Google App Engine Python

Douglas A. Starnes‡*

Abstract—In recent years, one of the fastest growing trends in information technology has been the move towards cloud computing. The scalable concept of computing resources on demand allows applications to dynamically react to increased usage instead of having to keep resources in reserve that are often not in use but are still paid for. There are several popular entrants into this market including Google App Engine. Modeled after Google's own architecture for building applications, Google App Engine (GAE) provides a scalable solution for web-based applications and services including data storage, communications, application deployment and monitoring, and management tools. With GAE, developers have the option of writing applications using an API exposed to Python. The same benefits of using Python in other applications are available in the cloud.

Index Terms—cloud computing, web, google, application development

Overview

To assist developers in writing their applications, GAE provides a variety of frameworks and services which will be discussed later. An broad look at the entire application structure is in order first.

GAE can be broken down into three pieces: your application, the SDK and tools, and the server itself. The application is configured by a set of files using the YAML ("Yet Another Markup Language" or the recursive "YAML Ain't Markup Language") specification. The main configuration file must be named app.yaml and contains metadata about the application such as the name of the application and the version number. The application name must be unique across all of GAE. Google provides a service to look up available application names at registration. The application name will also be a subdomain of appspot.com where the application's default version will be. Also in the app.yaml file is a set of mappings. Both scripts and static content can be mapped to URL endpoints. There are built-in modules and other options such as security settings which can be configured in app.yaml as well.

Several important tools are provided by the SDK for GAE. Primary among these are appcfg and dev_appserver. The appcfg tool is used to deploy applications to the server based on the configuration in the app.yaml file. Other functions the appcfg tool includes are to download the code and server logs for an application and to manage cron jobs and datastore indexes. The other important tool is dev_appserver. The dev_appserver tool is a local development server that emulates the services of GAE. Not all services are equally emulated. For example, the cron service does not run locally. Also, the email service simply dumps a trace of the message contents to the logs. Two other tools of note are the bulkloader and remote_api_shell. The bulkloader is useful for migrating large amounts of existing data to GAE all at one time. The remote_api_shell provides an interactive Python session with the live datastore.

The GAE server exposes the API and services that your application is built upon and is emulated by the local development server. Among the most important services is the datastore. GAE also exposes an HTTP stack in webapp. Other services such as the task queue, federated identity with Google and GMail accounts, email and XMPP messaging exist.

* Corresponding author: douglas@poweredbyalt.net
‡ University of Memphis - Institute For Intelligent Systems

webapp

The webapp framework is a Python library to handle network traffic using the HTTP protocol. With webapp, the application defines request handlers that are mapped to endpoints using both the app.yaml settings as well as a set of URL routes defined within the request handler itself. Each handler is a class derived from a base RequestHandler class provided by webapp. The RequestHandler class primarily defines methods for handling the different HTTP methods (e.g. get, post). Also within the RequestHandler class are objects representing the request and response for the current HTTP transmission so the application can retrieve the URI requested or the status code and also return data to the caller or redirect to another resource. When the blobstore is needed, webapp defines a group of handlers to be used specifically with uploading data to the blobstore.

```
from google.appengine.ext import webapp
    from google.appengine.ext.webapp import util

    class MyHandler(webapp.RequestHandler):
        def get(self):
                # get data from the datastore,
                    #render a template ...
                    self.response.out.write(data)

    if __name__ == '__main__':
        application =
                webapp.WSGIApplication(
                            [('/.*', MyHandler)]
                    )
            util.run_wsgi_app(application)
```

Data Storage

GAE gives developers several avenues of data storage. For long term storage, data is persisted into the datastore and blobstore. Short term storage is available in the memcache.

