



THE UNIVERSITY OF
CHICAGO



MPCS 51033 • AUTUMN 2017 • SESSION 2

BACKENDS FOR MOBILE APPLICATIONS

CLASS NEWS

CLASS NEWS

- Office hours
Thursday 10AM in
Young 308
- Threading in
Slack

The screenshot shows the Slack interface. On the left is the sidebar with channels like All Threads, Channels, Direct Messages, and Apps. The main area shows the #general channel with messages from Andrew and others. A specific message from Andrew on September 28th links to a Google Cloud Platform blog post about the new code editor. A red circle highlights the reply button in the message input field. To the right, a direct message thread between Andrew and Yi Wang is shown, with Yi Wang replying that it worked with their Gmail.

#general

Andrew 10:37 AM added an integration to this channel: [twitter](#)

Andrew 9:09 PM Hi. I'm not going to have any formal office hours tomorrow but let me know if you have any questions. I plan on having regular office hours Thursday around 10 starting next week.

Thursday, September 28th

Andrew 10:58 AM <https://cloudplatform.googleblog.com/2016/10/introducing-Google-Cloud-Shells-new-code-editor.html?m=1>

Google Cloud Platform Blog [Introducing Google Cloud Shell's new code editor](#)
Posted by Sachin Kotwani, Product Manager We've heard from a lot of Google Cloud Platform (GCP) users that they like to edit code and c... (9kB)

Here's an example of how you can use the Cloud Shell code editor to create a sample app, push your changes to Google Cloud Source Repository, deploy the app to Google App Engine Standard, and use Stackdriver Debugger.

Andrew 11:15 AM

Message #general @ 😊

Thread

Yi Wang and you

Yi Wang Today at 10:56 AM Direct message

I received the coupon email in my uchicago email box, but when I clicked the url to redeem, it redirected to a page that is logged in with my gmail. I didn't notice that and finished the whole process.

2 replies

Andrew 6 hours ago If it worked with your gmail, then we can go ahead and try it. I'll take a look in class before we actually start using it to make sure it looks ok. I will send an email to my contact at google just to double check.

Yi Wang 5 hours ago OK. Thank you!

Reply... 😊

MOBILE APP BACKEND SERVICES WITH GCP

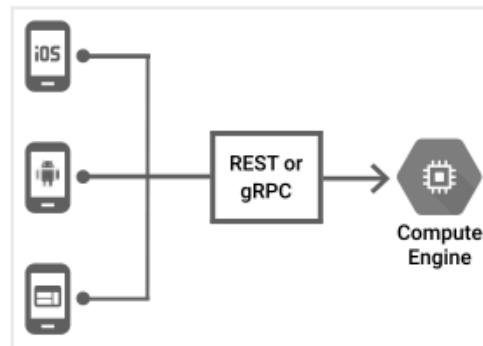
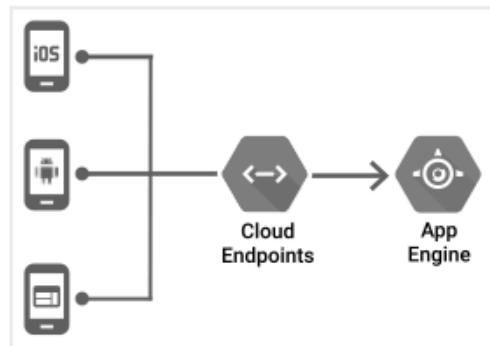
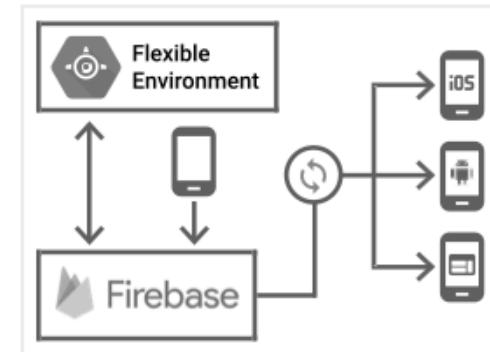
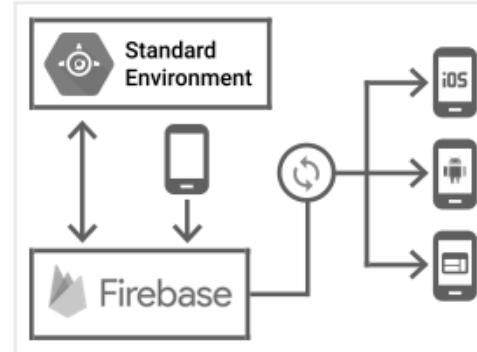
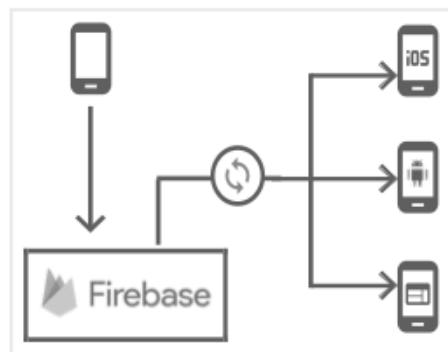
BACKEND SERVICES WITH GCP

- Apple's trend is to do almost everything on device. Why do we need a backend?
- Most mobile apps and games need a backend service for things that can't be done solely on-device
 - Sharing and processing data from multiple users
 - Storing large files
 - Group analytics statistics (leaderboard)

BACKEND SERVICES WITH GCP

- Building a backend service for a mobile app is similar to building a web-based service, with some additional considerations
 - Limit on-device data storage
 - Synchronize data across multiple devices
 - Handle the online/offline transition gracefully
 - Send notifications and messages
 - Minimize battery drain (be a good citizen)

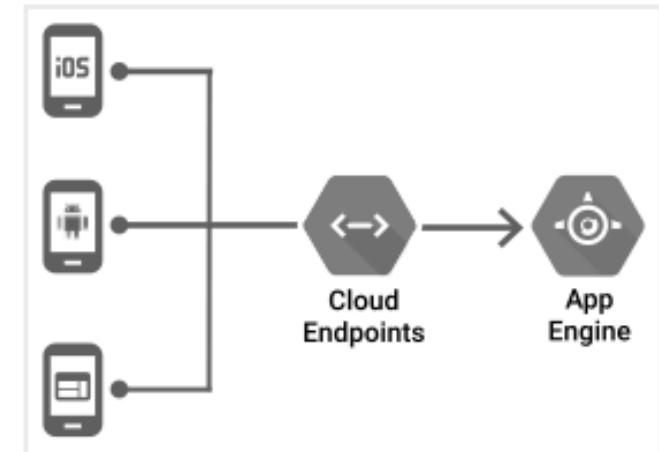
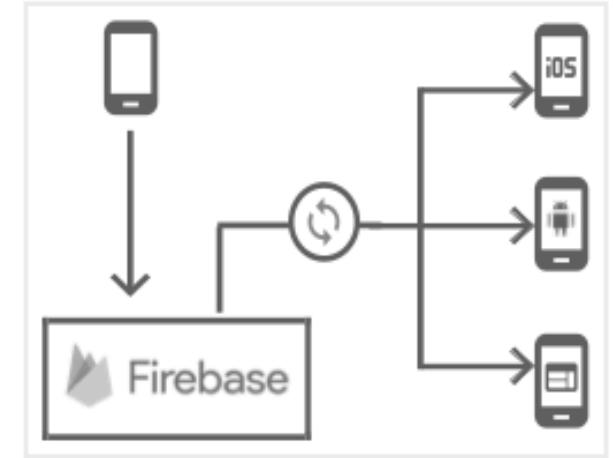
BACKEND SERVICES WITH GCP



DESIGN PATTERNS USING GCP FOR
BACKENDS SERVICES

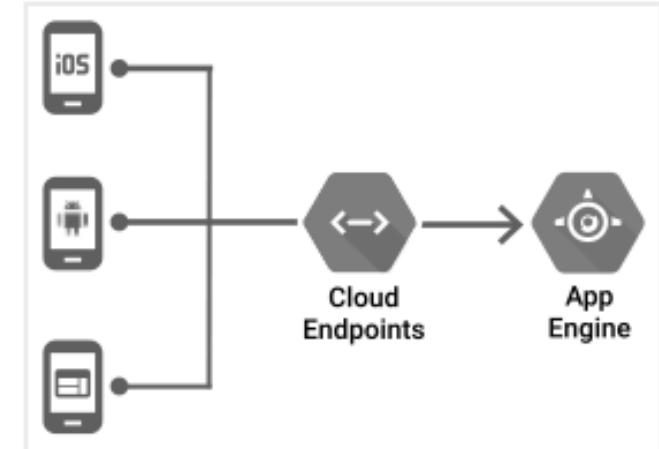
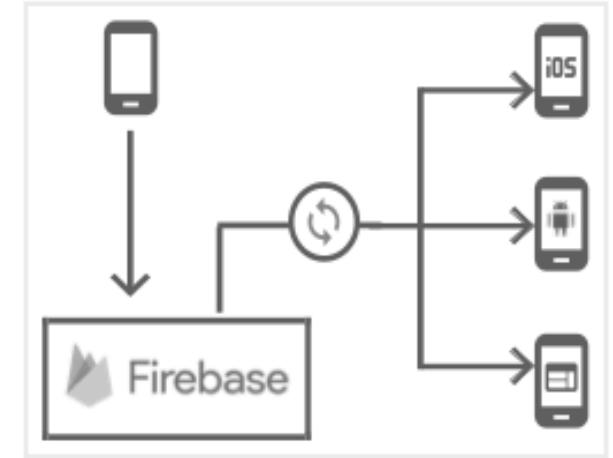
BACKEND SERVICES WITH GCP

