OpenStack Swift

Like S3, but
Data Plane Python
Distributed with no master
Scalable for the world’s largest clusters
How?
Overall Architecture

Internet

Internal clients

LB

Proxy

Auth

Keystone

Client traffic

Replication

Storage

Storage

Storage

Storage
Key Architectural Features

- Rational distribution of functions front to back
  - Proxy hides the implementation for the sake of helpful isolation
  - But layers are not isolated for fewer traversals
- DHT and Ring
  - No directories; accounts and containers (buckets) are Rings
  - Fully static, fully distributed with no leader
- Eventual consistency
  - C in CAP is given up
  - Replicators
  - Auditors
  - Updaters
Python Highlights

- Workers and fork()
- Eventlet
  - Greenthreads
    - eventlet.hubs
    - eventlet.socket
  - WSGI
    - eventlet.wsgi
    - Middleware – slo, s3api, cors
- swift.common.swob
- Buffered HTTP client
- rsync
- Evolving together with Python
  - Py3
  - No more pickles
Workers and fork()

- Workers are forked in a conventional way
  ```python
def run_wsgi(conf_path, app_section, *args, **kwargs):
    read_fd, write_fd = os.pipe()
    pid = os.fork()
    if pid == 0:
      def notify():
        os.write(write_fd, b'ready')
        os.close(write_fd)
      run_server(conf, logger, sock, ready_callback=notify)
  ```

- The multiprocessing is used, but only for semaphores
  - Most of it is not flexible enough

- Each worker hosts a number of green threads
  - Number of workers is the number of cores
  - Number of threads is the number of requests
Eventlet

- Eventlet monkey-patches everything, thus enabling green threads

```python
from eventlet import GreenPool, wsgi
eventlet.hubs.use_hub('poll')
logging._lock = logging.threading.RLock()
def run_server(conf, logger, sock, global_conf=None):
    eventlet.patcher.monkey_patch(all=False, socket=True,
                                 select=True, thread=True)
    app = loadapp(conf['__file__'], global_conf=global_conf,)
    pool = GreenPool(size=max_clients)
    server_kwargs = { 'custom_pool': pool, ...... }
    wsgi.server(sock, app, wsgi_logger, **server_kwargs)
```

- We will see how app is defined on the next page
WSGI

- The WSGI app is PEP-(3)333

```python
class Application(object):
    def __init__(self, conf, logger=None, container_ring=None):
        self.container_ring = container_ring or \
            Ring(swift_dir, ring_name='container')
    def __call__(self, env, start_response):
        req = self.update_request(Request(env))
        return self.handle_request(req)(env, start_response)
```

- Beyond the app, WSGI permeates the whole project

```python
def __call__(self, env, start_response):
    req = Request(env)
    self.logger.txn_id = req.headers.get('x-trans-id', None)
    if not check_utf8(wsgi_to_str(req.path_info)):
        res = HTTPPreconditionFailed(
            body='Invalid UTF8 or contains NULL')
```
swift.common.swob

- Replicates Webob which has an unstable protocol

```python
class Request(object):
    method = _req_environt_property('REQUEST_METHOD')
    content_length = _header_int_property('content-length')
    def __init__(self, environ):
        self.environ = environ
        self.headers = HeaderEnvironProxy(self.environ)

class HeaderEnvironProxy(MutableMapping):
    def __setitem__(self, key, value):
        if value is None:
            self.environ.pop(header_to_environ_key(key), None)
        elif isinstance(value, six.binary_type):
            self.environ[header_to_environ_key(key)] = value.decode('latin1')
        else:
            self.environ[header_to_environ_key(key)] = str(value)
```
HTTP Client

- This time, we do not fork, only override

```python
class BufferedHTTPConnection(HTTPConnection):
    response_class = BufferedHTTPResponse

class BufferedHTTPResponse(HTTPResponse):
    def readline(self, size=1024):
        # You'd think Python's httpplib would provide this, but
        # it doesn't. It does, however, provide a comment in the
        # HTTPResponse class:
        #   # XXX It would be nice to have readline and __iter__
        #   # for this, too.
        # Yes, it certainly would.
        while (b'\n' not in self._readline_buffer
               and len(self._readline_buffer) < size):
            read_size = size - len(self._readline_buffer)
            chunk = HTTPResponse.read(self, read_size)
            if not chunk:
                break
```
We mostly use our own method (ssync) over HTTP for updating databases

Objects are replicated with rsync

```python
class ObjectReplicator(Daemon):
    def rsync(self, node, job, suffixes):
        self.rsync_module = '{replication_ip}::object'
        args = ['rsync', '--recursive', '--whole-file',
                '--human-readable', '--xattrs',
                '--itemize-changes', '--ignore-existing',
                '--timeout=%s' % self.rsync_io_timeout,
                '--contimeout=%s' % self.rsync_io_timeout, ...]
        rsync_module = r_m_interpolation(self.rsync_module, node)
        args.append(join(rsync_module, node['device'], ...))
        proc = subprocess.Popen(args, stdout=subprocess.PIPE, ...)
```
For More Information

Source:

https://opendev.org/openstack/swift/

Documentation:

https://docs.openstack.org/swift/latest/

    https://docs.openstack.org/swift/latest/overview_architecture.html

Questions:

irc://irc.oftc.net/#openstack-swift

mailto:zaitcev@redhat.com