Python Infrastructure

Release ∞

Oct 09, 2017

Contents

CHAPTER 1

PSF Infrastructure Overview

The PSF runs a wide variety of infrastructure services to support its mission from the PyCon site to the CPython Mercurial server. The goal of this page is to enumerate all these services, where they run, and who the main contact points are.

The Infrastructure Team

The infrastructure team is ultimately responsible maintaining PSF infrastructure. It is not, however, required to be a member of the infrastructure to run a PSF service. Indeed, the day to day operations of most services are handled by people not on the infrastructure team. The infrastructure team can assist in setting up new services and advise maintainers of individual services. Its members also generally handle changes to sensitive global systems such as DNS. The currrent team members are:

- Noah Kantrowitz (lead)
- Ernest W. Durbin III
- Benjamin W. Smith
- Donald Stufft
- Benjamin Peterson
- Mark Mangoba (PSF IT Manager)
- Alex Gaynor (has no responsibilities)

The best way to contact the infrastructure team is mailing infrastructure-staff@python.org. There's also often people hanging out on the #python-infra channel of Freenode.

Infrastructure Providers

The PSF uses several different cloud providers and services for its infrastructure.

XS4ALL XS4ALL is the by far the oldest Python infrastructure provider. There are two physical servers owned by the PSF at XS4ALL: albatross and dinsdale. (There also used to be one called ximinez, but it seems to be unreachable now.) albatross is the mail server. dinsdale hosts a number of legacy services. We try not to put anything new on the XS4ALL servers, preferring modern cloud providers.

OSUOSL Oregon State University Open Source Lab hosts VMs for the PSF. These VMs are provisioned using Chef and their configuration management is in the psf-chef git repo.

Rackspace Rackspace is the newest cloud provider utilized by the PSF. It hosts PyPI and an increasing number of python.org-related services (Python downloads, docs.python.org). Salt is used for configuration management.

Dyn & Gandi Gandi is our domain registar, and we use Dyn for DNS hosting on most of our domains.

Pingdom Pingdom provides monitoring and complains to us when services are down.

Fastly Fastly generously donates its content distribution network (CDN) to the PSF. Our highest traffic services (i.e. PyPI, www.python.org, docs.python.org) use this CDN to improve end-user latency.

Datacenters

PSF DC	Provider	Region
iad1	Rackspace	IAD
ord1	Rackspace	ORD
ams1	Digital Ocean	AMS3
nyc1	Digital Ocean	NYC3

Details of Various Services

This section enumerates PSF services, generalities about their hosting, and contact information for the owners.

Buildbot The buildbot master is a service run by python-dev@python.org, particularly Antoine Pitrou and Zach Ware.

bugs.python.org bugs.python.org is hosted using a server donated by Upfront Systems. The tracker-discuss@python.org list is used for discussion of the tracker.

docs.python.org The Python documentation is hosted on a Rackspace VM, served through Fastly, and owned by Benjamin Peterson and Georg Brandl.

hg.python.org The CPython Mercurial repositories are hosted on a Rackspace VM. The service is owned by Antoine Pitrou and Georg Brandl.

mail.python.org The python.org Mailman instance is hosted on https://mail.python.org as well as SMTP (Postfix). All inquiries should be directed to postmaster@python.org.

planetpython.org and planet.jython.org These are hosted on a Rackspace VM. The Planet code and configuration are hosted on GitHub and maintained by the team at planet@python.org.

pythontest.net pythontest.net hosts services and files used by the Python test suite. python-dev@python.org is ultimately responsible for its maintenance.

speed.python.org speed.python.org is a Codespeed instance tracking Python performance. The web interface is hosted on a Rackspace VM, benchmarks are run on a beefy machine at OSUOSL and scheduled by the Buildbot master. Maintained by speed@python.org and Zach Ware.

wiki.python.org This is hosted on an OSUOSL VM. Marc-André Lemburg owns it.

www.jython.org This is hosted on a Rackspace VM. The setup is quite simple and shouldn't require much tweaking, but Benjamin Peterson can be poked about it.

www.python.org The main Python website is a Django app hosted on a Rackspace VM. Its source code is available on GitHub, and issues with the site can be reported to the GitHub issue tracker. Python downloads (i.e. everything under https://www.python.org/ftp/) are hosted on a separate Rackspace VM. The whole site is behind Fastly. There is also https://staging.python.org for testing the site. http://legacy.python.org is the old website hosted on dinsdale.

