# 5th Slide Set Cloud Computing

Prof. Dr. Christian Baun

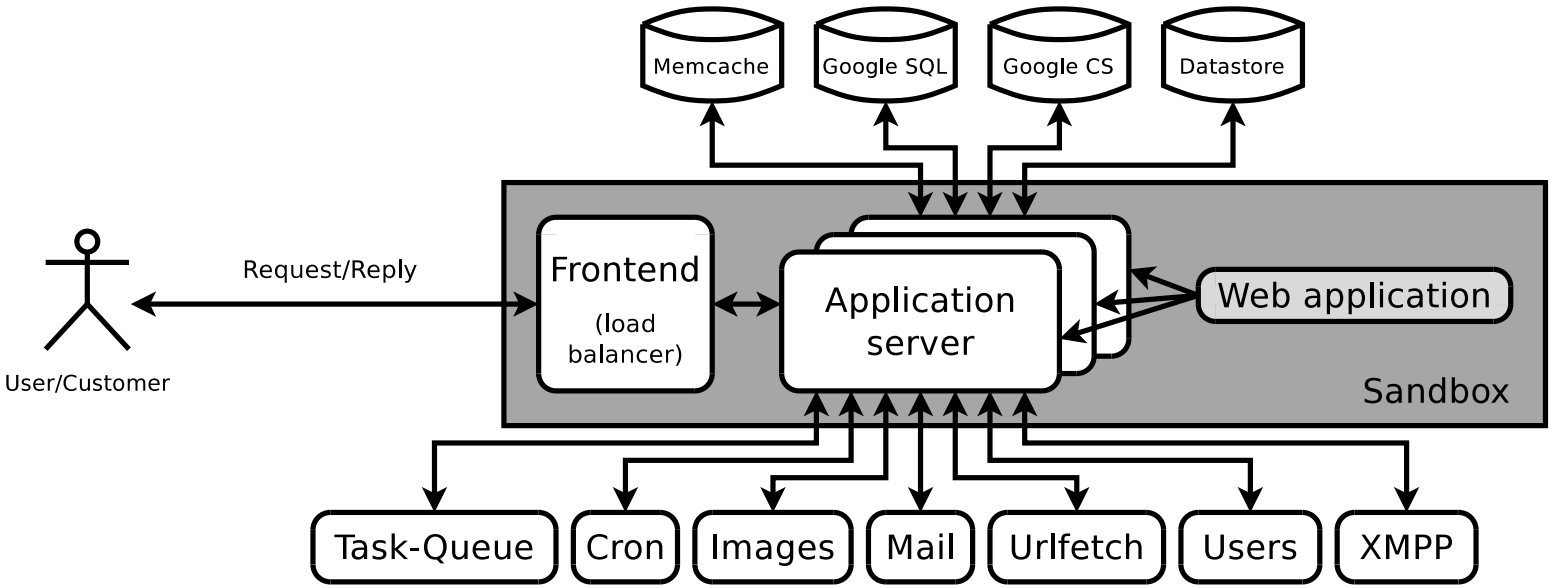
Frankfurt University of Applied Sciences
(1971–2014: Fachhochschule Frankfurt am Main)
Faculty of Computer Science and Engineering
christianbaun@fb2.fra-uas.de

# Agenda for Today

- Google App Engine
  - Google App Engine APIs
  - Required software
  - Working with the Google App Engine
  - Simple example with the Google App Engine
  - Resource limitations (Quotas)
  - Realize a guestbook with the Google App Engine

# Google App Engine (GAE)

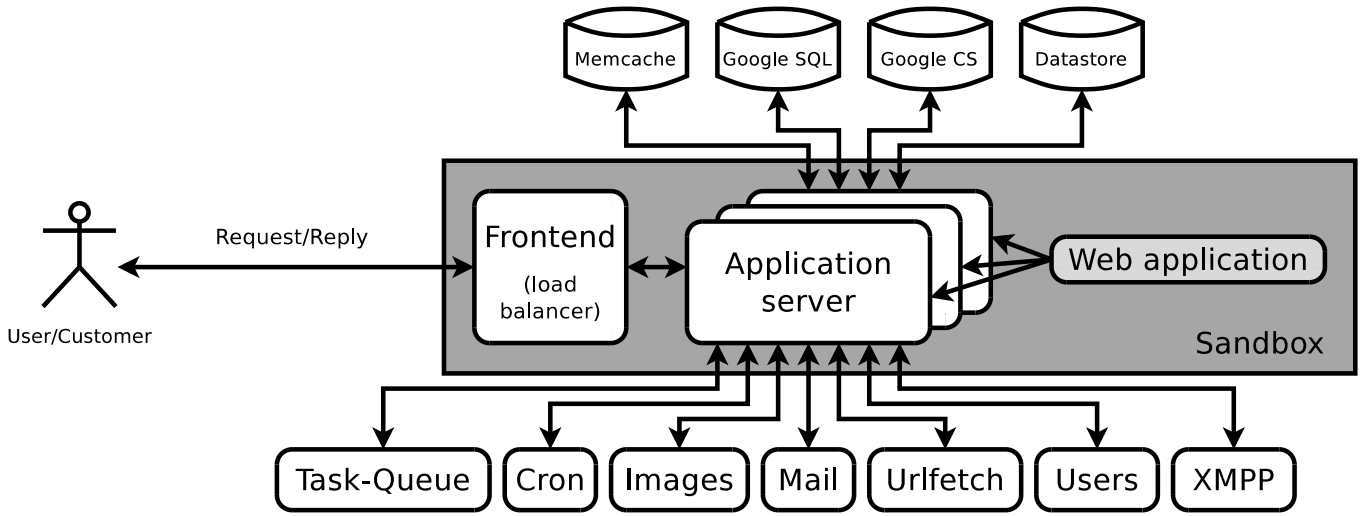
- Platform service for web applications (Python, Java, Go and PHP)
  - Available since May 2008
- Scales automatically as required
- Usage is free of charge under some of quantity limits
- Applications can use different infrastructure and storage services