- Two-tier architecture with Firebase
 - Mobile app and Firebase manipulate data directly
 - Security and data validation handled in console

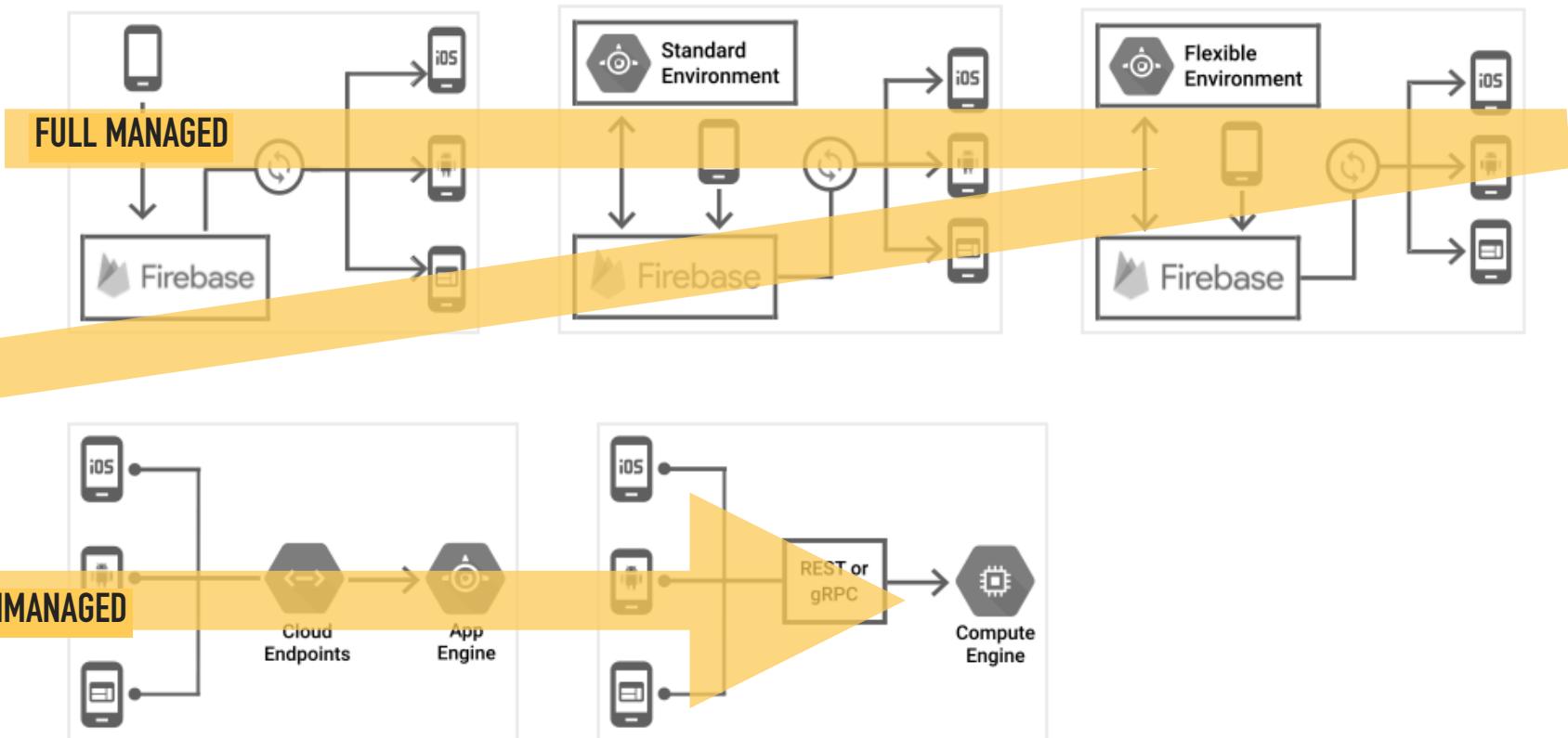


BACKEND SERVICES WITH GCP

- Three-tier architecture with App Engine
 - Communication layer between app and backend
 - Responsible for authentication and data-validation

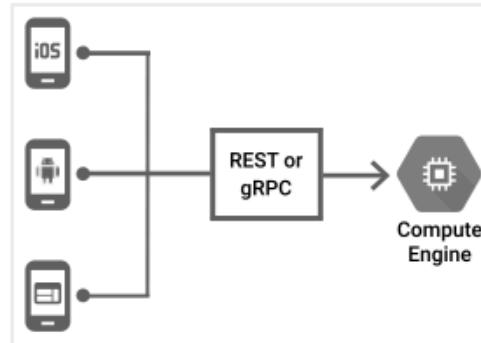
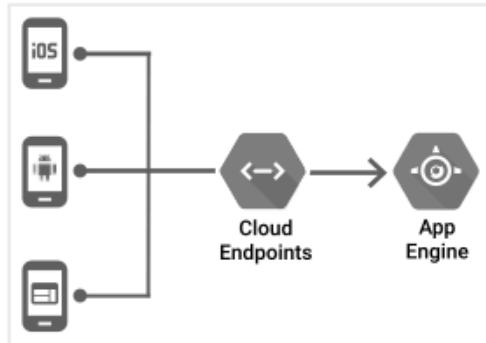
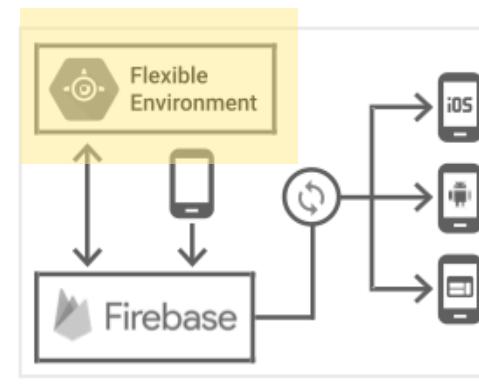
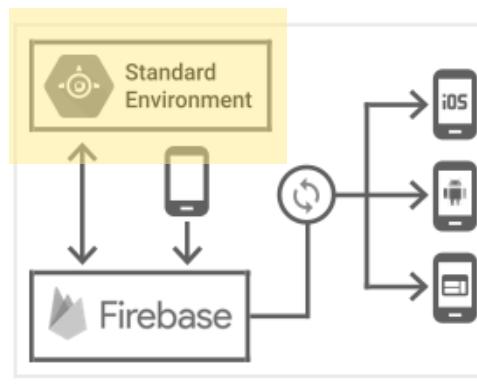
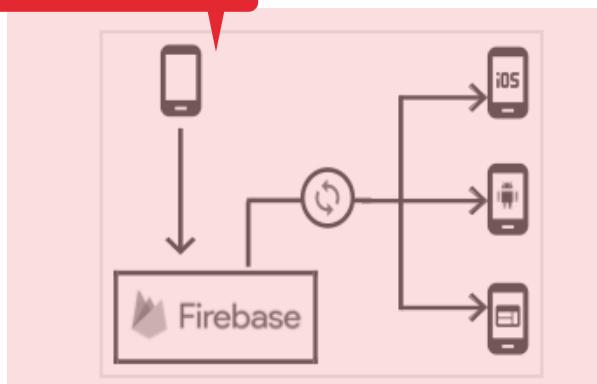


BACKEND SERVICES WITH GCP



BACKEND SERVICES WITH GCP

LATER IN THE COURSE



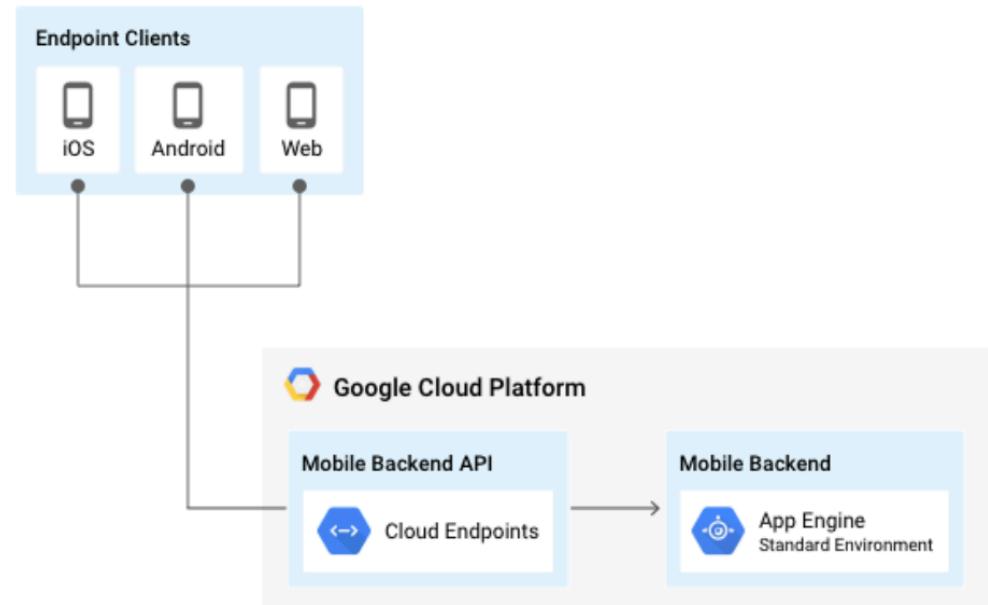
HYBRID PROVIDES MIX OF
EASE OF USE AND
FLEXIBILITY

CLOUD ENDPOINTS

BACKEND SERVICES WITH GCP

CLOUD ENDPOINTS

- Google Cloud Endpoints generates APIs, client libraries, and discovery documentation for an App Engine application
- You don't write wrappers to handle communication with App Engine
- Client libraries are generated by Cloud Endpoints to make direct API calls from your mobile app