PyCon The PyCon website is hosted on Rackspace VMs. The contact address is pycon-tech@python.org.

PyPI The Python Package Index sees the most load of any PSF service. All of its infrastructure runs on Rackspace configured by pypi-salt, and it is served over Fastly. The infrastructure is maintained by Ernest W. Durbin, Donald Stufft, and Richard Jones.

PyPy properties The PyPy website is hosted on a OSUOSL VM and maintained by pypy-dev@python.org.

CHAPTER 2

Server Guide

Bootstrap a Server

Unless otherwise required all machines operated by the PSF Salt infrastructure should be running Ubuntu 14.04 and they will have their configuration managed by psf-salt. Each machine should be given a hostname which matches the pattern `serviceN.dc.psf.io` where `serviceN` is replaced by a service name (such as `pg`) and a unique number, and `dc` is replaced by the PSF DC identifier for the DC that this machine is in. A full example would be `pg0.iad1.psf.io`. You'll need to add this hostname to `pillar/prod/roles.sls` and `pillar/dev/roles.sls` to put the machine in the correct configuration nodegroup.

Once you have a machine, you can bootstrap it by simply executing `inv salt.bootstrap <public address>`. This will SSH into the machine, install all the required software, register it with the salt master and run highstate on it. Within 15 minutes the salt master will also setup the DNS for the machine and it will live at the hostname that you have given it at the psf.io domain.

CHAPTER 3

APT Packages

PSF Infrastructure has the ability to create apt repositories stored on S3. The primary one of these is named "psf" and it is located at `http://apt.psf.io/psf/`. The psf repository has been added to all servers by default and can be used to ship things which are not available in Ubuntu or a ppa or for which there are patched versions required.

Install from the PSF repository

Generally nothing different needs to happen to install from the PSF repository, just a simple `apt-get install <something>` or adding it to a salt state should pick up the package automatically.

Uploading to the PSF repository

In order to upload to the PSF repository you must be a member of the `aptly-uploaders` group. You can add yourself to this group by simply editing `pillar/users.sls` and adding:

```
access:
  packages:
    groups:
      - aptly-uploaders
```

below your user account.

After you're a member of that `aptly-uploaders` group, you can upload a `.deb` or a `.dsc` file by simplying placing it in the `/srv/aptly/incoming/psf-trusty/` or `/srv/aptly/incoming/psf-precise/` directory on the packages.psf.io server depending on if the package is for trusty or precise. For example:

```
$ scp python-wal-e_0.7.2-2_all.deb packages.psf.io:/srv/aptly/incoming/psf-trusty/
```

Every 5 minutes `packages.psf.io` will scan this directory for new files, process them and then they will be available in the PSF repository.

CHAPTER 4

Security

TLS Cipher Suites

Any place where TLS is being used and a cipher string can be specified a cipher string from the `tls.ciphers` pillar should be used. Ideally this will be used in a way like:

```
{{ pillar["tls"]["ciphers"].get("a unique name", pillar["tls"]["ciphers"]["default"])␣
↪}}
```

This will ensure that the ciphers can all be configured in one place, that by default the same cipher strings are used everywhere, but still allow overriding the cipher strings for each service where it makes sense.

Service Authenticity

In order to validate that a particular server is allowed to function as a particular service the infrastructure makes use of TLS certificates signed by a custom certificate authority.

A new service can be given a certificate by editing the `pillar/base/tls.sls` file and adding a new section under the `gen_certs` key. This should look something like:

```
tls:
  gen_certs:
    postgresql.psf.io:
      days: 7
      roles:
        - postgresql
```

Where `days` is how many days a particular certificate should be valid for, and `roles` is a list of roles which need access to this certificate. It is important that the `days` argument be kept short so that a compromised key is only valid for a small window. The system will ensure that it replaces any soon to expire certificates with new certificates before they expire.

This certificate will then be available on the servers at `/etc/ssl/private/{{ name }}.pem`. That file contains both the certificate itself and the private key for the certificate. It can be validated against the `/etc/ssl/certs/PSF_CA.pem` file which is available on all servers as well.