- Free reimplementations: AppScale, typhoonAE (†), JBoss CapeDwarf

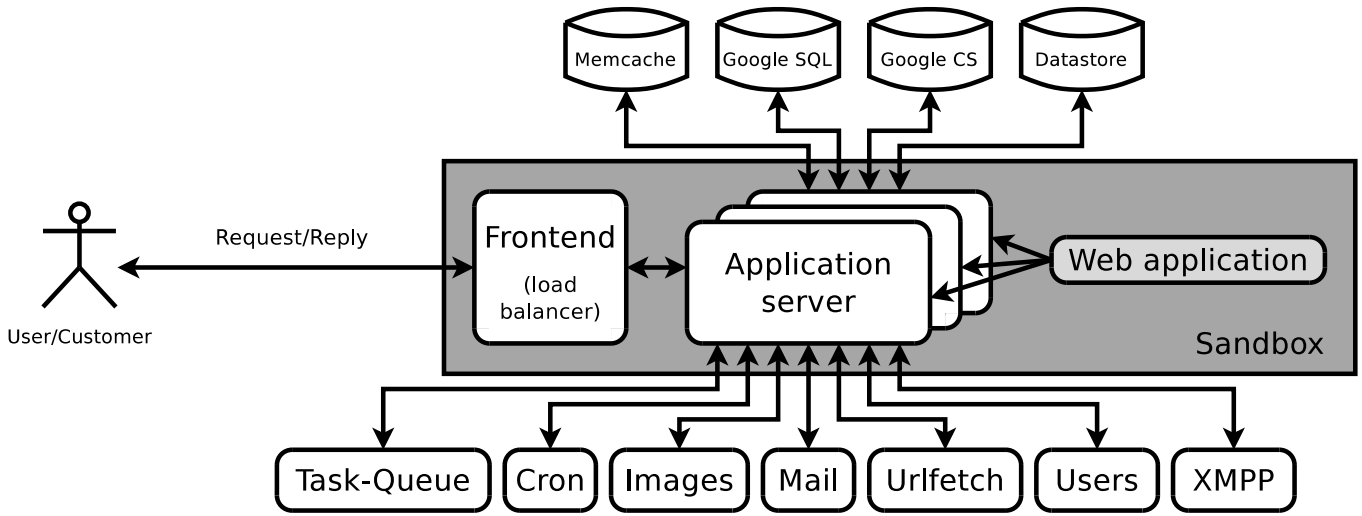
# Google App Engine APIs (1/5)

- **Authentication (Users)**
  - Authentication/Authorization is possible via Google accounts
  - Implementing an own authentication solution is not necessary
- **Datastore**
  - Persistent storage, implemented as a key/value database
  - Transactions are atomic
  - For the definition, retrieval and manipulation of data, an own language, the GQL (Google Query Language) is provided
    - GQL has similar with SQL (Structured Query Language)