BACKEND SERVICES WITH GCP

CLOUD ENDPOINTS

- Google Cloud Endpoints generates APIs, client libraries, and discovery documentation for an App Engine application
- You don't write wrappers to handle communication with App Engine
- Client libraries are generated by Cloud Endpoints to make direct API calls from your mobile app

Cloud Endpoints

About Cloud Endpoints Frameworks

Contents

Basic frameworks architecture

Libraries and tools

Using NDB Datastore with the frameworks

Requirements

Development process

Getting Started

Migrating from Endpoints 1.0

Google Cloud Endpoints Frameworks for App Engine consists of tools, libraries and capabilities that allow you to generate APIs and client libraries from an App Engine application. Referred to as an *API*, they simplify client access to data from other applications, such as web clients and Android mobile clients.

 **Note:** Cloud Endpoints Frameworks supports only the App Engine standard environment; the App Engine flexible environment is not supported.

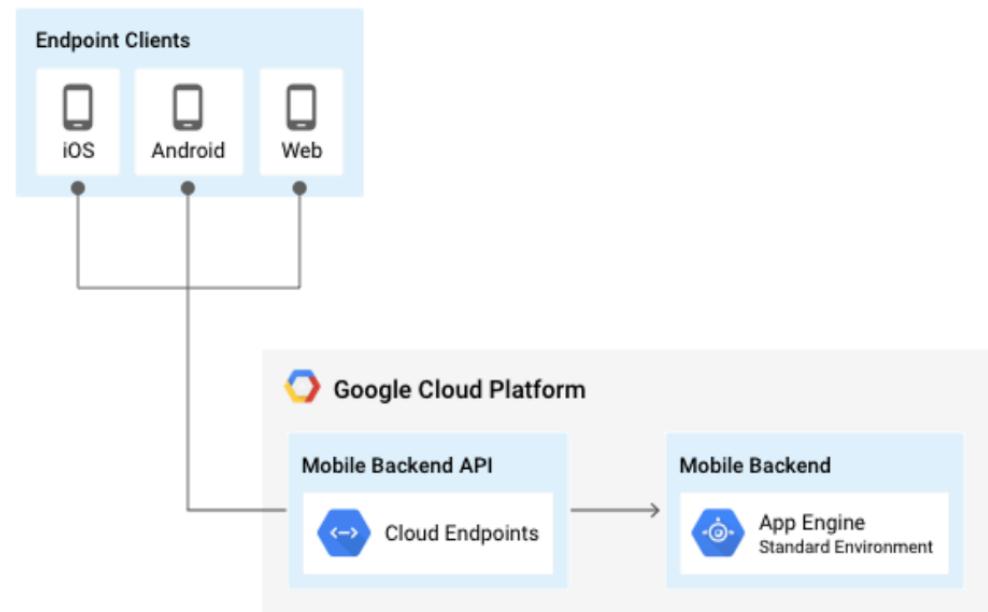
Cloud Endpoints Frameworks supports the following API management features provided by Cloud Endpoints:

- [API key management](#)
- [API sharing](#)
- [user authentication](#)
- View API [metrics](#)
- View API [logs](#)

BACKEND SERVICES WITH GCP

CLOUD ENDPOINTS

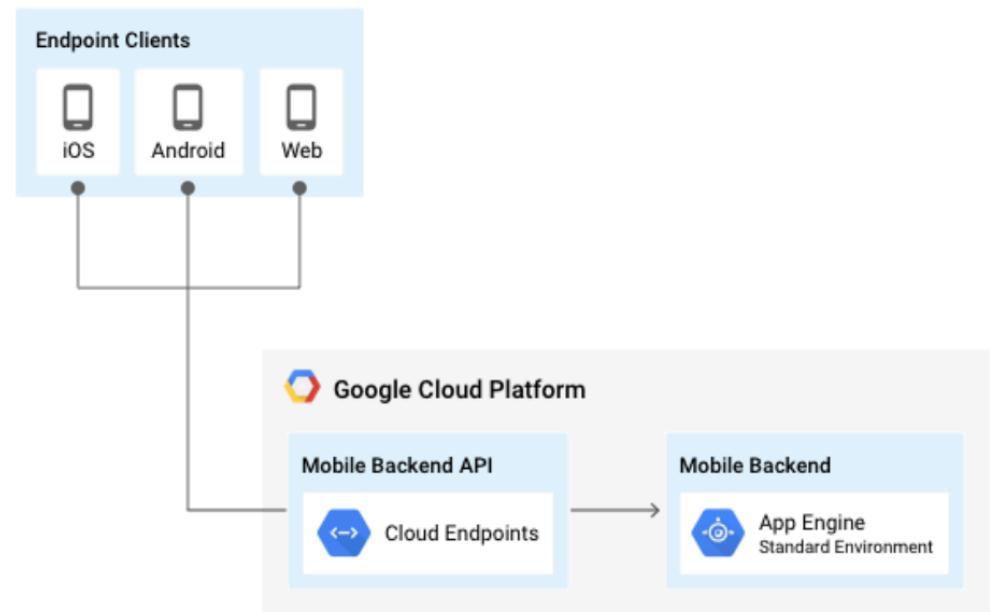
- Recommended use
 - Automated generation of client libraries that apps can use to call the backend service directly (ie ease of use)
 - Reducing on-device storage by moving files to Cloud Storage
 - Sending notifications by calling Cloud Messaging



BACKEND SERVICES WITH GCP

CLOUD ENDPOINTS

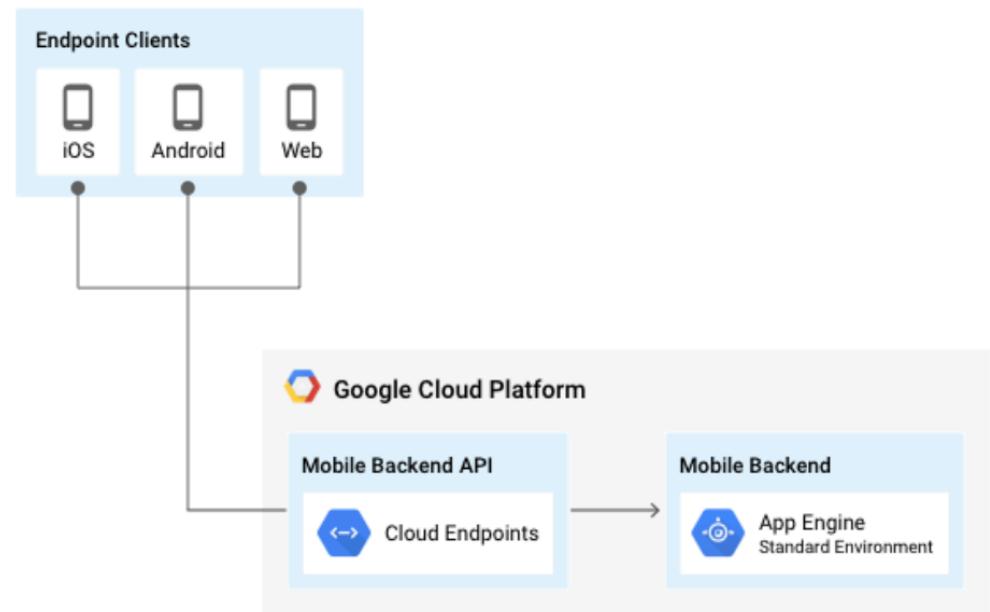
- Not recommended for
 - Apps that require automatic real-time data synchronization across devices
 - Backend services that require custom server or third party libraries.
 - Systems that do not support SSL; SSL is required by Cloud Endpoints.



BACKEND SERVICES WITH GCP

CLOUD ENDPOINTS

- Not recommended for
 - Apps that require automatic real-time data synchronization across devices
 - Backend services that require custom server or third party libraries.
 - Systems that do not support SSL; SSL is required by Cloud Endpoints.

