Building Scalable Web Apps with Python and Google Cloud Platform

Dan Sanderson, April 2015
June 2015
pre-order now
Agenda

• Introducing GCP & GAE
• Starting a project with gcloud and Cloud Console
• Understanding request handlers and instances
• Using Python libraries and frameworks
• Using Flask
• Testing
• Deploying, monitoring, debugging
Google Cloud Platform

- Computation
- Networking
- Data storage
- Data analysis
- Deployment, monitoring, source control, access control, services…
- Scaling infrastructure
Google Cloud Platform

- Web applications and services
- Gaming and mobile backends
- Scientific computing
- Video and graphics rendering
Google Cloud Platform

Computation and networking

Google App Engine (GAE)

Google Container Engine (GKE)

Google Compute Engine (GCE)
Google Cloud Platform

Data storage and analysis

Google Cloud
Datastore

Google Cloud
SQL

Google Cloud
Storage
Google App Engine

- Scalable serving infrastructure optimized for web apps
- High level of abstraction
- Streamlined, sandboxed, language-specific
- Scalable services
  - (Some services specific to App Engine)
Google App Engine

• Easy to deploy, easy to manage
• Pay only for what you use
  • Free tier, daily allocation
• Preview launched in 2008 as Python-only; Java, Go, PHP added later
Setting up a development environment

- Cloud SDK: http://cloud.google.com/sdk/
  - The gcloud command
- Python 2.7
- git
- pip, virtualenv
gcloud

• Authentication
• Project set-up
• Component installation and management
• Service-specific tools
gcloud

• App Engine SDK, Python libraries
• App Engine development server
  • Development server console
• App Engine deployment
Cloud Console

https://console.developers.google.com/
Demo
gcloud

- gcloud auth login
- gcloud init <project>
  - creates .gcloud file, clones git repo
- gcloud app run app.yaml
- gcloud app deploy app.yaml

(gcloud preview app …)
Instances and Request Handlers
Instances and Request Handlers

- Request handlers are ephemeral: now you see them, now you don’t
- Can’t rely on data persistence between requests
- Use storage services to persist data
Instances and Request Handlers

• In practice, app initialization is expensive

• An app instance is long running, can handle multiple requests in its lifetime

• Environment initialized; instance memory loaded

• Started and stopped as needed

• Can’t rely on a single user’s session to go to the same instance
Instance

Request handler
threading enabled
threading enabled
threading enabled
The Python Runtime Environment
Runtime Environment

- Sandboxing
- Data isolation
- Performance isolation
- Python 2.7: app code must be Python only
- Sandboxing $\rightarrow$ scalability
Runtime Environment

- Limits
  - Request timer
  - Restricted access to filesystem, sockets
  - More performance isolation: RAM, CPU
  - Data sizes: requests, responses, API calls, storage objects
- Limits $\rightarrow$ scalability
Python Libraries

- Runtime includes standard lib and GAE libs automatically
- App code must be “pure Python”
  - Can add “pure Python” libs to your app code
- App config can request specific libs be added
Python Libraries

standard libraries
GAE libraries

application code
Python Libraries

- Python 2.7 standard library
- GAE libraries (google.appengine...)
- Runtime / dev server has GAE libs on path already
- For other scripts, like test runners:
  ```python
  import sys
  sys.path.insert(0,
                  '../google-cloud-sdk/platform/
                     google_appengine/)
  ```
Python Libraries

- standard libraries
- GAE libraries

application code
Python Libraries

standard libraries
GAE libraries

application code
./lib/ ...
Python Libraries

% mkdir ./lib
% pip install -t lib Flask==0.10
Python Libraries

% mkdir ./lib
% pip install -t lib Flask==0.10

# appengine_config.py
from google.appengine.ext import import vendor
vendor.add('lib')
Python Libraries

% mkdir ./lib
% pip install -t lib Flask==0.10

# appengine_config.py
from google.appengine.ext import import vendor
vendor.add('lib')

# .gitignore
lib/*
Python Libraries

standard libraries
GAE libraries

application code
./lib/ ...
Python Libraries

standard libraries
GAE libraries

requested libraries

application code

./lib/ ...

app.yaml: “libraries: …”
Python Libraries

django
endpoints
jinja2
lxml
markupsafe
matplotlib
MySQLdb
numpy
PIL

protorpc
PyAMF
pycrypto
setuptools
ssl
webapp2
webob
yaml
Python Libraries

# app.yaml

libraries:
  - name: jinja2
    version: "2.6"
  - name: markupsafe
    version: "0.15"
Python Libraries

standard libraries
GAE libraries

requested libraries

application code

./lib/ ...

app.yaml: “libraries: …”
Python Libraries

Use virtualenv to contain requested libraries in your development environment.