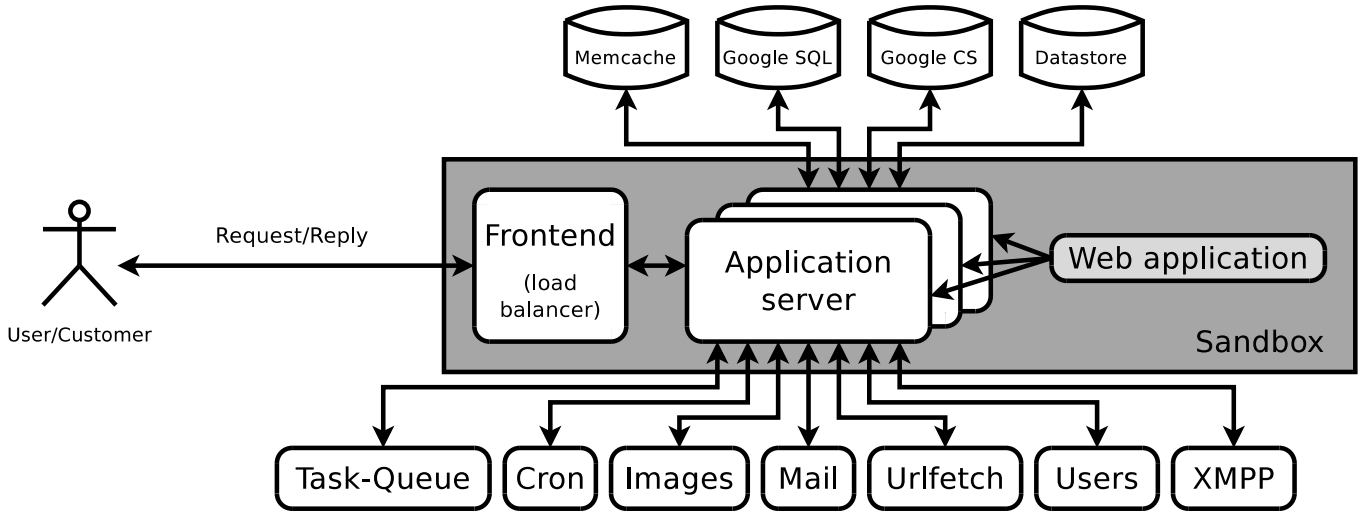
# Google App Engine APIs (2/5)

- **Memcache**
  - High-performance storage for temporary data, consisting of main memory
  - Very fast access times
  - Each record is stored with a unique key
  - For each record, a time to live in seconds is specified
    - If the time to live has expired, the record will be removed from memcache
  - Depending on the workload of the memcache, it can happen that records are erased earlier



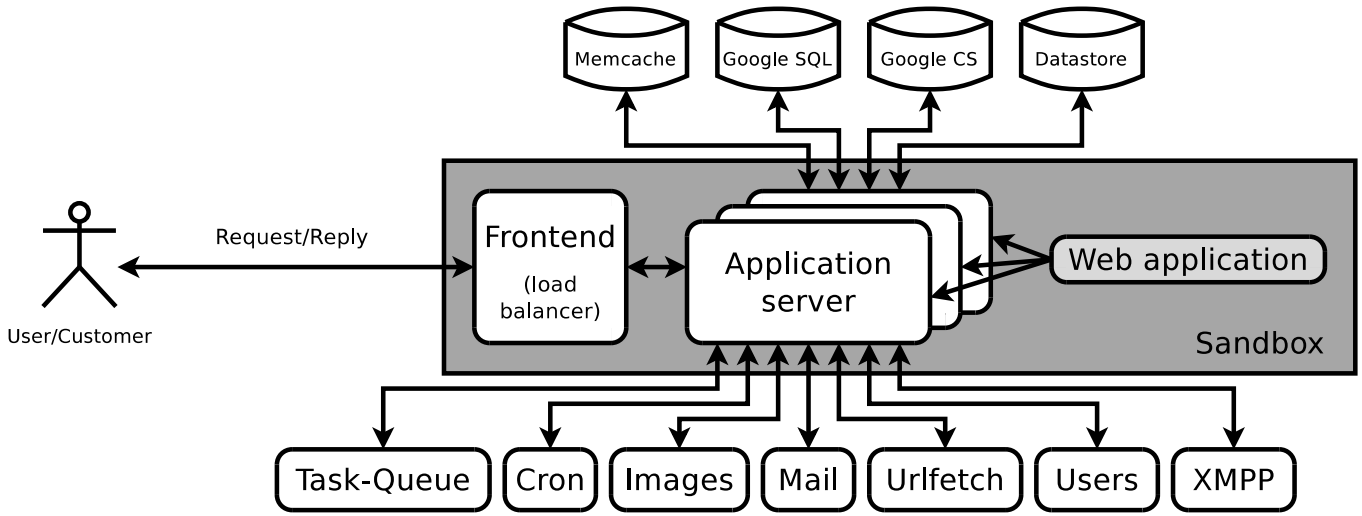
# Google App Engine APIs (3/5)

- **URL Fetch**
  - Allows to access internet content
  - Communication is possible via RESTful web services
    - Supported methods: `GET`, `POST`, `PUT`, `DELETE` and `HEAD`
- **Mail**
  - Sending and Receiving emails via the Gmail gateway
  - The sender address of the e-mail cannot be freely selected
    - Allowed sender addresses are the email address of the logged in user or the email address of the owner of the application