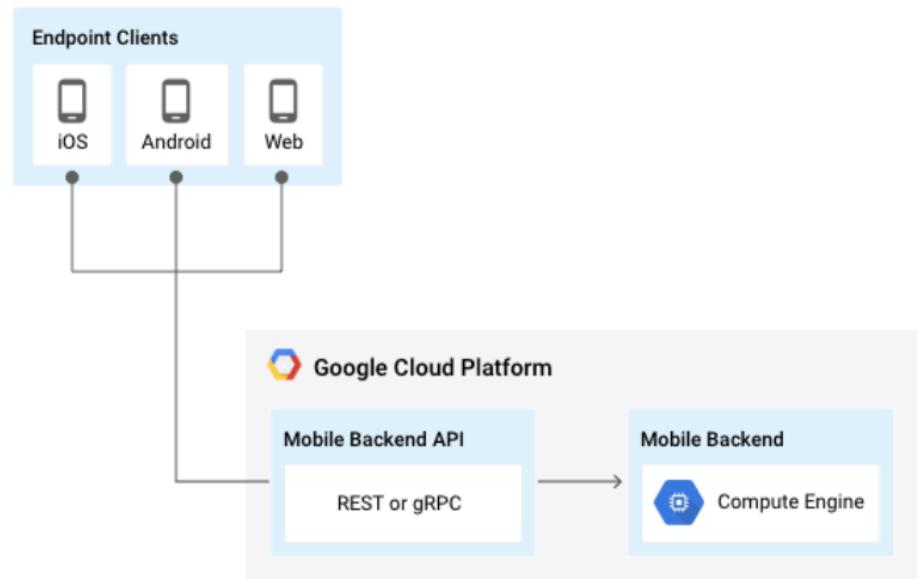


COMPUTE ENGINE

BACKEND SERVICES WITH GCP

COMPUTE ENGINE

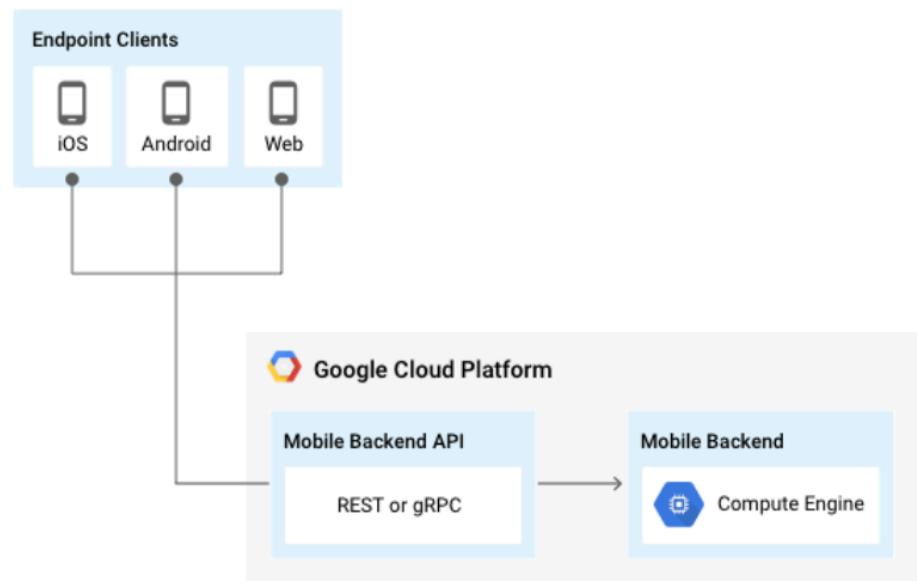
- Computer Engine lets you create and run virtual machines on Google infrastructure
- You have administrator rights to the server and full control over its configuration
 - You are responsible for updates and maintenance :(



BACKEND SERVICES WITH GCP

COMPUTE ENGINE

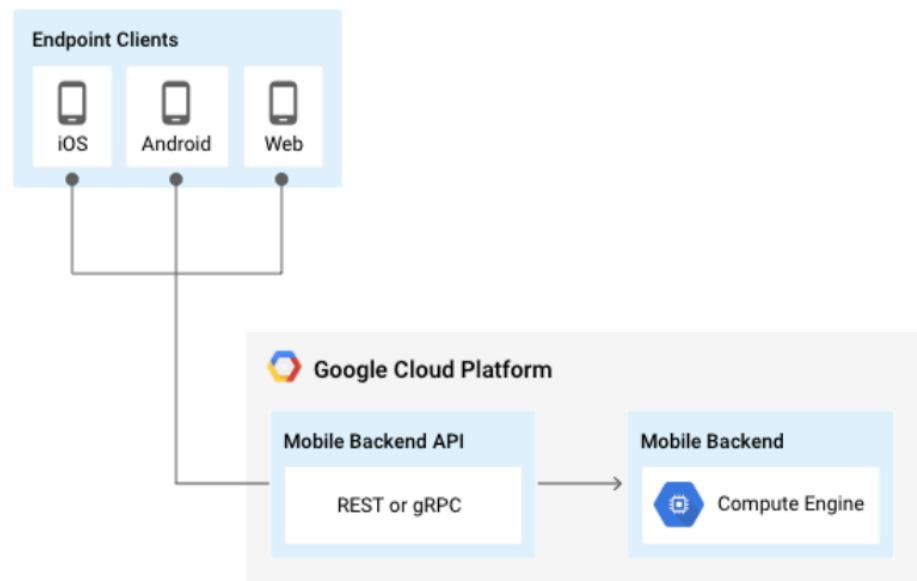
- Recommended for:
 - Porting an existing backend service running on an on-premise server or a virtual machine.
 - Backend services that require a custom server or third-party libraries.



BACKEND SERVICES WITH GCP

COMPUTE ENGINE

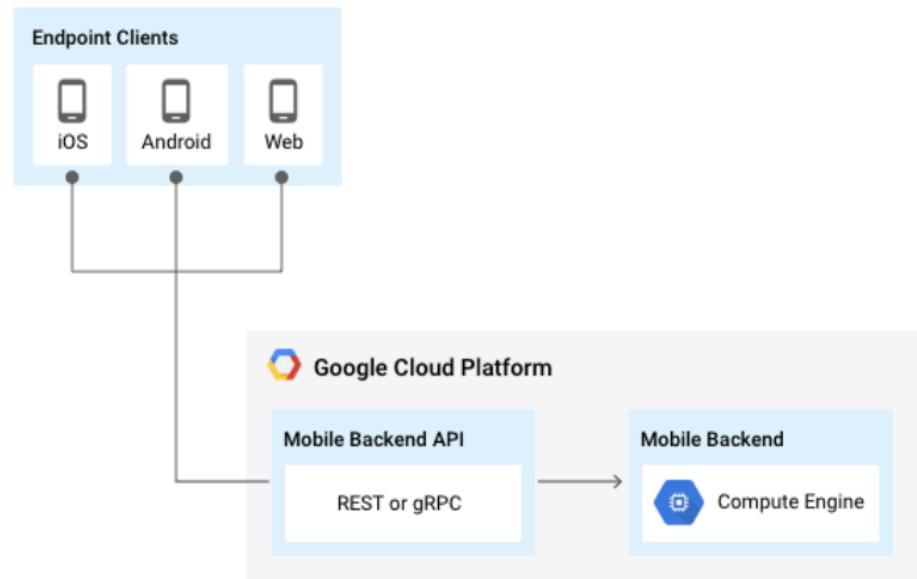
- Not recommended for:
 - Apps that require automatic real-time data synchronization across devices.
 - Automatic maintenance; you must maintain and upgrade the server yourself.
 - Automatic scaling; you must manually configure and manage an autoscaler.



BACKEND SERVICES WITH GCP

COMPUTE ENGINE

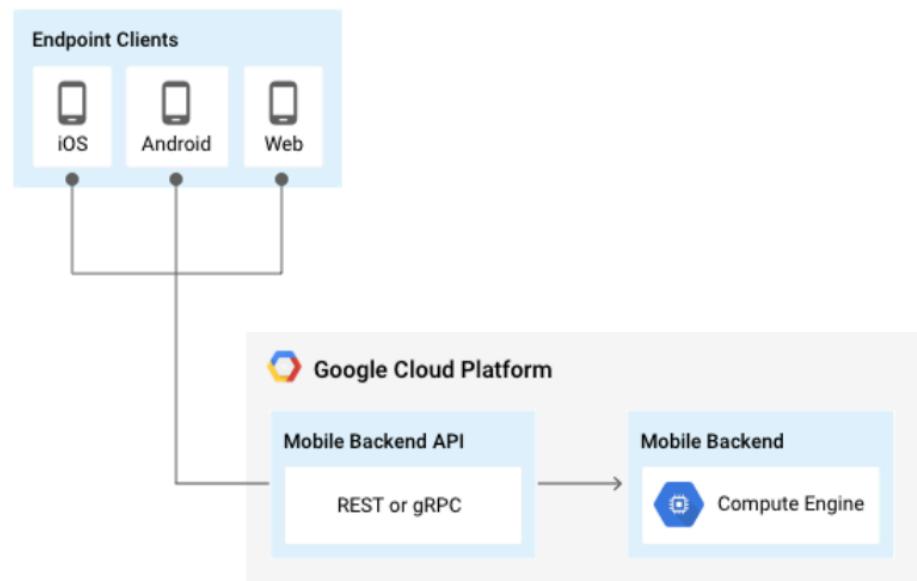
- Protocols used to connect to compute engine
 - REST - architecture based on HTTP
 - gRPC - framework using http/2