This requires configuration on the master like:

```
extension_modules: srv/salt/_extensions

ext_pillar:
  - ca:
      name: PSF_CA
      cert_opts:
        C: US
        ST: NH
        L: Wolfeboro
        O: Python Software Foundation
        OU: Infrastructure Team
        emailAddress: infrastructure@python.org
```

CHAPTER 5

Server List

Name	Purpose	Contact	Datacenter	Region
consul0.iad1.psf.io	Runs https://www.consul.io/service discovery	dstufft	Rackspace	Northern Virginia (IA
consul1.iad1.psf.io	Runs https://www.consul.io/service discovery	dstufft	Rackspace	Northern Virginia (IA
consul2.iad1.psf.io	Runs https://www.consul.io/service discovery	dstufft	Rackspace	Northern Virginia (IA
docs.iad1.psf.io	Builds and serves CPython's documentation	benjamin	Rackspace	Northern Virginia (IA
lb0.iad1.psf.io	Load Balancer	dstufft	Rackspace	Northern Virginia (IA
lb1.iad1.psf.io	Load Balancer	dstufft	Rackspace	Northern Virginia (IA
lb2.iad1.psf.io	Load Balancer	dstufft	Rackspace	Northern Virginia (IA
mail.python.org	Mail and Mailman Server	postmasters	Digital Ocean	Amsterdam (AMS3)
monitoring.iad1.psf.io	Monitoring	dstufft	Rackspace	Northern Virginia (IA
monitoring.pypi.io	Monitoring.pypi.io	dstufft	Rackspace	Northern Virginia (IA
pg0.iad1.psf.io	postgresql cluster	dstufft	Rackspace	Northern Virginia (IA
pg1.iad1.psf.io	postgresql cluster	dstufft	Rackspace	Northern Virginia (IA
planet.iad1.psf.io	Planet Python.	benjamin	Rackspace	Northern Virginia (IA
pydotorg-prod.iad1.psf.io	Python.org production	benjamin	Rackspace	Northern Virginia (IA
pydotorg-staging.iad1.psf.io	Python.org staging	benjamin	Rackspace	Northern Virginia (IA
pythontest.nyc1.psf.io	Test resources for CPython's test suite.	benjamin	Digital Ocean	New York City (NYC
slack.iad1.psf.io	Slack IRC bridge	dstufft	Rackspace	Northern Virginia (IA
speed-web.iad1.psf.io	web interface for speed.python.org	dstufft	Rackspace	Northern Virginia (IA
web0.pypa.iad1.psf.io	Python Packaging Authority	dstufft	Rackspace	Northern Virginia (IA
web0.pypi.io	Holds bootstrap.pypa.io	dstufft	Rackspace	Northern Virginia (IA
web1.pypi.io	Backend servers for PyPI legacy	dstufft	Rackspace	Northern Virginia (IA
evote.python.org	PSF Voting Platform	mmangoba	Rackspace	Chicago (ORD)
virt-h669vt.psf.osuosl.org	Loadbalancer OSUOSL	noah	OSUOSL	OSUOSL
speed-python.osuosl.org	Python Speed	noah	OSUOSL	OSUOSL
virt-wzmlmm.psf.osuosl.org	Advocacy	noah	OSUOSL	OSUOSL
virt-gwhg4e.psf.osuosl.org	Coverity	noah	OSUOSL	OSUOSL
virt-ys0nco.psf.osuosl.org	Wiki	marc-andre	OSUOSL	OSUOSL
virt-et2yi0.psf.osuosl.org	Buildmaster	noah	OSUOSL	OSUOSL
virt-wdiwcy.psf.osuosl.org	PyPy Codespeed	noah	OSUOSL	OSUOSL

Continued on next pa

Table 5.1 – continued from previous page

Name	Purpose	Contact	Datacenter	Region
virt-sxw5uy.psf.osuosl.org	Load Balancer OSUOSL	noah	OSUOSL	OSUOSL
virt-k4b2sa.psf.osuosl.org	rsnapshot	noah	OSUOSL	OSUOSL
virt-8joqck.psf.osuosl.org	RPI	noah	OSUOSL	OSUOSL
virt-7tac5q.psf.osuosl.org	PyPy Home	noah	OSUOSL	OSUOSL
virt-l99amx.psf.osuosl.org	Monitoring OSUOSL	noah	OSUOSL	OSUOSL