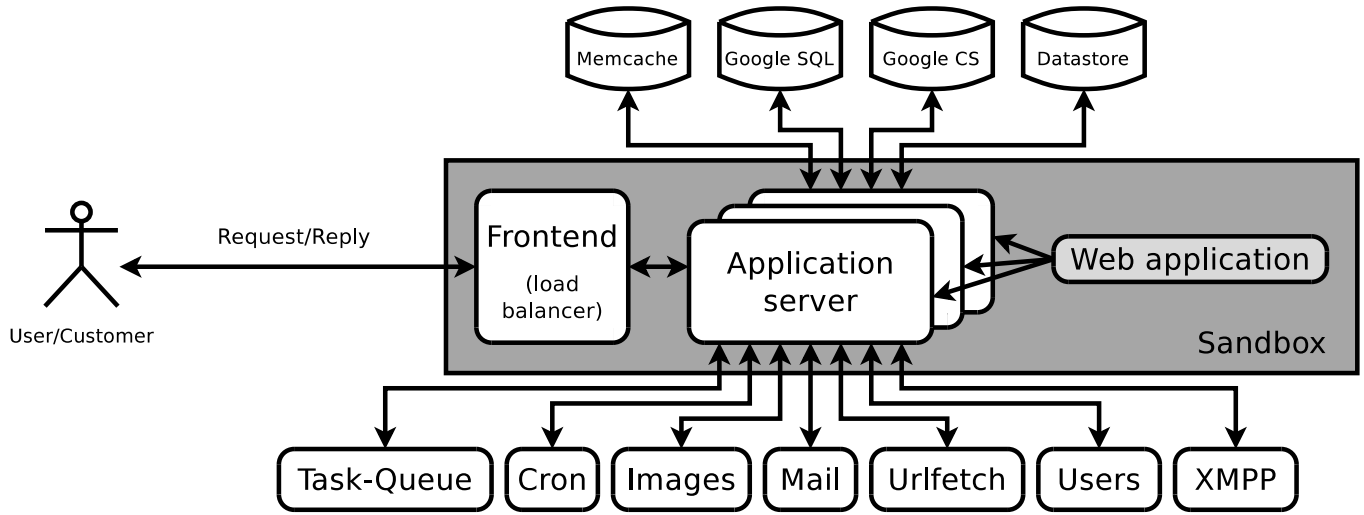
# Google App Engine APIs (4/5)

- **Google Cloud Storage**
  - Option to access/create objects inside the Google CS storage service
- **Image Manipulation**
  - Functions to rotate, flip, crop images and to change the image size
- ~~**Messaging via XMPP**~~ (service shutdown in October 2017)
  - ~~Messages can be sent to any XMPP-compatible messaging system (e.g. Google Talk) and can be received~~



# Google App Engine APIs (5/5)

- **Google Cloud SQL**
  - Is a database service for relational databases
  - Users can choose between MySQL and PostgreSQL
- **Cron**
  - The cron service allows to configure regularly scheduled tasks that operate at defined times or regular intervals
  - A cron job makes an `HTTP GET` request to a URL as scheduled



# Working with the GAE

- Applications are available online:
  `http://<application_name>.appspot.com`
- The dashboard contains all important information about the application
  `https://console.cloud.google.com/appengine`
- Multiple versions of each application can be uploaded
  - Each version can be accessed directly
    `http://<version>.latest.<application_name>.appspot.com`
  - Good for testing

# Required Software

- Browser
- Google App Engine SDK for Python (and/or JAVA/PHP/Go)
  - `https://cloud.google.com/appengine/downloads`
  - Available for Linux/UNIX, MaxOS X and Windows
- Some useful tools
  - Google Cloud Tools for Eclipse
  - PyDev extension for Eclipse (for Python applications)
    - `http://pydev.org`

# Python under Linux

- GAE supports only Python 2.7

```
$ python --version
Python 2.7.2
```

- Installation of the SDK

```
$ wget https://storage.googleapis.com/appengine-sdks/featured/
  google_appengine_1.9.63.zip
$ unzip google_appengine_1.9.63.zip
$ echo -e "export PATH=\$PATH:~/google_appengine \n" > ~/.bashrc
$ bash
$ appcfg.py -h
Usage: appcfg.py [options] <action>
...
```

The Python 2.5.2 support has expired

# Hard Limits of the GAE

- Only few programming languages are supported
- Communication with other web applications or servers is only possible via URL Fetch, ~~XMPP~~ or email
  - Only via the ports 80, 443, 4443, 8080-8089, 8188, 8444 and 8990
    - Communication via FTP or SSH is impossible
  - Maximum time, allowed for a response: 60 s
- Maximum size of. . .
  - HTTP replys: 32 MB
  - Emails including the attachment(s): 1 MB
  - ~~Incoming and outgoing XMPP messages: 100 kB~~
  - Images, which are processed by the API: 1 MB
  - Records in the datastore and memcache: 1 MB
- Read-only access to the file system