APP ENGINE

BACKEND SERVICES WITH GCP

- Everything that follows



APP WALK THROUGH: PHOTO TIMELINE

PHOTO TIMELINE

- Backend for a photo timeline application
- App behaviors
 - Users post pictures with a comment
 - Users retrieve pictures

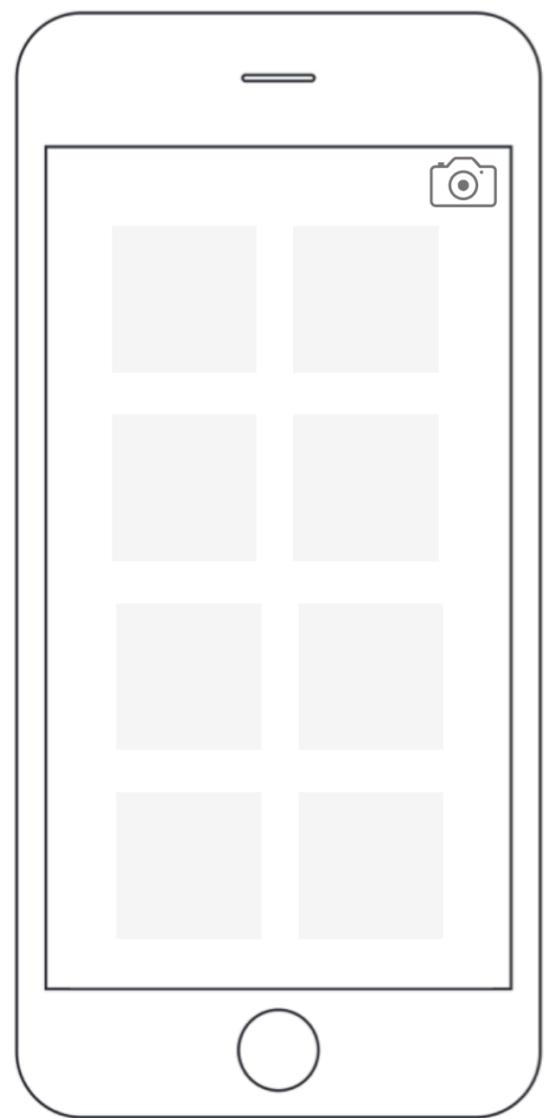
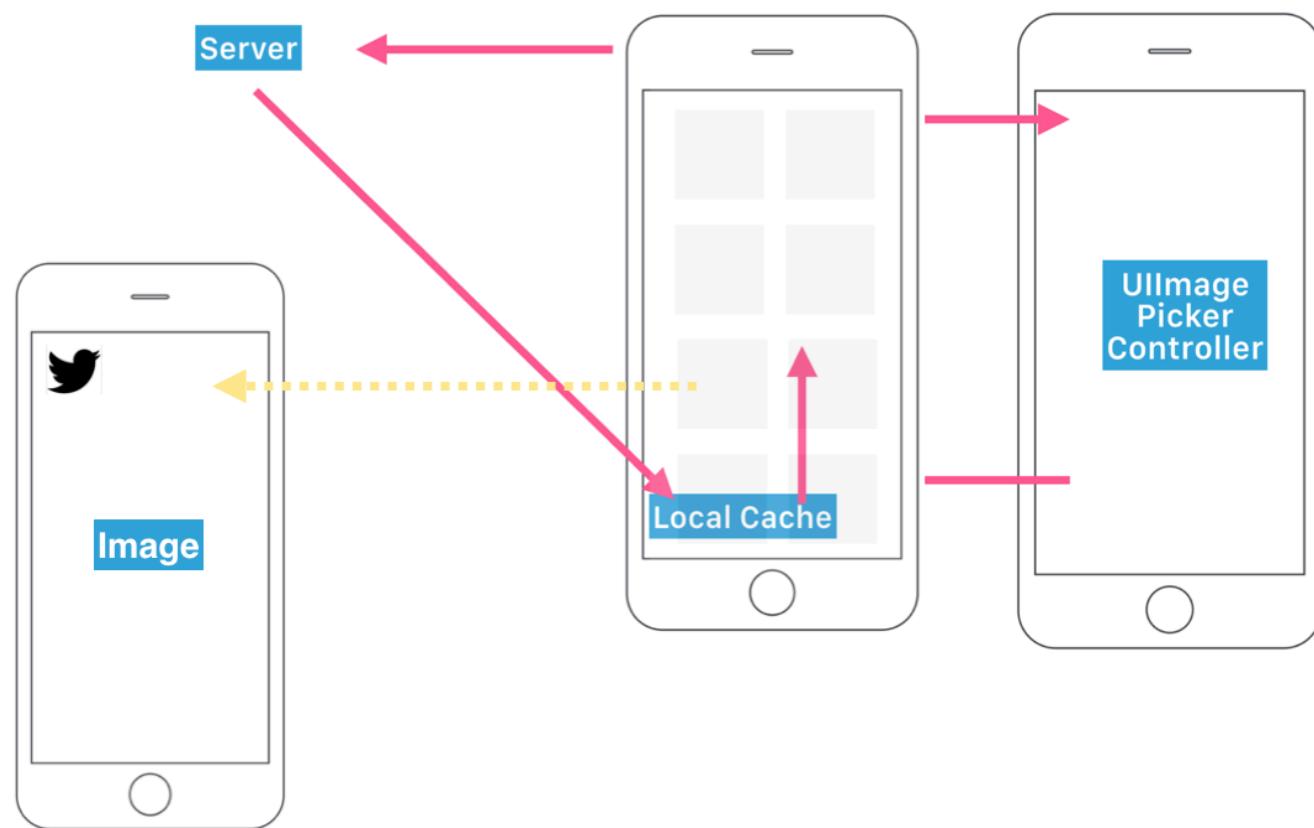


PHOTO TIMELINE



API

API

Get a json list of most recent submitted pictures

```
http://--.appspot.com/user/<USERNAME>/json/
```

See a list of the most recent on a web page (useful for debugging)

```
http://--.appspot.com/user/<USERNAME>/web/
```

Endpoint for posting images to server. There is an optional "caption" parameter that you can use.

```
http://--.appspot.com/user/<USERNAME>/post/
```

PHOTO TIMELINE

API

```
# Get a json list of most recent submitted pictures  
http://--.appspot.com/user/<USERNAME>/json/
```

```
# See a list of the most recent on a web page (useful for  
debugging  
http://--.appspot.com/user/<USERNAME>/web/
```

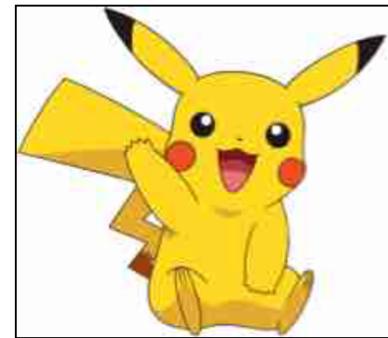
```
# Endpoint for posting images to server. There is an  
optional "caption" parameter that you can use.  
http://--.appspot.com/post/<USERNAME>/
```

API



Caption: bye
User: tabinks
Date:2016-08-01 14:13:30.511470

<http://stachesandglasses.appspot.com/user/tabinks/web/>



Caption: bye
User: tabinks
Date:2016-08-01 14:06:58.791530

API

```
"user": "default"}, {"date": "2015-04-15 08:55:14.177850", "caption": "Test2", "image_url":  
"image/ahNzfnN0YWNoZXNhbmRnbGFzc2Vzch4LEgRVc2VyIgdkZWZhdWx0DAsSBVBob3RvGIm  
"user": "default"}, {"date": "2014-04-29 17:20:39.047800", "caption": "", "image_url":  
"image/ahNzfnN0YWNoZXNhbmRnbGFzc2Vzch0LEgRVc2VyIgdkZWZhdWx0DAsSBVBob3RvGPI  
"user": "default"}, {"date": "2014-02-06 02:33:25.947030", "caption": "", "image_url":  
"image/ahNzfnN0YWNoZXNhbmRnbGFzc2Vzch0LEgRVc2VyIgdkZWZhdWx0DAsSBVBob3RvGOk  
"user": "default"}, {"date": "2013-05-08 18:11:30.375380", "caption": "test", "image_url":  
"image/ahNzfnN0YWNoZXNhbmRnbGFzc2VzchwLEgRVc2VyIgdkZWZhdWx0DAsSBVBob3RvGA  
"user": "default"}]}
```

<http://stachesandglasses.appspot.com/user/tabinks/json/>

API



A screenshot of a Mac OS X desktop environment showing a web browser window. The window title is "stachesandglasses.appspot.com". The address bar also contains the URL. The browser menu bar includes "File", "Edit", "View", "Window", and "Help". Below the menu bar is a toolbar with icons for Mail, iCal, and Spotlight search. The main content area of the browser shows a yellow Pikachu character from the Pokémon series, smiling and holding a small lightning bolt. The Pikachu is positioned over a white background. At the bottom of the browser window, there is a yellow rectangular box containing a URL.

