Building Scalable Web Applications with Google App Engine

Dan Sanderson

May 15, 2012


Google
Developers


Build and Run Scalable Web Apps on Google's Infrastructure
Programming
Google App Engine
O'REILLY
Google Press
Dan Sanderson

scalability

Each user gets the same quality of experience, regardless of how many users there are.

Google App Engine

- Platform for building scalable web applications
- Built on Google infrastructure
- Based on Google's internal best practices
- Pay for what you use
  - Apps, instance hours, storage, bandwidth, service calls
  - Free to start!
- Preview opened 2008; out of preview in 2011

Google App Engine

- Easy development
- Easy deployment
- No servers to manage, no OS to update; App Engine does this for you
- Based on standard technologies

Google App Engine

- Request handling infrastructure
- Application runtime environments
- Services
- Tools and libraries
- Administration Console

Demo

Request Handlers

Request Handlers

- Requests come in, responses go out
- Request handler computes the response for a request
- Call services to manipulate stored data, perform special tasks
- Handler created when request arrives, destroyed after response is sent
- “Stateless” → scalability

Request Handlers

- All code runs in a request handler
- Web hooks
- Unit of computation for larger jobs

Instances

- Longer-lived containers for request handlers
- Reduce initialization costs
- Maximize CPU utilization
- Exploit instance RAM
- Parameters to tune instance creation and destruction
- “Instance hours” are the billable unit for computation

Runtime Environments

Runtime Environments

- Python
  - Python 2.7; Python 2.5
  - WSGI; CGI
- Java
  - Full Java 6 JVM, J2EE servlet container
  - Java, Scala, Ruby (JRuby), PHP (Quercus), JavaScript (Rhino), Python (JPython)
- Go

Runtime Environments

```
import logging
import webapp2

class MainPage(webapp2.RequestHandler):
    def get(self):
        status = self.request.get(‘status’)
        if status not in ('running', 'error', 'success'):
            logging.warning('Invalid status')
            status = ‘error’

        template = template_env.get_template('home.html')
        context = {
            'status': status,
        }
        self.response.out.write(template.render(context))

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

Runtime Environments

```
import javax.servlet.http.*;

@SuppressWarnings("serial")
public class MainPageServlet extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse resp)
        throws IOException, ServletException {
            String status = req.getAttribute("status");
            if (!status.equals("running") ||
                !status.equals("success"))
                status = "error";

            req.setAttribute("status", status);
            resp.setContentType("text/html");
            RequestDispatcher jsp =
                req.getRequestDispatcher("/WEB-INF/home.jsp");
            jsp.forward(req, resp);
        }
    }
```

Runtime Environments

- Sandboxing
  - Data isolation
  - Performance isolation
- Sandboxing → scalability

Runtime Environments

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

Services

Services

- Features with their own scalable infrastructure
- Architecturally distinct from the runtime environments
- Synchronous and asychronous calling APIs

Services

- Data storage
- Communication
- Data processing and manipulation
- Computation primitives

Datastore

- Scalable object storage
- Replication using Paxos
- Named properties, typed values
- “Schemaless;” data modeling libraries
- Keys, kinds, and indexed queries

Datastore

```
from google.appengine.ext import db

class UserPrefs(db.Model):
    user = db.UserProperty(auto_current_user_add=True)
    created_datetime = db.DateTimeProperty(auto_now_add=True)
    tz_offset = db.IntegerProperty(default=0)
    subscribed = db.BooleanProperty()
```

```
class NewUser(webapp2.RequestHandler):
    def post(self):
        new_user = UserPrefs()
        new_user.subscribed = \
            self.request.get('subscribed')) is not None
        new_user.put()

        self.redirect('/welcome')
```

Datastore

- Datastore queries
  - *All queries are pre-indexed*
  - Built-in indexes do most of the work
  - Custom indexes for complex queries; SDK helps you out
  - Cursors

Datastore