**Quotas and pricing**

`https://cloud.google.com/appengine/quotas`
`https://cloud.google.com/appengine/pricing`

# Latest Developments of the Google App Engine

- **Second Generation runtimes enable portable web apps**

  *App Engine's original release predates Google Cloud Platform and its rich set of services that support modern web apps. When App Engine first launched, GCP services like Cloud Datastore and Firebase Authentication didn't exist yet, so we built* ***App Engine APIs for common web app functionality, like storage, caching and queueing. This helped customers write apps with minimal setup, but it also led to reduced code portability****.*

  *The new Python 3.7 runtime supports the Google Cloud client libraries so you can integrate GCP services into your app, and run it on App Engine, Compute Engine or any other platform. At Google Cloud Next we also announced Cloud Scheduler and Cloud Tasks—services that mirror App Engine's popular scheduling and queueing functionality (App Engine cron and Task Queues). We are progressively evolving the original App Engine APIs to make them accessible across all GCP platforms.* ***At this time, the original App Engine-only APIs are not available in Second Generation runtimes, including Python 3.7.***

**Source**

`https://cloud.google.com/blog/products/gcp/introducing-app-engine-second-generation-runtimes-and-python-3-7`
Date: August 8th, 2018

# An Attempt to look to the Future

- Situation now:
  - The 1st generation of the App Engine is the perfect tool to do a rapid development of web applications...
    - by the cost of being limited to using the set proprietary API and a whitlist of third-party libraries.
  - Python 2.7 will not be maintained past 2020
    - `https://pythonclock.org`
    - It is unlikely that Google will provide the 1st generation of the App Engine with Python 2 forever
  - The 2nd generation of the App Engine has fewer restrictions on the runtime environment...
    - but no App Engine APIs
    - `https://cloud.google.com/appengine/docs/standard/python3/python-differences`
  - The migration of Python 2 source code to Python 3 is not simple
    - Better don't start new GAE projects with Python 2 now

# GAE Example – Guestbook

- Objective: Develop a guestbook
- Main features:
  - Authentication information of Google are used
  - Data is stored in the Datastore
  - Records can be erased
  - Web pages are created using the webapp framework

This example uses Python 2.7 and closely follows this tutorial:
**Developing and deploying an application on Google App Engine**
`http://www.youtube.com/watch?gl=DE&hl=de&v=bfg0-LXGpTM`

# Configuration File – app.yaml

- For each application, a configuration file app.yaml must exist

```
application: cloud-vorlesung
version: 1
api_version: 1
runtime: python27
threadsafe: no

handlers:
- url: .*
  script: main.py
```

- application: Application name
- version: Version of the application
- runtime: Runtime environment
  - python27 $\Longrightarrow$ Python 2.7
- api_version: API-Version of the GAE
  - Version 1 is the latest one
- handlers: URLs mapped to scripts
  - In this example, all requests to the Python script are forwarded to main.py

# Mini Application

- For each application, a directory with the equal name should exist locally

```
$ mkdir cloud-vorlesung
$ cd cloud-vorlesung
```

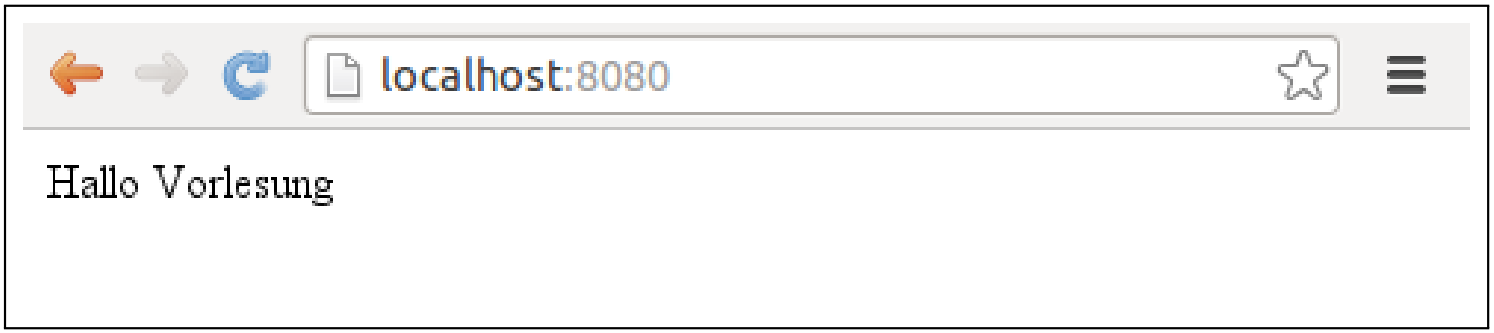
- Create the mini application in `main.py`

```
1 #!/usr/bin/env python
2
3 print "Hallo Vorlesung"
```