<http://stachesandglasses.appspot.com/image/ahNzfnN0YWNoZXNhbmRnbGFzc2Vzch4LEgRVc2Vylgd0YWJpbmtzDAsSBVBob3RvGNOGAww/>

API

```
8 func uploadRequest(user: NSManagedObject, image: UIImage, caption: String) {
9
10    let boundary = generateBoundaryString()
11    let scaledImage = resize(image)
12    let imageJPEGData = UIImageJPEGRepresentation(scaledImage, 0.5)
13
14    guard let imageData = imageJPEGData else {
15        return
16    }
17
18    // Create the URL, the user should be logged in
19    let url = NSURL(string: "https://stachanov.com/api/v1/users/\(user.id)/image")
20
21    let request = NSMutableURLRequest(url: url!)
22    request.HTTPMethod = "POST"
23
24    // Set the boundary
25    let mimetype = "multipart/form-data; boundary=\(boundary)"
26    // This is not needed, but it's good practice
27    let fileName = "test.jpg"
28
29    // Create data for the body
30    let body =NSMutableData()
31    body.appendData("\r\n--\(\boundary)\r\n")
32
33    // Caption data
34    body.appendData("Content-Disposition: form-data; name=\"caption\"\r\n")
35    body.appendData(NSUTF8StringEncoding)!)
36
37    // Image data
38    body.appendData("--\(\boundary)\r\n")
39    body.appendData("Content-Type: \(mimetype)\r\n")
40    body.appendData("Content-Disposition: form-data; name=\"image\";\r\n")
41    body.appendData("filename=\(fileName)\r\n")
42    body.appendData(NSUTF8StringEncoding)!)
43
44    // Setting header
45    request.setValue(mimetype, forHTTPHeaderField: "Content-Type")
46
47    // Setting body
48    request.HTTPBody = body
49
50    let task = session.dataTaskWithRequest(request) { (data, response, error) in
51        if let error = error {
52            print(error)
53        } else {
54            print(data)
55        }
56    }
57
58    task.resume()
59}
```

There is a playground for you to play around in the playground (and your assignment)

FILE STRUCTURE

PHOTO TIMELINE

- Application files
 - app.yaml - App Engine application settings
 - main.py - Main application
 - models.py - Datastore models
 - index.yaml - Generated files for indexing

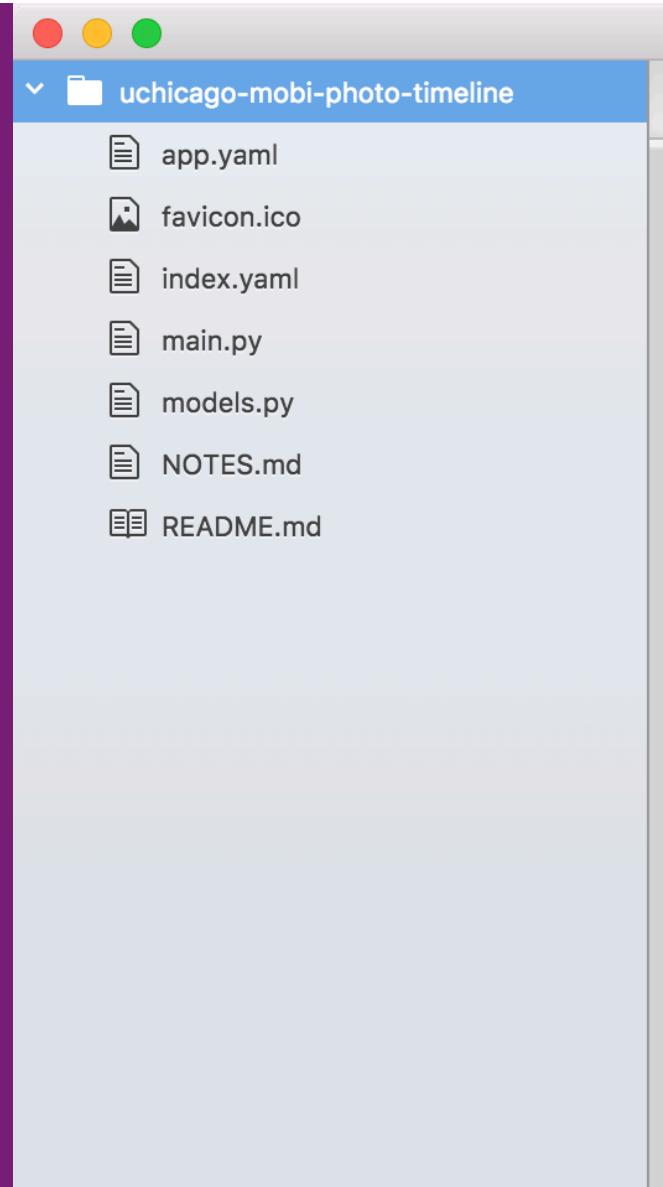


PHOTO TIMELINE

- app.yaml file
- More options
 - Specify machines
 - Global variables
 - ...

How should the application route requests

libraries to import

```
application: uchicago-mobi-photo-timeline
runtime: python27
api_version: 1
threadsafe: true
```

Environment

```
# [START handlers]
handlers:
- url: /favicon\.ico
  static_files: favicon.ico
  upload: favicon\.ico
- url: /.*
  script: main.app
# [END handlers]
```

```
# [START libraries]
libraries:
- name: webapp2
  version: latest
- name: jinja2
  version: latest
# [END libraries]
```

```
skip_files:
- ^(.*/)?#.*#$
```

```
- ^(.*/)?.*~$
```

```
- ^(.*/)?.*\py[co]$
```

```
- ^(.*/)?\..*$
```

```
- ^(.*/)?\.txt$
```

Files to skip when uploading

PHOTO TIMELINE

Model class entities

- models.py
 - Datastore classes
 - Common organization

class
methods
to query
entities

```
from google.appengine.ext import ndb

class Photo(ndb.Model):
    """Models a user uploaded photo entry"""

    user = ndb.StringProperty()
    image = ndb.BlobProperty()
    caption = ndb.StringProperty()
    date = ndb.DateTimeProperty(auto_now_add=True)

    @classmethod
    def query_user(cls, ancestor_key):
        """Return all photos for a given user"""
        return cls.query(ancestor=ancestor_key).order(-cls.date)

    @classmethod
    def query_user_alternate(cls, ancestor_key):
        """Return all photos for a given user using the gql syntax.
        It returns the same as the above method.
        """
        return ndb.gql('SELECT * '
                      'FROM Photo '
                      'WHERE ANCESTOR IS :1 '
                      'ORDER BY date DESC LIMIT 10',
                      ancestor_key)
```

PHOTO TIMELINE

MODELS.PY

```
@classmethod
def query_user(cls, ancestor_key):
    """Return all photos for a given user"""
    return cls.query(ancestor=ancestor_key).order(-cls.date)

@classmethod
def query_user_alternate(cls, ancestor_key):
    """Return all photos for a given user using the gql syntax.
    It returns the same as the above method.
    """
    return ndb.gql('SELECT * '
                  'FROM Photo '
                  'WHERE ANCESTOR IS :1 '
                  'ORDER BY date DESC LIMIT 10',
                  ancestor_key)
```

same thing
different
way

PHOTO TIMELINE

MAIN.PY

URL Routing

```
app = webapp2.WSGIApplication([
    ('/', HomeHandler),
    webapp2.Route('/logging/', handler=LoggingHandler),
    webapp2.Route('/image/<key>/', handler=ImageHandler),
    webapp2.Route('/post/<user>/', handler=PostHandler),
    webapp2.Route('/user/<user>/<type>/', handler=UserHandler)
],  
debug=True)
```

PHOTO TIMELINE

MAIN.PY

```
app = webapp2.WSGIApplication([
    ('/', HomeHandler),
    webapp2.Route('/logging/', handler=LoggingHandler),
    http://localhost:8080/
    webapp2.Route('/image/', handler=ImageHandler),
    webapp2.Route('/post/<user>/', handler=PostHandler),
    webapp2.Route('/user/<user>/<type>/', handler=UserHandler)
],
debug=True)
```

PHOTO TIMELINE

MAIN.PY

http://localhost:8080/logging/

```
app = webapp2.WSGIApplication([
    ('/', HomeHandler),
    webapp2.Route('/logging/', handler=LoggingHandler),
    webapp2.Route('/image/<key>/', handler=ImageHandler),
    webapp2.Route('/post/<user>/', handler=PostHandler),
    webapp2.Route('/user/<user>/<type>/', handler=UserHandler)
],  
debug=True)
```

PHOTO TIMELINE

MAIN.PY

```
app = webapp2.WSGIApplication([
    ('/', HomeHandler),
    http://localhost:8080/post/<user>/,
    webapp2.Route('/image/<key>/', handler=ImageHandler),
    webapp2.Route('/post/<user>/', handler=PostHandler),
    webapp2.Route('/user/<user>/<type>/', handler=UserHandler)
],
debug=True)
```