CHAPTER 6

SSL List

Date Issued	Domain/Purpose	Certificate Authority	Key	CSR by and C
2/13/17	pythonhosted.org	Gandi	4096 bit	Mark Mangoba
2/10/17	pypy.org	Gandi	4096 bit	Mark Mangoba
2/12/17	bugs.python.org	Gandi	4096 bit	Mark Mangoba
2/3/17	.python.org	Gandi	4096 bit	Mark Mangoba
10/15/16	mail.python.org	LetsEncrypt	4096 bit	Mark Sapiro
9/14/16	bugs.python.org	StartSSL	4096 bit	Mark Mangoba
7/8/16	.psfmember.org	StartSSL	4096 bit	Kurt B. Kaiser
7/3/16	upload.pypi.org	StartSSL	4096 bit	Donald Stufft
7/3/16	upload.pypi.io	StartSSL	4096 bit	Donald Stufft
5/2/16	.pypi.io	StartSSL	2048 bit	Donald Stufft
2/6/16	Code signing certificate, Python Windows installers	StartSSL	4096 bit	Marc-Andre Le
11/24/15	www.pycon.org	StartSSL	4096 bit	Marc-Andre Le
9/10/15	.pl.pycon.org	StartSSL	4096 bit	Dariusz Grzesis
9/8/15	.cz.pycon.org	StartSSL	4096 bit	Tom Ehrlich
6/1/15	.za.pycon.org	StartSSL	4096 bit	Simon Cross
5/21/15	kiwi.pycon.org and nz.pycon.org	StartSSL	4096 bit	Danny Adair
4/13/15	pl.python.org	StartSSL	4096 bit	Piotr Tynecki
2/12/15	.pythonhosted.org	StartSSL	2048 bit	Donald Stufft
2/3/15	tw.pycon.org	StartSSL	4096 bit	Yung-Yu Chen
2/2/15	.python.org and us.pycon.org	StartSSL	2048 bit	Donald Stufft
1/6/15	.pypa.io	StartSSL	2048 bit	Donald Stufft
12/19/14	mail.python.org	StartSSL	4096 bit	Patrick Ben Koe
11/5/14	status.python.org	Not issued by PSF	Not issued by PSF	Not issued by P
9/23/14	bugs.python.org	Gandi	2048 bit	David Murray
9/5/14	www.python.org	DigiCert	EV certificate	Fastly CDN Dig
9/5/14	python.org	DigiCert	EV certificate	Fastly CDN Dig
9/5/14	pypi.python.org	DigiCert	EV certificate	Fastly CDN Dig
9/5/14	docs.python.org	DigiCert	EV certificate	Fastly CDN Dig
9/5/14	testpypi.python.org	DigiCert	EV certificate	Fastly CDN Dig

Table 6.1 – continued from previous page

Date Issued	Domain/Purpose	Certificate Authority	Key	CSR by and C
9/5/14	bugs.python.org	DigiCert	EV certificate	Fastly CDN Dig
9/5/14	wiki.python.org	DigiCert	EV certificate	Fastly CDN Dig
9/5/14	hg.python.org	DigiCert	EV certificate	Fastly CDN Dig
9/5/14	mail.python.org	DigiCert	EV certificate	Fastly CDN Dig
9/5/14	packaging.python.org	DigiCert	EV certificate	Fastly CDN Dig
9/5/14	pythonhosted.org	DigiCert	EV certificate	Fastly CDN Dig
9/5/14	pythonhosted.org	DigiCert	EV certificate	Fastly CDN Dig
9/5/14	test.pythonhosted.org	DigiCert	EV certificate	Fastly CDN Dig
9/5/14	us.pycon.org	DigiCert	EV certificate	Fastly CDN Dig
9/5/14	id.python.org	DigiCert	EV certificate	Fastly CDN Dig
6/30/14	za.pycon.org	Not issued by PSF	Not issued by PSF	Not issued by P
2/11/14	vote.python.org	Gandi	PSF TA account	PSF TA account
7/29/13	za.pycon.org	Not issued by PSF	Not issued by PSF	Not issued by P

CHAPTER 7

Services

Discovery

The Python Infrastructure uses Consul to implement service discovery. There is an agent running in client mode on all servers in the iad1 datacenter. These can be queried on the standard ports on localhost. Consul also has a web interface running on localhost on all nodes.