- Start with the mini application with the development server

```
$ dev_appserver.py cloud-vorlesung/
INFO    2013-10-05 14:08:48,702 sdk_update_checker.py:245] Checking for updates to the SDK.
INFO    2013-10-05 14:08:49,113 api_server.py:138] Starting API server at: http://localhost:41116
INFO    2013-10-05 14:08:49,296 dispatcher.py:164] Starting module "default" running at: http://localhost:8080
INFO    2013-10-05 14:08:49,297 admin_server.py:117] Starting admin server at: http://localhost:8000
```

# Test Mini Application




- The development server returns information about what happened

```
$ dev_appserver.py cloud-vorlesung/
INFO     2013-10-05 14:16:08,991 sdk_update_checker.py:245] Checking for updates to the SDK.
INFO     2013-10-05 14:16:09,545 api_server.py:138] Starting API server at: http://localhost:33666
INFO     2013-10-05 14:16:09,636 dispatcher.py:164] Starting module "default" running at: http://localhost:8080
INFO     2013-10-05 14:16:09,638 admin_server.py:117] Starting admin server at: http://localhost:8000
INFO     2013-10-05 14:16:12,391 module.py:593] default: "GET / HTTP/1.1" 200 16
INFO     2013-10-05 14:16:12,785 module.py:593] default: "GET /favicon.ico HTTP/1.1" 200 16
```

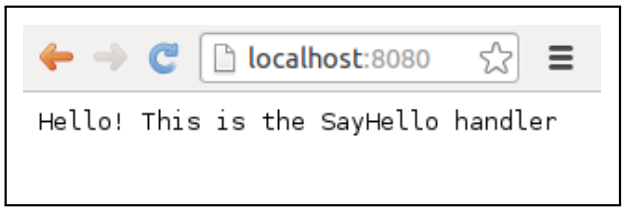
- There was a successful request $\Longrightarrow$ HTTP status code 200

# Simple Output with a better Source Code Basis

## app.yaml

```
application: cloud-vorlesung
version: 2
api_version: 1
runtime: python27
threadsafe: no

handlers:
- url: .*
  script: main.py

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

```
$ dev_appserver.py cloud-vorlesung/
```




## main.py

```
1  #!/usr/bin/env python
2  import os, sys
3  import wsgiref.handlers
4  import webapp2
5  from google.appengine.ext.webapp.util import run_wsgi_app
6
7  class SayHello(webapp2.RequestHandler):
8    # Method to process HTTP GET requests
9    def get(self):
10     self.response.headers['Content-Type'] = 'text/plain'
11     self.response.out.write('Hello! This is the SayHello handler')
12
13 # When the application receives a request, it tries to find the
14 # matching entry and then calls the corresponding handler
15 app = webapp2.WSGIApplication([('/', SayHello)],
16                               debug=True)
17
18 def main():
19   # Runs a WSGI application in App Engine's CGI environment
20   run_wsgi_app(app)
21
22 # Program can act as either reusable module, or standalone program
23 if __name__ == "__main__":
24   main()
```

- A handler processes the HTTP GET requests
  - The HTTP GET requests are assigned to a class

# Guestbook – main.py (1/2)