The datastore is one of the most prominent services of GAE. The datastore is intended for structured table storage. While the

datastore uses tables, it is not a relational database. The non-relational nature of the the datastore puts it in the category of "NoSQL" in the opinions of some. Unlike the popular NoSQL databases MongoDB and CouchDB, the datastore is referred to as "column oriented". MongoDB and CouchDB are document-oriented and schema-less and do not define a formal data model for the documents. The GAE datastore requires a data model but is more flexible. The tables - or "kinds" to use GAE terms - are defined by a classes derived from a base Model class from the db module. This Model class has all of the CRUD (create, read, update and delete) operations built in. The developer then has only to define the properties of the kind along with data types, and options such as if the property is required and its length.

```
from google.appengine.ext import db

class Product(db.Model):
    name = db.StringProperty(required=True)
        price = db.FloatProperty(default=0.99)
        suppliers = db.StringListProperty()
```

Please notice the unusual yet useful StringListProperty. The GAE datastore can have columns that are lists. There is also a ListProperty type for non-string values.

Adding a kind to the datastore needs no preparation from the developer, other than to define the kind. Simply call the put method on an instance of the kind and the first time the datastore will take care of all the housekeeping to define the kind in the datastore.

```
product = Product()
# initialize required properties
product.put()
```

While the datastore does require a data model, the model can be defined or extended after the kind class has been defined. The Expando class in the db module has this capability.

For querying the datastore, there exists a simple language called GQL (Google Query Language) that as the name suggests, is similar to SQL. GQL has several limitations, mainly that it can only retrieve data. All other operations (insert, update, delete) must be performed programmatically. Selects can be performed programmatically as well but GQL provides a simpler way. GQL also has a few enhancements over SQL such as bound parameters that are referenced by position or name. The following code demonstrates this:

```
db.GqlQuery(
    "select * from Product where name = :1",
        "Gadget")

db.GqlQuery(
    "select * from Product where price <= :price_point",
        price_point=1.99)
```

The datastore also supports indexing. For simple queries, indexes are constructed by the datastore. Simple queries include those with only equality comparisons, and those with only one inequality comparison or one sort orders. Other queries must be defined manually. These are in another YAML configuration file called index.yaml. When the index.yaml file is deployed along with the app, the datastore will examine the definitions and build the indexes. Progress can be monitored through the online web control panel for GAE. Manually defining an index is not a lot of work because if GAE requires an index for a query that has not been defined, it will return an error along with a suggested definition. TextProperty and BlobProperty columns cannot be indexed. As is the case with relational databases, it is best to define indexes before GAE notices.

A complement to the datastore is the blobstore. The blobstore is intended to persist unstructured binary data such as images. A blob has a maximum size of 2 gigabytes. These blobs are immutable. Once created, blobs can be read or deleted but not modified. There is also no way to reference a blob from the datastore. The datastore does support the BlobProperty column type but that blob is stored in the datastore. The blobstore is separate. Managing blobs can only be done through the online control panel for the app. Here the user can view, download and delete blobs. The only way to get blobs into the datastore is through a web form. There is experimental support for writing files to the blob. This would be useful for creating blobs in response to a cron job or something else that does not require a user to start it.

For short term storage of small values exists the memcache. Entries in the memcache are simple key/value pairs. Entries stored in the memcache will eventually expire. By default, GAE keeps entries as long as there is enough memory. If an application begins to consume a lot of memory, older entries will be freed to make room for newer ones. Also, in the event of a system failure, entries will not be retained as they are stored in memory, not persisted to disk. Memcache values can be no more than 1MB in size.

Task Management

Most request to GAE should be short lived. There is a 30 second limit on HTTP requests. In the logs, requests are flagged as lengthy when they begin to exceed about 500ms. Anything more than 30 seconds will throw an exception and the request will be terminated. If a longer running task is needed, an application can start a background worker. These have a maximum time limit of 10 minutes. Creating a new worker is easy: call a method on the taskqueue API and pass it the endpoint of the worker along with an object of any parameters. There does not appear to be a mechanism using the default method of processing queues to have a callback method for notification of when a worker is complete. Using pull queues, an application can take over the method of processing queues itself. With pull queues there is a REST API so that the processing can be external to GAE.

If the task requires even more time, it can be handled by a backend. A backend is a separate GAE instance which has no constraints on time to run. Furthermore, backends are more configurable and have access to more resources such as memory and CPU. For applications using pull queues, tasks can be passed to a backend. Backends can consume large amounts of resources so there is an extra charge for them. They are billed in 15 minute increments up until the backend has been idle for 15 minutes. A backend can be resident meaning it must be shut down explicitly or dynamic meaning it will start in response to code and shut down after it have been idle for 15 minutes. Neither seems to have an impact on the hourly rate though. Backends do not scale automatically as normal GAE instances do. The number of backends is allocated explicitly in a configuration file (backends.yaml).