Using the Web Interface

To use the Consul web interface simply ssh into any server with a SSH port forward such as `ssh salt-master.psf.io -L 8500:localhost:8500` and then navigate to http://localhost:8500/ in your browser.

Using a Service

Generally the best way to handle pointing to a service is to utilize the `consul-template` utility which will automatically watch a service and update a configuration file and reload/restart a process whenever it is updated. This takes a template (which can be rendered by salt) and will detect which services it needs based on that.

This is best demonstrated by an example, say haproxy:

```
haproxy:
  service.running:
    - enable: True
    - reload: True
    - require:
      - service: haproxy-consul

/etc/haproxy/haproxy.cfg.tmpl:
  file.managed:
    - source: salt://haproxy/config/haproxy.cfg.jinja
    - template: jinja
```

```
haproxy-consul:
  file.managed:
    - name: /etc/init/haproxy-consul.conf
    - source: salt://consul/init/consul-template.conf.jinja
    - template: jinja
    - context:
        templates:
          - "/etc/haproxy/haproxy.cfg.tmpl:/etc/haproxy/haproxy.cfg:chmod 644 /etc/
↪haproxy/haproxy.cfg && service haproxy reload"
    - require:
      - pkg: consul

  service.running:
    - enable: True
    - restart: True
    - require:
      - pkg: consul
    - watch:
      - file: haproxy-consul
      - file: /etc/consul-template.conf
      - file: /etc/haproxy/haproxy.cfg.tmpl
```

Essentially this sets up a `haproxy` proxy service which requires the `haproxy-consul` service. Then it renders a configuration template and finally it creates a `haproxy-consul` instance which has an upstart config and is set to running. The `templates` variable passed into the contents is important. It is a list of template, destiniation, command tuples. This is in the form of `"/path/to/template:/path/to/destination:shell command to run"`.

The template file can contain blocks that look like:

```
{{range service "my-service@iad1"}}
server {{.Name}} {{.Address}}:{{.Port}} check{{end}}
```

This will render a server line for every entry in the `"my-service"` service in the iad1 datacenter and tt will use the name, address, and port of the registered service. For a full list of everything you can do in this template file take a look at the consul-template documentation.

Registering a Service

Registering an internal service (e.g. one that is running on a server in the iad1 datacenter) is quite easy. You simply need to pick a name, port, and any tags that you wish to associate with the service. Then simply add the below block to the state files:

```
/etc/consul.d/service-my-service.json:
  file.managed:
    - source: salt://consul/etc/service.jinja
    - template: jinja
    - context:
        name: my-service
        port: 9000
        tags:
          - tag1
          - tag2
    - user: root
    - group: root
    - mode: 644
```

```
    - require:
      - pkg: consul
```

Where the `name`, `port`, and `tags` context variables control the values that will be entered into the system. This will be available the next time that salt runs the highstate command. It is likely you'll want this state to require whatever states setup the service that you're exposing as any watchers will start using it near instantly.

Registering an External Service

Not all services are hosted internally, some services are external services where we cannot install a consul client on their servers. The Consul service system can handle this quite easily as well. To add an external service simply edit `pillar/dev/consul.sls` or `pillar/prod/consul.sls` and add a new entry in the `external` dictionary. The keys are `datacenter`, `node`, `address`, `service`, `port`. Using an external service is exactly like using an internal service.

Example:

```
consul:
  external:
    - datacenter: vagrant
      node: pythonanywhere
      address: www.pythonanywhere.com
      service: console
      port: 443
```

CDN (Powered by Fastly)

The PSF has a CDN donated by Fastly which is available to any service hosted on our infrastructure. This CDN is running a geo distributed varnish cluster which respects various headers for handling caching. HTTPS is available from fastly for any `*.python.org` domain including the EV Certificate.

The setup of our infrastructure means that the only configuration which needs to change to move a service between fronted by the CDN or not is what address the service CNAME points to. Everything else is identically configured.

