

# Google App Engine

For Mac Administrators

Welcome

Ed Eigerman [edward@eigerman.com](mailto:edward@eigerman.com) @trashmonkey

Jeff Kreer [jkreer@google.com](mailto:jkreer@google.com) @jdkreer

# You Down With HTTP?

- HTTP has become the go-to server service.
- LAMP is cool, but not perfect.
  - Requires maintaining a server and the whole stack.
  - Security is hard.
  - Dynamic content gets complicated quickly.
  - Scaling takes planning.
  - Geographically fixed.

More and more as administrators when we need a service that our clients are going to communicate with we're turning to HTTP. It's easier and simpler than some other options and there's lots of client code that we can reuse to connect to it.

The most common way to implement HTTP as a service is, of course, the LAMP stack or if you're a hardcore Mac nerd, or have some otherwise useless Xserves, MAMP. I have nothing to say against LAMP. It's great, but it's not perfect.

You have to maintain the entire stack, usually including the server itself.

You have to make sure it's as secure as you want it to be. This includes managing user accounts.

# Google Cloud Platform

- Run on Google's Infrastructure
- [cloud.google.com](https://cloud.google.com)
- Bunch of things
  - Of which App Engine is one.



Google makes a number of cloud service products. 13 according to this graphic. They are all pretty cool. The obvious competitors are AWS and Azure, of course, and some of these are pretty much in line with those things (which I don't know much about), but we're going to talk about one specific cloud offering. AppEngine.

# Google App Engine

- “Google App Engine is a platform for building scalable web applications and mobile backends.”
- You can't make this stuff up.



# Google App Engine

- Automatic Scaling
- Global Availability
- Built in security
- Built in account management
- Manage only your code
- Automated Security Scanning



So the big advantage of AppEngine is that you're running your code on Google's infrastructure, which is pretty good.

# Downsides

- It's a special snowflake
- NoSQL
- Only does one thing
- Must Have Google Account
- Hosted By Someone Else
- Can't Just Throw Some Stuff On The Server



It's not perfect.

# Cost

- Really Cheap
- Some Amount Is Free
- More Than That Costs \$Complicated Pricing
- But Still Cheap
- For Instance:
  - 70,000 Simian Clients are about \$13/day\*

\*If we paid for it

	FREE LIMIT PER DAY	PRICE ABOVE FREE LIMIT
Instances	20 instance hours	\$0.01 / instance / hour
Cloud Database (MySQL)	<ul style="list-style-type: none"> <li>• 50k read/write/email</li> <li>• 1 GB storage</li> </ul>	<ul style="list-style-type: none"> <li>• \$0.04 / 100k read or write ops</li> <li>• Small operations free*</li> <li>• \$0.18 / GB / month</li> </ul>
Network Traffic (outgoing)	1 GB	\$0.12 / GB
Network Traffic (incoming)	1 GB	FREE
Cloud Storage	5 GB	\$0.036 / GB / month
Memcache	<ul style="list-style-type: none"> <li>• Free Usage of Shared Pool</li> <li>• No free quota for Dedicated Pool</li> </ul>	<ul style="list-style-type: none"> <li>• Free Usage of Shared Pool</li> <li>• Dedicated Pool: \$0.06 / GB / hour</li> </ul>
Search	<ul style="list-style-type: none"> <li>• 1000 basic operations</li> <li>• 0.01 GB indexing documents</li> <li>• 0.25 GB document storage</li> <li>• 100 searches</li> </ul>	<ul style="list-style-type: none"> <li>• 0.50 / 10k searches</li> <li>• 0.00 / GB indexing documents</li> <li>• 0.18 / GB / month Storage</li> </ul>
Email API	100 recipients	<a href="#">Contact Sales</a>
Logs API	100 MB	\$0.12 per GB
Task Queue	5 GB	\$0.036 / GB / month
Logs Storage	1 GB	\$0.036 / GB / month
SSL, Virtual IPs	-	\$29 / virtual IP / month
Bundled Services	CDN, Image Manipulation, PageSpeed, DNS SSL, Certificates, Socket API, Task Queue API, URL Fetch, Users API	



# Choice of Environments

- Java
- PHP
- Go
- Python



# AppEngine Flow

- app.yaml File
- App main calls handlers
- WebApp2 Library Deals With Request and Response