PHOTO TIMELINE

MAIN.PY

```
app = webapp2.WSGIApplication([
    ('/', HomeHandler),
    webapp2.Route('/logging/', handler=LoggingHandler),
    webapp2.Route('/image/<key>/', handler=ImageHandler),
    webapp2.Route('/post/<user>/', handler=PostHandler),
    webapp2.Route('/user/<user>/<type>/', handler=UserHandler)
],  
debug=True)
```

http://localhost:8080/user/<user>/json/

PHOTO TIMELINE

MAIN.PY

```
app = webapp2.WSGIApplication([
    ('/', HomeHandler),
    webapp2.Route('/logging/', handler=LoggingHandler),
    webapp2.Route('/image/<key>/', handler=ImageHandler),
    webapp2.Route('/post/<user>/', handler=PostHandler),
```

"""The home page of the app"""

```
class HomeHandler(webapp2.RequestHandler):
```

"""Show the webform when the user is on the home page"""

```
def get(self):
```

PHOTO TIMELINE

```
webapp2.Route('/logging/', handler=LoggingHandler),  
webapp2.Route('/image/<key>', handler=ImageHandler),  
webapp2.Route('/post/<user>', handler=PostHandler),  
webapp2.Route('/user/<user>/<type>', handler=UserHandler)  
],  
debug=True)
```

"""Handle requests for an image ebased on its key"""

```
class ImageHandler(webapp2.RequestHandler):
```

```
def get(self, key):
```

"""Write a response of an image (or 'no image') based on a key

```
photo = ndb.Key(urlsafe=key).get()
```

```
if photo.image:
```

```
    self.response.headers['Content-Type'] = 'image/jpeg'  
    self.response.out.write(photo.image)
```

PHOTO TIMELINE

```
webapp2.Route('/post/<user>/' , handler=PostHandler),  
webapp2.Route('/user/<user>/<type>/' ,handler=UserHandler)  
],  
debug=True)
```

"""Handle activities associated with a given user"""

```
class UserHandler(webapp2.RequestHandler):
```

"""Print json or html version of the users photos"""

```
def get(self,user,type):  
    ancestor_key = ndb.Key("User", user)
```

PHOTO TIMELINE

- The URI (uniform resource identifiers) is a matter of style

```
app = webapp2.WSGIApplication([
    ('/', HomeHandler),
    webapp2.Route('/logging/', handler=LoggingHandler),
    webapp2.Route('/image/<key>/', handler=ImageHandler),
    webapp2.Route('/post/<user>/', handler=PostHandler),
    webapp2.Route('/user/<user>/<type>/', handler=UserHandler)
],  
    debug=True)
```

/users/<user>/json

?user=<user>&style=json

**Same data
is passed**

PHOTO TIMELINE

```
# Test your application from the command line using `curl`  
  
curl -X GET http://localhost:8080/  
  
curl -X GET http://localhost:8080/user/default/json/  
  
curl -X POST -H "Content-Type: multipart/form-data" -F  
caption='curl' -F "image=@kitten.jpg" http://localhost:  
8080/post/lolakitty/
```

DEPLOY TO APP ENGINE

DEPLOY TO APP ENGINE

- Create a new project

The screenshot shows the Google Cloud Platform dashboard for the project "mpcs51033-2017-autumn-photos".

Project info:

- Project name: mpcs51033-2017-autumn-photos
- Project ID: mpcs51033-2017-autumn-photos
- Project number: 699588220284

[Go to project settings](#)

Compute Engine:

CPU (%) ▾

There is no data for this chart

[Go to the Compute Engine dashboard](#)

APIs:

Requests (requests/sec)

There is no data for this chart

[Go to APIs overview](#)

Google Cloud Platform status:

All services normal

[Go to Cloud status dashboard](#)

Billing:

Estimated charges
For the billing period Oct 1 – 2, 2017

[View detailed charges](#)

Error Reporting:

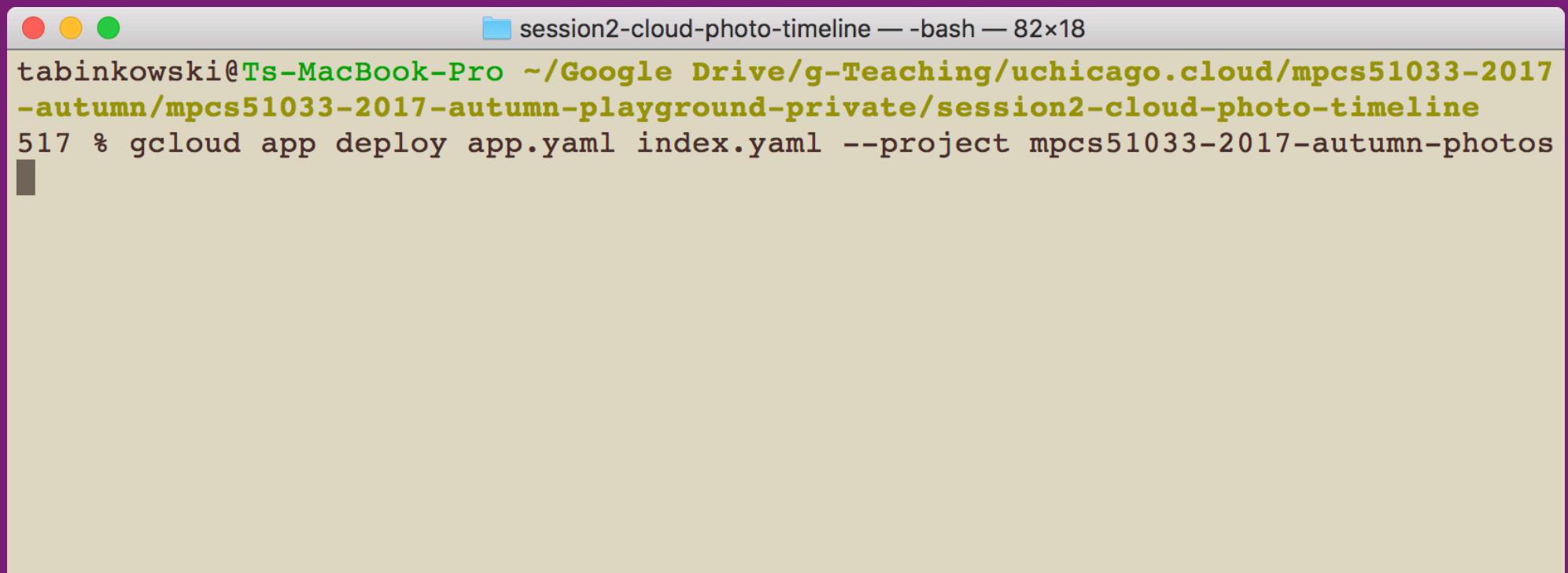
No sign of any errors. Have you set up Error Reporting?

[Learn how to set up Error Reporting](#)

News:

- Profiling Kubernetes init time: Google Cloud Performance Atlas
10 hours ago
- Google Container Engine - Kubernetes 1.8 takes advantage of the cloud built for containers
3 days ago

DEPLOY TO APP ENGINE



```
tabinkowski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.cloud/mpcs51033-2017-autumn/mpcs51033-2017-autumn-playground-private/session2-cloud-photo-timeline 517 % gcloud app deploy app.yaml index.yaml --project mpcs51033-2017-autumn-photos
```

- Deploy using gcloud

DEPLOY TO APP ENGINE

```
517 % gcloud app deploy app.yaml index.yaml --project mpcs51033-2017-autumn-photos
You are creating an app for project [mpcs51033-2017-autumn-photos].
WARNING: Creating an App Engine application for a project is irreversible and the region
cannot be changed. More information about regions is at
<https://cloud.google.com/appengine/docs/locations>.
```

Please choose the region where you want your App Engine application located:

```
[1] us-central      (supports standard and flexible)
[2] europe-west    (supports standard and flexible)
[3] europe-west3   (supports standard and flexible)
[4] europe-west2   (supports standard and flexible)
[5] us-east1        (supports standard and flexible)
[6] us-east4        (supports standard and flexible)
[7] asia-northeast1 (supports standard and flexible)
[8] australia-southeast1 (supports standard and flexible)
[9] southamerica-east1 (supports standard and flexible)
[10] cancel
```

Please enter your numeric choice: █

DEPLOY TO APP ENGINE

Services to deploy:

```
descriptor:      [/Users/tabinkowski/Google Drive/g-Teaching/uchicago.cloud/mpcs51033-2017-autumn/mpcs51033-2017-autumn-playground-private/session2-cloud-photo-timeline/app.yaml]
source:         [/Users/tabinkowski/Google Drive/g-Teaching/uchicago.cloud/mpcs51033-2017-autumn/mpcs51033-2017-autumn-playground-private/session2-cloud-photo-timeline]
target project: [mpcs51033-2017-autumn-photos]
target service: [default]
target version: [20171002t194803]
target url:     [https://mpcs51033-2017-autumn-photos.appspot.com]
```

Configurations to update:

```
descriptor:      [/Users/tabinkowski/Google Drive/g-Teaching/uchicago.cloud/mpcs51033-2017-autumn/mpcs51033-2017-autumn-playground-private/session2-cloud-photo-timeline/index.yaml]
type:           [datastore indexes]
target project: [mpcs51033-2017-autumn-photos]
```