Setup a Service with Fastly

Services served through Fastly are still hosted through the loadbalancers and should be configured as normal there. Note that if you're using nginx you can include the `fastly_params` file inside the server block to set up some common settings for fastly backends (`include fastly_params;`). In order to configure these services in Fastly you should do:

1. Create a new Service at Fastly.
2. Configure the backends in Fastly to match all of the loadbalancers (currently `lb0.psf.io` and `lb1.psf.io`) on port `20004`. These should also have TLS setup with the hostname `lb.psf.io`, and the PSF_CA certificate (see below). The SSL Client Certificate and Key should be blank.
3. Configure a Header under the "Content" tab to set a `Fastly-Token` header set to the static value from `pillar/prod/secrets/fastly.sls`.

PSF_CA

The current value of the PSF CA is:

```
-----BEGIN CERTIFICATE-----
MIIFIjCCBAqgAwIBAgIVAJRtfWNZoFYyqBv7Ol8/voe3sH/gMA0GCSqGSIb3DQEB
CwUAMIGsMQswCQYDVQQGEwJVUzELMAkGA1UECBMCTkgxEjAQBgNVBAcTCVdvbGZl
Ym9ybzEjMCEGA1UEChMaUHl0aG9uIFNvZnR3YXJlIEZvdW5kYXRpb24xHDAaBgNV
BAsTE0luZnJhc3RydWN0dXJlIFRlYW0xDzANBgNVBAMUBlBTRl9DQTEoMCYGCSqG
SIb3DQEJARYZaW5mcmFzdHJ1Y3R1cmVAcHl0aG9uLm9yZzAeFw0xNDEwMTkxOTUx
MTlaFw0xOTEwMTgxOTUxMTlaMIGsMQswCQYDVQQGEwJVUzELMAkGA1UECBMCTkgx
EjAQBgNVBAcTCVdvbGZlYm9ybzEjMCEGA1UEChMaUHl0aG9uIFNvZnR3YXJlIEZv
dW5kYXRpb24xHDAaBgNVBAsTE0luZnJhc3RydWN0dXJlIFRlYW0xDzANBgNVBAMU
BlBTRl9DQTEoMCYGCSqGSIb3DQEJARYZaW5mcmFzdHJ1Y3R1cmVAcHl0aG9uLm9y
ZzCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALPFdrBHn1e3j4HiWKLr
VKOD07y1Vv+qhJbIEhcGS71aqjBvrdV5enKTdTMp+/66l/hZs272/w8ozNMy7fTk
zmQ9CTw+JALYuRkuY2IKDx1TYviu/r9ssLwUdCknIMD5iSlrcBzW1XXB/FnoN0Qd
lrxjYLUbG5b+xVULG+gCg26yMk5iJoAqwRAK3iiBdRzZBP+lOspS2bC6H9srsmJb
26bOpofOBZa3Mt+xDtspA6w2QOMMPPPxbUV4VvSdazpNjAkYEsR0FKivWTHUDe9m
wATEXxDX4XZoNM0L2vc7DkCS0MgyJy3gHT5sVj2+hWUEYH67WSu3TvotWMMzckrC
c68CAwEAAaOCATcwggEzMBIGA1UdEwEB/wQIMAYBAf8CAQAwDgYDVR0PAQH/BAQD
AgEGMB0GA1UdDgQWBBRvXo8qm3WXAuvKe5Ig5AuE28obNjCB7QYDVR0jBIHlMIHi
gBRvXo8qm3WXAuvKe5Ig5AuE28obNqGBsqSBrzCBrDELMAkGA1UEBhMCVVMxCzAJ
BgNVBAgTAk5IMRIwEAYDVQQHEwlXb2xmZWJvcm8xIzAhBgNVBAoTGlB5dGhvbiBT
b2Z0d2FyZSBGb3VuZGF0aW9uMRwwGgYDVQQLExNJbmZyYXN0cnVjdHVyZSBUZWFt
MQ8wDQYDVQQDFAZQU0ZfQ0ExKDAmBgkqhkiG9w0BCQEWGWluZnJhc3RydWN0dXJl
QHB5dGhvbi5vcmeCFQCUbX1jWaBWMqgb+zpfP76Ht7B/4DANBgkqhkiG9w0BAQsF
AAOCAQEAR57NNg5fvj+tSrgdu6CmXvP65E1buECC7dRedRnJ2yDenI8o1jiFmMBD
jG/+x0whosZREsQIdRIm0f+1cmiN7voYs3tJB8j2EKH9i3+usifQJiOxN0jv2i0E
7ty2LLKwVhvYZQ8VxHGFdGnwUwGqKFtg+0C0ybrSmklGPY7uTxCio2sx28x4tIuU
HhNzp4DeCelgj+1OodrBW0GcZm1O9fnCew3nsuiN+E94/ptaN5F0FkLuEYC8qqq9
AqD60RC7HEJDWY+I33sSG0qcCNmkbyHkUUFxgD4+OrM7YA9/X9mixvRaUeS78EtS
YdKdKzNYasuc6cNDSo2I7HccjGE3ig==
-----END CERTIFICATE-----
```