# We Go Hands On

- <https://cloud.google.com/appengine/docs/python/gettingstartedpython27>
- [https://cloud.google.com/appengine/downloads#Google App Engine SDK for Python](https://cloud.google.com/appengine/downloads#Google_App_Engine_SDK_for_Python)

# Hello, World!

- Install SDK
- Create Config File
- Create Our First Handler

- Drag Install GoogleAppEngineLauncher
- Launch It
- Make Symlinks By Clicking “OK”
- Create a Directory Called HelloWorld. Or Don't, I'm a Slide, Not a Cop
- Create Two Files in That Directory:
  - app.yaml
  - helloworld.py

# app.yaml

```
version: 1
runtime: python27
api_version: 1
threadsafe: true

handlers:
- url: /*.*
  script: helloworld.app
```

# helloworld.py

```
import webapp2

class MainPage(webapp2.RequestHandler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/plain'
        self.response.write('Hello, World!')

app = webapp2.WSGIApplication([
    ('/', MainPage),
], debug=True)
```

# Launch It

```
/usr/local/google_appengine/dev_appserver.py </path/dir/>
```



# Try It

`http:127.0.0.1:8080`

# WebApp2

- WSGI Library
- You Could Use
  - Django
  - CherryPy
  - Pylons
  - web.py
  - web2py
- Handles Request And Response

# WebApp2

- WebApp2 Applications Have Two Parts

- Part 2: Request Handlers

```
class MainPage(webapp2.RequestHandler):  
    def get(self):  
        self.response.headers['Content-Type'] = 'text/plain'  
        self.response.write('Hello, World!')
```

- Part 1: App Declaration

```
app = webapp2.WSGIApplication([  
    ('/', MainPage),  
], debug=True)
```

# Users

```
from google.appengine.api import users

import webapp2

class MainPage(webapp2.RequestHandler):

    def get(self):
        # Checks for active Google account session
        user = users.get_current_user()

        if user:
            self.response.headers['Content-Type'] = 'text/html; charset=utf-8'
            self.response.write('Hello, ' + user.nickname())
        else:
            self.redirect(users.create_login_url(self.request.uri))

app = webapp2.WSGIApplication([
    ('/', MainPage),
], debug=True)
```

# Users

- AppEngine Handles User Authentication
- `users.get_current_user()` Returns None if the User Is Not Logged in to Google
- Otherwise Returns Google AppEngine User

# Project 2: GuestBook

- Forms
- Datastore

# app.yaml

```
version: 1
runtime: python27
api_version: 1
threadsafe: true

handlers:
- url: /*
  script: guestbook.app

libraries:
- name: webapp2
  version: latest
```

# guestbook.py

```
import cgi
from google.appengine.api import users
import webapp2

MAIN_PAGE_HTML = """\
<html>
  <body>
    <form action="/sign" method="post">
      <div><textarea name="content" rows="3" cols="60"></textarea></div>
      <div><input type="submit" value="Sign Guestbook"></div>
    </form>
  </body>
</html>
"""

class MainPage(webapp2.RequestHandler):
    def get(self):
        self.response.write(MAIN_PAGE_HTML)

class Guestbook(webapp2.RequestHandler):
    def post(self):
        self.response.write('<html><body>You wrote:<pre>')
        self.response.write(cgi.escape(self.request.get('content')))
        self.response.write('</pre></body></html>')

app = webapp2.WSGIApplication([
    ('/', MainPage),
    ('/sign', Guestbook),
], debug=True)
```



# guestbook.py

- Two Handlers

```
app = webapp2.WSGIApplication([
    ('/', MainPage),
    ('/sign', Guestbook),
], debug=True)
```

- post Method

```
class Guestbook(webapp2.RequestHandler):
    def post(self):
        ...
```

- Accessing Request Post Data (Also Works For get)

```
self.request.get('content')
```

# DataStore

- Where Things Start Getting Interesting
- `ndb.Model`
- `ndb` Properties
  - `IntegerProperty`
  - `FloatProperty`
  - `BooleanProperty`
  - `StringProperty`
  - `TextProperty`
  - `BlobProperty`
  - ...