```
query = UserPrefs.all()
query.filter('subscribed =', True)

for userPref in query:
    # ... userPref.user ...
```

Datastore

Each user gets the same quality of experience, regardless of how many users there are.

- Fetch by key: O(1)
- Create and update: O(1)
- Queries: O(*number of results*)
  - *not* the total number of entities!

Datastore

- Transactional
- Local transactions
- Strong consistency
- Eventually consistent cross-group transactions
- Local and global indexes

Memcache

- Non-durable key-value store
- Distributed global cache
- Atomic set/add/replace, get, incr/decr of a single value
- Essential technique for speeding up user requests

Blobstore

- Datastore and Memcache limited to 1MB objects
- Blobstore object size is *unlimited*
- Direct uploads and downloads
- Limited read/write app access

Communication

- HTTP
  - Out: URL Fetch
  - In: request handlers!
- *app-id*.appspot.com
- *www.your-domain.com*
- SSL: https://*app-id*.appspot.com/ ;
  custom domain support soon

Communication

- Email
  - Out: a simple service call
  - In: request handlers!
- *app-id*@appspotmail.com
- *anything*@*app-id*.appspotmail.com
- “From” addresses: administrator, receiving address, current user (Google Accounts)

Communication

- Instant messages (XMPP)
  - Out: a simple service call
  - In: request handlers!

- Addresses (“JIDs”)
  - *app-id*@appspot.com
    anything@*app-id*.appspotchat.com
  - (No custom domain support.)

Communication

- Channels
- Real-time push messages to browsers
- Comet connection
- JavaScript adapter provided

Data Processing

- Images
- Search *(experimental!)*
- Prospective search *(experimental!)*

Task Queues

- Defer work out of the user request
- Add a task to a queue, loosely FIFO
- Producers and consumers

Task Queues

- Push queues
  - A task is a request to your app!
    URL path + data
  - Processed at a configurable rate
  - Retried until “successful”
  - Task chains
  - App is both producer and consumer

Task Queues

- Pull queues
  - A task is any kind of payload