PostgreSQL

The Python Infrastructure offers PostgreSQL databases to services hosted in the Rackspace datacenter.

- Currently running PostgreSQL 9.4
- Operates a 2 node cluster with a primary node configured with streaming replication to a replica node.
  - Each node is running a 15 GB Rackspace Cloud Server.
- Each app node has pgbouncer running on it pooling connections.
  - The actual database user and password is only known to pgbouncer, each node will get a unique randomly generated password for the app to connect to pgbouncer.
- The primary node also backs up to Rackspace CloudFiles in the ORD region via WAL-E. A full backup is done once a week via a cronjob and WAL-E does WAL pushes to fill in between the full backups.

Creating a New Database

1. Edit `pillar/postgresql/server.sls` and edit the `databases` dictionary to include a new entry where the left side is the name of the database you wish to create, and the right hand side is the name of

the user to create.

2. Edit `pillar/secrets/postgresql-users/all.sls` to add a new entry for the username with a randomly generated password.

Giving Applications Access

1. Create a sls under `pillar/{dev,prod}/secrets/postgresl-users` for the role for the application and add it to `pillar/{dev,prod}/top.sls`.
2. Add the `postgresql.client` state to the role for the application or to the states for the application.
3. Setup the application to the connect the server(s), there is one primary read/write server and zero or more (currently one) read only slaves. The addresses for this can be discovered using the service discovery mechanisms detailed here.

```
dattabases:
  {{with service "primary.postgresql@aid1"}}
  primary: {{ "{{(index . 0).Address}}" }}:{{ "{{(index . 0).Port}}" }}
  {{end}}
  read_only:
    {{range service "replica.postgresql@aid1"}}
    - {{.Address}}:{{.Port}}{{end}}
```

Clients should also be configured with `sslmode = verify-full`, `sslrootcert = /etc/ssl/certs/PSF_CA.pem`, and `host = postgresql.psf.io`. The ip addresses from above should be configured as `hostaddr` instead of `host`. This will ensure that the client will connect via TLS, verify the certificate is valid and from our internal CA, and will verify that it is for the postgresql server. An example of this in Django is:

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql_psycopg2",
        "HOST": "postgresql.psf.io",
        "NAME": "dbname",
        "USER": "dbname",
        "PASSWORD": "the password",
        "OPTIONS": {
            "hostaddr": "192.168.50.1",
            "sslmode": "verify-full",
            "sslrootcert": "/etc/ssl/certs/PSF_CA.pem",
        },
    },
}
```

Application Integration

The PostgreSQL has been configured to allow an application to integrate with it to get some advanced features.

(A)synchronous Commit

By default the PostgreSQL primary will ensure that each transaction is commited to persistent storage on the local disk before returning that a transaction has successfully been commited. However it will asynchronously replicate that transaction to the replicas. This means that if the primary server goes down in a way where the disk is not recoverable prior to replication occuring than that data will be lost.

Applications may optionally, on a per transaction basis, request that the primary server has either given the data to a replica server or that a replica server has also written that data to persistent storage.

This can be acchived by executing:

```
-- Set the transaction so that a replica will have received the data, but
-- not written the data out before the primary says the transaction is
-- complete.
SET LOCAL synchronous_commit TO remote_write;

-- Set the transaction so that a replica will have written the data to
-- persistent storage before the primary says the transaction is complete.
SET LOCAL synchronous_commit TO on;
```

Obviously each of these options will mean the write will fail if the primary cannot reach the replica server. These options can be used when ensuring data is saved is more important than uptime with the minimal risk the primary goes completely unrecoverable.