The last method of background processing GAE has is cron jobs. Cron jobs on GAE work similar to cron jobs on UNIX-based systems. In GAE, a configuration file named cron.yaml defines the tasks to be run. The cron.yaml file has entries for the endpoint of the task and the frequency of the task. The task frequency is expressed using a format that is more verbose than the UNIX crontab format but is also easier to interpret. For example to have a task run every 24 hours:

"every 24 hours"

is the expression to use in the cron.yaml file. More specific expressions such as the following are also possible:

"1st tue of november 0:00"

There are two differences to consider when using cron jobs on GAE. First is that cron jobs do not run in the local development environment. You can view what jobs are defined and access the endpoints for them but the schedule will not be followed. Second, cron jobs always run on the default version of the application. If you define a cron job in a development version of an app, it will not be run to avoid conflicts with the default version. Cron jobs always call endpoints using HTTP GET.

To remove a cron job from an application, remove its entry from cron.yaml and deploy the application. To remove all jobs from an application, deploy a cron.yaml without any job entries.

Application Environment

On the server, GAE hosts and serves applications at a subdomain of appspot.com that is the same as the registered application name in the app.yaml file. The official version of Python on GAE is 2.5.2. However, experience has shown that using 2.5.4 has no problems. There have been accounts of applications targeting 2.6 and 2.7 working with the development server. While these applications might run locally, any code that is specific to the later versions will not run on GAE. The development environment makes no attempt to ensure that the supported version of Python is being used. The Python Standard Library is available with a few exceptions. First several libraries such as PyYAML and simplejson have been added. Also, for security reasons, there are libraries which are not allowed such as marshal and socket. Importing these libraries will not cause an exception but import nothing so is the equivalent of a null operation. Any pure Python code that does not have dependencies on C extensions within the constraints above will run on GAE.

The SDK has versions for Windows, Mac OS X and Linux. On Windows and OS X there is a GUI launcher to access some of the command line tools. The launcher is helpful if more than one version of Python is running on the machine. The launcher has a setting to specify the location of Python to use with GAE on the development server. The development server also gives a console in the web control panel to run Python statements against the currently running application. The GAE SDK is also open source although modifying the source code will most likely result in an application which will fail to run correctly on the remote server.

Other Notable Features

GAE has a number of unique features that are outside the scope of this paper. The following are a few that deserve to be mentioned. First, it is worth knowing that there are two other languages supported by GAE, Java and Go. Go is a programming language developed by Google and described as a hybrid between C++ and Python. Go is experimental at the time of this writing. The features between the Python and Java runtimes are close to being identical. Python has a small lead as it was the first language for GAE. GAE also supports federated identity using Google accounts so anyone with a GMail or Google account can be authenticated using GAE with only a few lines of code. Also included with GAE is a profiling package called AppStats. Using this tool, very detailed timelines about the requests the application processes can be analyzed. The call stacks are also recorded and can be navigated through a web-based interface. AppStats works on both the remote server and local development environments. Finally, a new experimental library called ProtoRPC was recently added to the SDK. ProtoRPC simplifies the workflow for creating REST-based web services using GAE.

Conclusion

Benefits of using Python on GAE let developers prototype and develop applications in the cloud using familiar web technologies. A notable benefit is that GAE will support a free quota to test applications on the server before enabling billing. In addition, GAE integrates very well with the Python libraries for GData to access services such as Google Finance, Google Spreadsheets, Google Sites, and Picasa. Finally, there are several maintained application frameworks running on top of GAE that extend its functionality.

Google App Engine is a very thorough platform with many features. This paper has discussed only a few of them. To get more information, the reader is encouraged to visit http://code.google.com/appengine to register for a free developer account, get the documentation, SDK and sample code as well as information about the new pricing model for later this year when GAE leaves beta.