  - Consumer API can pull tasks individually or in batches
  - App API; useful with backends!
  - Authenticated REST API

Task Queues

You can enqueue tasks in a datastore transaction!

Tools and Libraries

Language-specific APIs

- Access services using the idioms of the language (Python, Java, Go)
- Manage synchronous and asynchronous calls
- Stubs for testing

Data Modeling

- Translate between datastore entities and program objects
- Enforce data schemas and structures
- Python: ext.db, ext.ndb
- Java: datastore API; JDO, JPA; Objectify

Pipelines

- Execute multiple tasks over large-scale data with parallelism
- Construct task dependencies
- Job status monitor
- http://code.google.com/p/appengine-pipeline

MapReduce

- Large-scale data processing
- Simple programming model
  - map(item) → (key, value)*
  - *shuffle*
  - reduce(key, values) → result
- Can chain multiple MapReduces using pipelines

MapReduce

- App Engine implementation mostly in application code!
  - http://code.google.com/p/appengine-mapreduce
- Primitive operations in services, e.g. Blobstore
- *Experimental*

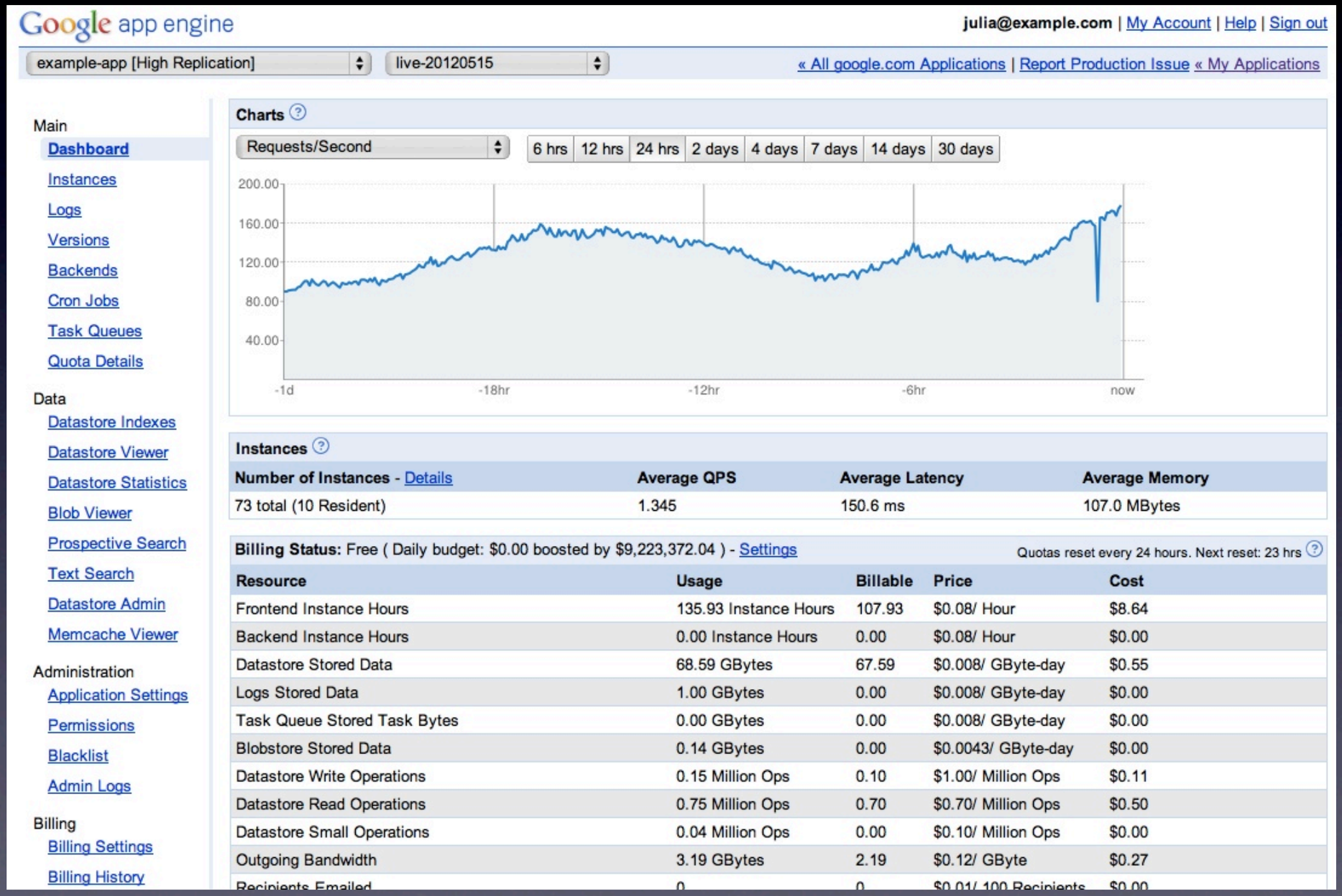
Development Server

- Simulates runtime environment and services on your local computer
- Suitable for functional testing
- Python: Launcher, dev_appserver.py
- Java: Eclipse plugin, dev_appserver.sh

Deployment Tools

- Deploy software, files, configuration
- Download logs for offline analysis
- Python: Launcher, appcfg.py
- Java: Eclipse plugin, appcfg.sh

Administration Console



Administration Console

- View traffic graphs
- Browse and search logs
- Inspect live data; backup/restore
- Manage instances
- Adjust budget and billing settings

Google App Engine

- Request handling infrastructure
- Computation infrastructure
- Storage: fast, durable, arbitrarily large
- Simple development model,
  high productivity
- Managed infrastructure: just add code

developers.google.com/appengine

appengine.google.com

ae-book.appspot.com

Programming Google App Engine, 2nd ed.
Summer 2012

Dan Sanderson
profiles.google.com/dan.sanderson