% virtualenv venv
% source venv/bin/activate
(venv)% pip install jinja2==2.6
Python Libraries

Use `pip requirements files` to install and document dependencies.

```bash
# requirements_cfg.txt
jinja2==2.6
markupsafe==0.15

(venv)% pip install -r requirements_cfg.txt
```
Python Libraries

Use pip requirements files to install and document dependencies.

# requirements_vnd.txt
Flask==0.10
google-api-python-client

(venv)% pip install -t lib \ 
    -r requirements_vnd.txt
Using Flask
Using Flask

• Install Flask in ./lib

• Request jinja2 and markup safe in app.yaml; install in virtualenv
  • lib/markupsafe’s C-based speed-up is not usable but app.yaml will override with built-in

• app = Flask()

• No need to call app.run(); GAE dev server will take care of it
from google.appengine.ext import ndb

class Entry(ndb.Model):
    title = ndb.StringProperty(required=True)
    text = ndb.TextProperty(required=True)
    last_updated_date = ndb.DateTimeProperty(auto_now=True)
entry = Entry(
    title=flask.request.form[‘title’],
    text=flask.request.form[‘text’])
entry.put()

entries = Entry.query().order(
    -Entry.last_updated_date)
for entry in entries:
    # ...
from google.appengine.api import users

user = users.get_current_user()
if user:
    # user.nickname...

login_url = users.create_login_url('/')
logout_url = users.create_logout_url('/')
Google Accounts

# app.yaml

handlers:
- url: /add
  script: main.app
  login: required

- url: .*
  script: main.app
Demo
Testing
Testing

• It’s just Python, right?
• Activate your virtualenv
• Make sure google_appengine, the app root, and ./lib are in PYTHONPATH
• (also yaml)
• Use your favorite test framework (e.g. unittest)
testbed

• Service API stubs for a clean test environment, including Cloud Datastore

• Provided by the GAE SDK
  • google.appengine.ext.testbed

• Datastore: can fuzz test consistency guarantees

• Users: can set signed-in status, address, is-admin status
import unittest
from google.appengine.ext import import testbed

import main
import models

class MyBlogTestCase(unittest.TestCase):
    def setUp(self):
        # ...

    def tearDown(self):
        # ...
class MyBlogTestCase(unittest.TestCase):
    def setUp(self):
        self.tb = testbed.Testbed()
        self.tb.activate()
        self.tb.init_datastore_v3_stub()
        self.tb.init_memcache_stub()
        self.tb.init_user_stub()

        main.app.config['TESTING'] = True
        self.app = main.app.test_client()
class MyBlogTestCase(unittest.TestCase):
    def tearDown(self):
        self.tb.deactivate()
class TestEntry(MyBlogTestCase):
    def test_entry(self):
        m = models.Entry(
            title='Test',
            text='Test entry text')
        m.put()
        self.assertIsNotNone(m.last_updated_date)
class MyBlogInteractions(MyBlogTestCase):
    def test_no_entries(self):
        rv = self.app.get('/')
        assert 'No entries here so far' in rv.data

    def test_one_entry(self):
        e = models.Entry(title='Test', text='Test text')
        e.put()

        rv = self.app.get('/')
        assert 'No entries here so far' not in rv.data
testbed

tb.init_all_stubs()
tb.init_blobstore_stub()
tb.init_datastore_v3_stub()
tb.init_memcache_stub()
tb.init_images_stub()
tb.init_mail_stub()
tb.init_taskqueue_stub()
tb.init_urlfetch_stub()
tb.init_user_stub()
tb.init_xmpp_stub()

cloud.google.com/appengine/docs/python/tools/localunittesting
Flask Testing

flask.pocoo.org/docs/0.10/testing/
Deploying, Monitoring, Debugging
Demo
Where to go from here...

- Modules
- Task Queues
- Advanced Cloud Datastore: ndb, transactions, memcache
- Cloud SQL
- Cloud Storage
- Compute / Container Engine
- Managed VMs (beta)
Thank you!

cloud.google.com

ae-book.appspot.com

Programming Google App Engine with Python, ... with Java
Early Access / pre-order now
Fully available June 2015

Dan Sanderson
profiles.google.com/
dan.sanderson