```
1  #!/usr/bin/env python
2  import os, sys
3  import wsgiref.handlers
4  import webapp2
5  from google.appengine.ext import db
6  from google.appengine.ext.webapp.util import run_wsgi_app
7  from google.appengine.ext.webapp import template
8
9  class guestbook(db.Model):                                    # Name of the datastore: guestbook
10   message = db.StringProperty(required=True)                  # Define the structure of the datastore
11   when = db.DateTimeProperty(auto_now_add=True)
12   who = db.StringProperty()
13
14 class ShowGuestbookPage(webapp2.RequestHandler):
15   def get(self):                                              # Method to process HTTP GET requests
16     shouts = db.GqlQuery('SELECT * FROM guestbook ORDER BY when DESC') # Request for the Datastore
17     values = {'shouts': shouts}
18     self.response.out.write(template.render('main.html', values))  # Forward the data to main.html
19
20 class MakeGuestbookEntry(webapp2.RequestHandler):
21   def post(self):                                             # Method to process HTTP POST requests
22     shout = guestbook(message=self.request.get('message'),    # Grab the data from the HTML form
23                       who=self.request.get('who'))
24     shout.put()                                               # Write into the datastore
25     self.redirect('/')                                        # Redirect to URI /
```

# Guestbook – main.py (2/2)

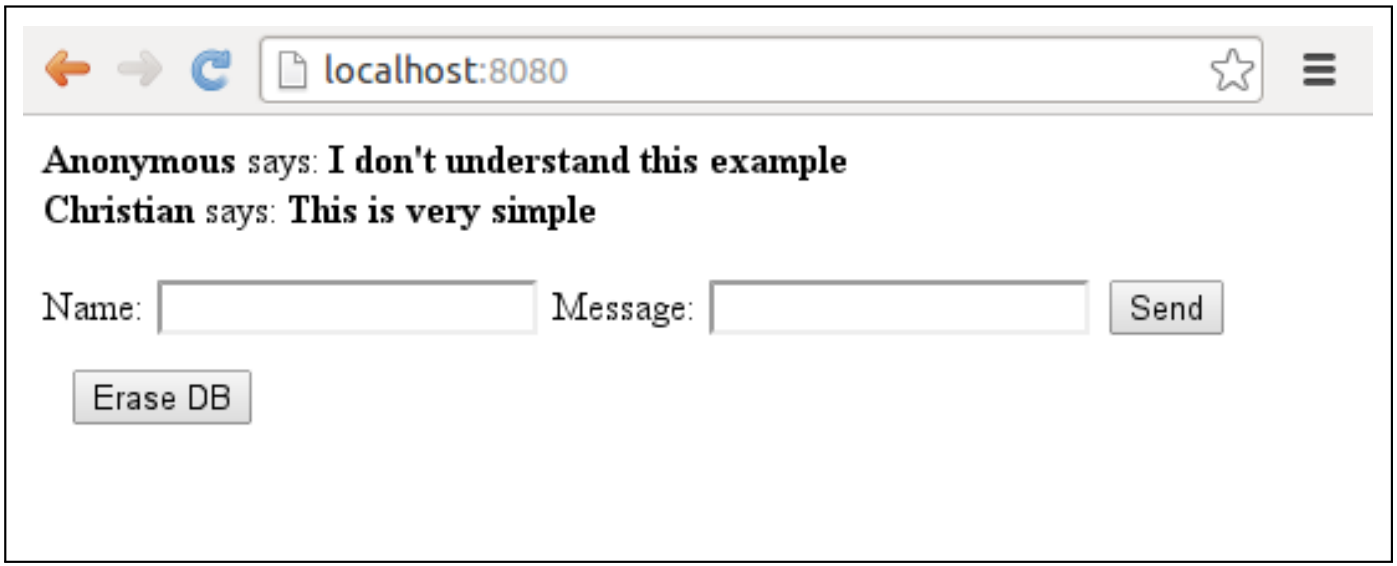
```
26 class EraseAllEntries(webapp2.RequestHandler):
27   def post(self):                                    # Method to process HTTP POST requests
28     erase_all_query = guestbook.all(keys_only=True)
29     db.delete(erase_all_query.fetch(limit=None))     # Erase all entries in guestbook
30     self.redirect('/')                               # Redirect to URI /
31
32 # When the application receives a URI request, it tries to find the
33 # matching entry and then calls the corresponding handler
34 app = webapp2.WSGIApplication([('/', ShowGuestbookPage),
35                                ('/make_entry', MakeGuestbookEntry),
36                                ('/erase_all', EraseAllEntries)],
37                               debug=True)
38
39 def main():
40   # Runs a WSGI application in App Engine's CGI environment
41   run_wsgi_app(app)
42
43 # Program can act as either reusable module, or standalone program
44 if __name__ == "__main__":
45   main()
```

# Guestbook – main.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>Simple Guestbook with the Google App Engine</title></head>
<body>

{% for shout in shouts %}
<div>
    {% if shout.who %}
        <b>{{shout.who}}</b>
    {% else %}
        <b>Anonymous</b>
    {% endif %}
    says:
    <b>{{shout.message}}</b>
</div>
{% endfor %}

<p>

<form action="make_entry" method="post" accept-charset="utf-8">
Name: <input type="text" size="20" name="who" value="" if="who">
Message: <input type="text" size="20" name="message" value="" if="message">
<input type="submit" value="Send"></form>

<form action="erase_all" method="post" accept-charset="utf-8">
<table border="0" cellspacing="5" cellpadding="5">
  <tr><td align="center"><input type="submit" value="Erase DB"></td></tr>
</table></form>

</body></html>
```

