the Getting started section to provide the needed configuration files for nginx.

CHAPTER 10

Roles documentation

provy is basically divided in two namespaces: `provy.core` and `provy.more`. It is divided like this to make it easier to find the roles you need.

`provy.core` features the code that actually makes *provy* run, like the console app or the base Role.

`provy.more` features every single specialized role that helps users in provisioning servers, like *NginxRole* or *AptitudeRole*.

If you're looking for all the modules tree, `click here`

CHAPTER 11

Supported Operating Systems

At this point, *provy* only supports Debian-based (Debian, Ubuntu etc) and CentOS distros. If you think your distribution should be here, please consider contributing.

This area details what features of *provy* are supported in each operating system.

11.1 Debian distributions and Ubuntu

Currently all features are supported under Debian-based distributions (including Ubuntu).

The easiest way to verify what's available is checking the Roles documentation section.

11.2 CentOS distributions

Currently support to user management, git repository management and packaging (via pip and yum) are supported.

More supported features to come soon. If you think you can help improve this, please consider contributing.

CHAPTER 12

provy Recipes

12.1 Django + Nginx same server

This recipe features a django website with 4 processes and 2 threads per process.

Nginx serves the requests via reverse proxy, while load balancing the 4 processes.

Each django process is a gunicorn process bound to a port ranging from 8000-8003.

Django's static files are served using nginx.

To read more about this recipe, `click here`.

CHAPTER 13

Who’s using provy?

CHAPTER 14

Ideas, features or suggestions

If you have any ideas that can improve *provy*, or feature requests or even just some suggestion on the library or on this website, please let us know! We value your opinion more than anything else. If you don't have the time to contribute to the project, contribute with your ideas.

We have setup an easy-to-use way of providing feedback with the help of the nice people at uservoice.com.

You can go to our uservoice.com page at http://provy.uservoice.com and vote in the ideas given by users, thus helping us select ideas that will be implemented first.

Or you can just click on the button in the right-bottom of the screen that says *feedback & support* and tell us any number of things you think will help the project.

In the event that you not only want to send us your idea, but want to implement it and contribute to the project, keep reading the Contributing section.

CHAPTER 15

Contributing

Contributions are very welcome. Specially roles. If you implement a role that you think others might be using, please contribute.

To contribute head to provy's github page, fork it and create a pull request.

15.1 Developing

15.1.1 Make it great :-)

We strive to keep the internal quality of provy to the best that we can; Therefore, it's very important to keep some things in mind when contributing with code for provy:

- Test everything you can, with automated tests. If possible, please develop code with TDD. If you're having a hard time building tests, don't hesitate to ask for help in the provy mailing list. We are happy to help you keep your code well-covered by tests;
- When writing actual code, follow the conventions in PEP 8 (except for maximum line length, which we don't follow because there are too many parts of the project that require large strings to be used);
- When writing docstrings, follow the conventions in PEP 257 (take a look at other docstrings in the project to get a feel of how we organize them);
  - Also, when writing docstrings for the API, provide examples of how that method or class works. Having a code example of a part of the API is really helpful for the user.

15.1.2 Setting up your environment

1. Make sure pip, virtualenv and virtualenvwrapper are installed;
2. Create a virtual environment for provy:

```
$ mkvirtualenv provy
```

3. Install the requirements:

```
$ pip install -r REQUIREMENTS
```

4. Run your first provy build, to make sure everything's ready for you to start developing:

```
$ make build
```

The command should run without accusing any error.

15.1.3 How to develop

There are basically two commands we run, when developing.

When building code, you need to test it and check if the code format is OK with the conventions we use:

```
$ make build
```

This Makefile target essentially does these steps:

1. It runs the tests over the project;
2. It builds a code coverage report (you should take a look if the total code coverage is not decreasing, when you build your code);
3. It runs flake8 over the entire codebase, making sure the code style is following the conventions mentioned above.

It's also important to keep the codebase well documented. We use Sphinx to generate the documentation, which is also used when our docs go to Read The Docs.

To build the docs in your environment, in order to test it locally (this is very useful to see how your docs will look like when they are rolled out), first go to the *provy/docs* directory, then run:

```
$ make html
```

Some warnings may show up in the command output - you should listen to them, in order to spot possible documentation problems -.

15.2 The team

15.2.1 The core team

The core team behind provy (in order of joining the project):

- Bernardo Heynemann (technical leader of this project)
- Rafael Carício
- Douglas Andrade
- Thiago Avelino
- Diogo Baeder

15.2.2 Other contributors

Other non-core members, but equally important, equally rocking, equally ass-kicking contributors can be seen in this list: https://github.com/python-provy/provy/network/members

There are also some more contributors that haven't send code to the project, but who help in other ways, when and how they can. We're very happy to have you, guys! :-)

CHAPTER 16

About

provy is an easy-to-use provisioning system in python.

Turn that tedious task of provisioning the infrastructure of your website into a repeatable no-frills reliable process.

CHAPTER 17

Documentation

(Looking for the API? Here's a shortcut!)

CHAPTER 18

Contacts

The place to create issues is provy's github issues. The more information you send about an issue, the greater the chance it will get fixed fast.

If you are not sure about something, have a doubt or feedback, or just want to ask for a feature, feel free to join our mailing list, or, if you're on FreeNode (IRC), you can join the chat #provy .

CHAPTER 19

License

provy is licensed under the MIT License

CHAPTER 20

Indices and tables

- genindex
- modindex
- search