# guestbook.py

```
import cgi
import urllib

from google.appengine.api import users
from google.appengine.ext import ndb

import webapp2

MAIN_PAGE_FOOTER_TEMPLATE = """\
<form action="/sign?%s" method="post">
  <div><textarea name="content" rows="3" cols="60"></textarea></div>
  <div><input type="submit" value="Sign Guestbook"></div>
</form>
<hr>
<form>Guestbook name:
  <input value="%s" name="guestbook_name">
  <input type="submit" value="switch">
</form>
<a href="%s">%s</a>
</body>
</html>
"""

DEFAULT_GUESTBOOK_NAME = 'default_guestbook'

# We set a parent key on the 'Greetings' to ensure that they are all
# in the same entity group. Queries across the single entity group
# will be consistent. However, the write rate should be limited to
# ~1/second.
```

# guestbook.py

```
def guestbook_key(guestbook_name=DEFAULT_GUESTBOOK_NAME):  
    """Constructs a Datastore key for a Guestbook entity.  
  
    We use guestbook_name as the key.  
    """  
    return ndb.Key('Guestbook', guestbook_name)
```

# guestbook.py

```
class Author(ndb.Model):  
    """Sub model for representing an author."""  
    identity = ndb.StringProperty(indexed=False)  
    email = ndb.StringProperty(indexed=False)
```

# guestbook.py

```
class MainPage(webapp2.RequestHandler):
    def get(self):
        self.response.write('<html><body>')
        guestbook_name = self.request.get('guestbook_name',
                                           DEFAULT_GUESTBOOK_NAME)

        # Ancestor Queries, as shown here, are strongly consistent
        # with the High Replication Datastore. Queries that span
        # entity groups are eventually consistent. If we omitted the
        # ancestor from this query there would be a slight chance that
        # Greeting that had just been written would not show up in a
        # query.
        greetings_query = Greeting.query(
            ancestor=guestbook_key(guestbook_name)).order(-Greeting.date)
        greetings = greetings_query.fetch(10)

        user = users.get_current_user()
        for greeting in greetings:
            if greeting.author:
                author = greeting.author.email
                if user and user.user_id() == greeting.author.identity:
                    author += ' (You)'
                self.response.write('<b>%s</b> wrote:' % author)
            else:
                self.response.write('An anonymous person wrote:')
            self.response.write('<blockquote>%s</blockquote>' %
                               cgi.escape(greeting.content))

        if user:
            url = users.create_logout_url(self.request.uri)
            url_linktext = 'Logout'
        else:
            url = users.create_login_url(self.request.uri)
            url_linktext = 'Login'

        # Write the submission form and the footer of the page
        sign_query_params = urllib.urlencode({'guestbook_name':
                                              guestbook_name})
        self.response.write(MAIN_PAGE_FOOTER_TEMPLATE %
                            (sign_query_params, cgi.escape(guestbook_name),
                             url, url_linktext))
```

# guestbook.py

```
class Guestbook(webapp2.RequestHandler):
    def post(self):
        # We set the same parent key on the 'Greeting' to ensure each
        # Greeting is in the same entity group. Queries across the
        # single entity group will be consistent. However, the write
        # rate to a single entity group should be limited to
        # ~1/second.
        guestbook_name = self.request.get('guestbook_name',
                                           DEFAULT_GUESTBOOK_NAME)
        greeting = Greeting(parent=guestbook_key(guestbook_name))

        if users.get_current_user():
            greeting.author = Author(
                identity=users.get_current_user().user_id(),
                email=users.get_current_user().email())

        greeting.content = self.request.get('content')
        greeting.put()

        query_params = {'guestbook_name': guestbook_name}
        self.redirect('/?' + urllib.urlencode(query_params))
```

# guestbook.py

```
app = webapp2.WSGIApplication([
    ('/', MainPage),
    ('/sign', Guestbook),
], debug=True)
```



# More On Datastore

- There is GQL, But It's Not Very Satisfying
- Datastore Specific Methods
  - `model.get_by_id('123')`
  - `model.query(model.property='123')`
  - `query.filter(model.other_property>='abc')`

# Some Final Notes

- I Lied, You Can Use Static Files

`handlers:`

```
- url: /stylesheets
  static_dir: stylesheets
```

- You Can Use Lots of Different HTML Template Systems
  - Jinja2
  - AngularJS

Q & A