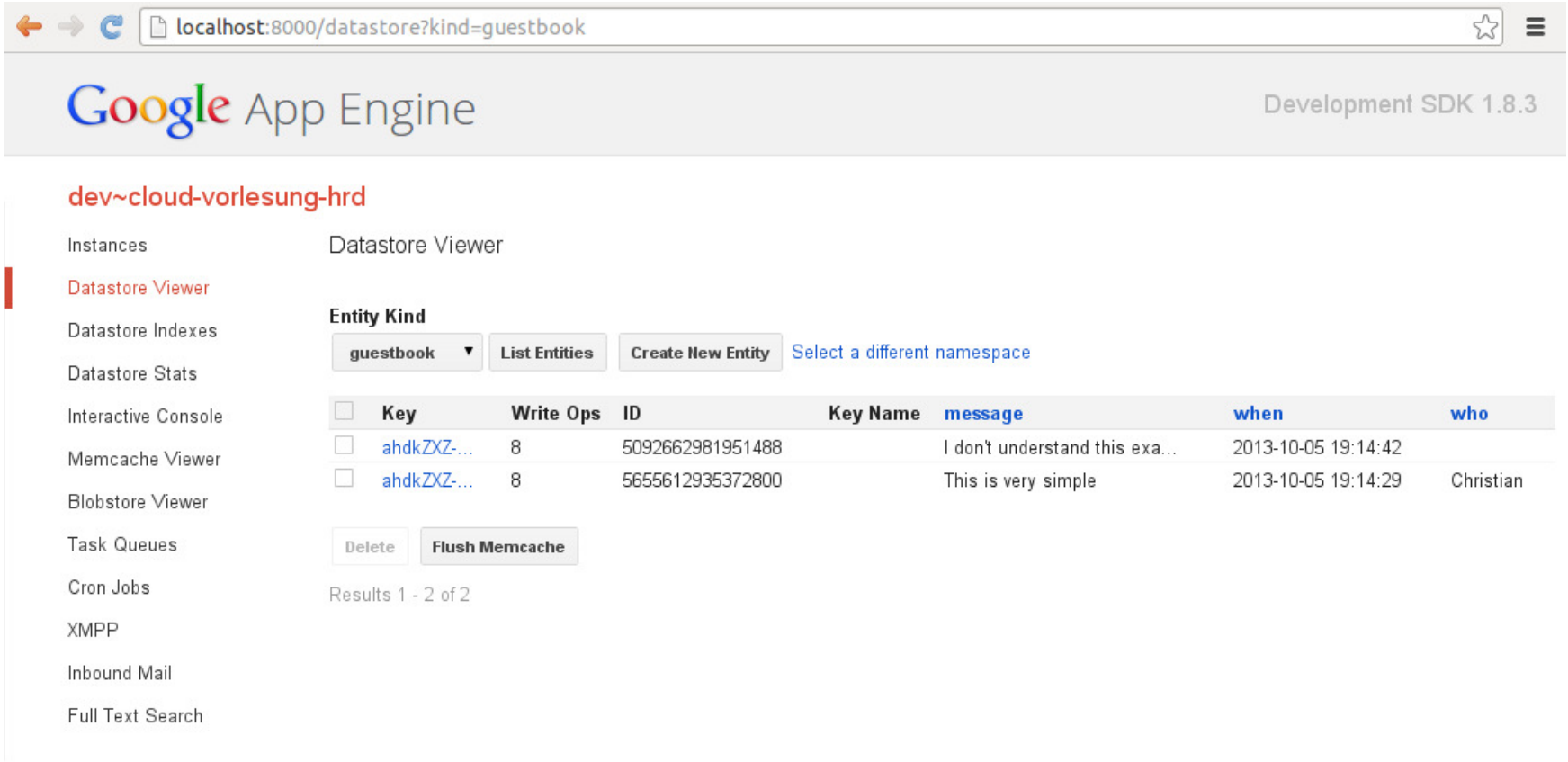
# Result


localhost:8080
Name:
Message:
Send
Erase DB


localhost:8080
Anonymous says: I don't understand this example
Christian says: This is very simple
Name:
Message:
Send
Erase DB

# Helpful Tool: Development Console

- Part of the development server
  - Available at `http://localhost:8000`




The datastore is stored locally at `/tmp/appengine.<application_name>.<user>/datastore.db`

# Upload Application and check Log Data

- Application upload via `appcfg.py` from the SDK

```
$ appcfg.py --email=wolkenrechnen@gmail.com update cloud-vorlesung/
```

- Get the log data via `appcfg.py`

```
$ appcfg.py --email=wolkenrechnen@gmail.com request_logs cloud-vorlesung/ logs.txt
```

```
$ cat logs.txt
87.178.89.156 - - [29/Apr/2010:06:01:20 -0700] "GET / HTTP/1.1" 200 540 - "Mozilla/5.0
(X11; U; Linux i686; de; rv:1.9.0.6) Gecko/2009020409 Iceweasel/3.0.14 (Debian-3.0.14-1),gzip(gfe)"
87.178.89.156 - - [29/Apr/2010:06:01:20 -0700] "GET /favicon.ico HTTP/1.1" 404 124 - "Mozilla/5.0
(X11; U; Linux i686; de; rv:1.9.0.6) Gecko/2009020409 Iceweasel/3.0.14 (Debian-3.0.14-1),gzip(gfe)"
```

- IP address of the client
- Date of the request
- Time of the request
- HTTP request
- HTTP status code (server response)
- Size of the response in bytes
- Client information
- ...