Do you want to continue (Y/n)? █

DEPLOY TO APP ENGINE

```
session2-cloud-photo-timeline — bash — 93x21
Uploading 8 files to Google Cloud Storage
File upload done.
Updating service [default]...done.
Updating service [default]...Waiting for operation [apps/mpcs51033-2017-autumn-photos/operations/3504ca7a-1189-4de0-ac1-f24639618faf] to complete...done.
Updating service [default]...done.
Deployed service [default] to [https://mpcs51033-2017-autumn-photos.appspot.com]
Updating config [index]...done.

Indexes are being rebuilt. This may take a moment.

You can stream logs from the command line by running:
$ gcloud app logs tail -s default

To view your application in the web browser run:
$ gcloud app browse
tabinkowski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.cloud/mpcs51033-2017-autumn/mpcs51033-2017-autumn-playground-private/session2-cloud-photo-timeline
```

DEPLOY TO APP ENGINE

Project info

Project name
mpcs51033-2017-autumn-photos

Project ID
[mpcs51033-2017-autumn-photos](#)

Project number
699588220284

[Go to project settings](#)

Resources

Compute Engine
You do not have permission to see this information

Cloud Functions
You do not have permission to see this information

Compute Engine

CPU (%) ▾

There is no data for this chart

[Go to the Compute Engine dashboard](#)

APIs

Requests (requests/sec)

There is no data for this chart

Google Cloud Platform status

All services normal

[Go to Cloud status dashboard](#)

Billing

Estimated charges \$0.00
For the billing period Oct 1 – 2, 2017

[View detailed charges](#)

Error Reporting

No sign of any errors. Have you set up Error Reporting?

[Learn how to set up Error Reporting](#)

DEPLOY TO APP ENGINE

The screenshot shows the Google Cloud Platform App Engine Dashboard for the project 'mpcs51033-2017-autumn-photos'. The dashboard displays deployment information for version 20171002t194803 (100%). The URL mpcs51033-2017-autumn-photos.appspot.com is shown, along with the region 'us-central'. A summary table indicates 'No requests in this time interval' for the date Oct 2, 2017 6:56 PM. The dashboard also includes tabs for Services, Versions, Instances, Task queues, Security scans, Firewall rules, Quotas, and Blobstore.

App Engine Dashboard - mpcs

Secure | https://console.cloud.google.com/appengine?project=mpcs51033-2017-autumn-photos&organizationId=4099945739...

Google Cloud Platform mpcs51033-2017-autum... SHOW INFO PANEL

Dashboard

Version 20171002t194803 (100%)

mpcs51033-2017-autumn-photos.appspot.com Region: us-central

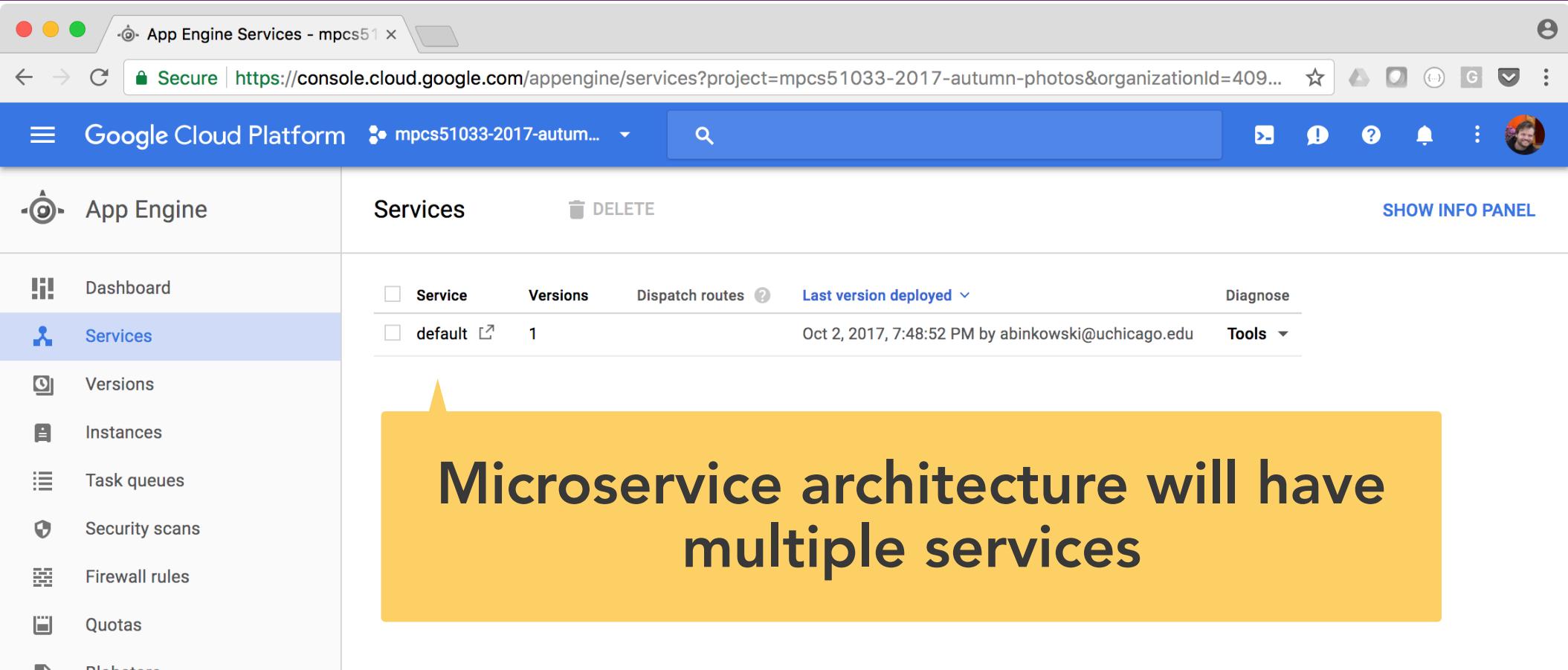
Summary 1 hour 6 hours 12 hours 1 day 2 days 4 days 7 days 14 days 30 days

Count/sec Oct 2, 2017 6:56 PM

No requests in this time interval

Services Versions Instances Task queues Security scans Firewall rules Quotas Blobstore

DEPLOY TO APP ENGINE



The screenshot shows the Google Cloud Platform App Engine Services dashboard. The left sidebar has a 'Services' item selected, highlighted with a blue background. The main content area displays a table for the 'Services' section. The table has columns for Service, Versions, Dispatch routes, Last version deployed, and Diagnose. One row is visible for the 'default' service, which was deployed on Oct 2, 2017, at 7:48:52 PM by abinkowski@uchicago.edu. A yellow callout box with a black border and white text is overlaid on the bottom right of the table, containing the text: "Microservice architecture will have multiple services".

Service	Versions	Dispatch routes	Last version deployed	Diagnose
default	1		Oct 2, 2017, 7:48:52 PM by abinkowski@uchicago.edu	Tools

Microservice architecture will have multiple services

DEPLOY TO APP ENGINE

Secure | https://console.cloud.google.com/appengine/versions?project=mpcs51033-2017-autumn-photos&organizationId=409...  

Google Cloud Platform mpcs51033-2017-autum...      

App Engine  Versions  DELETE  START   SHOW INFO PANEL

Dashboard 

Services 

Versions   Filter versions

 Version	Status	Traffic Allocation	Instances	Runtime	Environment	Size	Deployed
 20171002t194803 	Serving	 100%	0	python27	Standard	19 KB	Oct 2, 2017, 7:48:52 abinkowski@uchicago.edu

Instances 

Task queues 

Security scans 

Firewall rules 

Quotas 

Blobstore 

 Versioning

DEPLOY TO APP ENGINE

The screenshot shows the Google Cloud Platform interface for managing an application named 'mpcs51033-2017-autumn-photos'. The left sidebar is titled 'App Engine' and lists various management options: Dashboard, Services, Versions, Instances, Task queues, Security scans, Firewall rules, Quotas, and Disk quotas. The main content area is titled 'Settings' and is currently viewing the 'Application settings' tab. This tab includes sections for Daily spending limit (Unlimited), Google login cookie expiration (Default (1 day)), Referrers (Google Accounts API), and Email API authorized senders (None). There is also a 'Disable application' section with a note about stopping serving requests without losing data or state. The top navigation bar shows the project name and a search bar, along with standard browser controls.

App Engine Settings - mpcs51

Secure | https://console.cloud.google.com/appengine/settings?project=mpcs51033-2017-autumn-photos&organizationId=4099...

Google Cloud Platform mpcs51033-2017-autum... ...

App Engine

Dashboard

Services

Versions

Instances

Task queues

Security scans

Firewall rules

Quotas

Disk quotas

Settings

Application settings Custom domains SSL certificates

Edit

Daily spending limit Unlimited

Google login cookie expiration Default (1 day)

Referrers Google Accounts API

Email API authorized senders None

Disable application

Disabling an application will stop all serving requests, but you will not lose any data or state. Billing charges will still incur when applicable. You can re-enable your application at any time.

DEPLOY TO APP ENGINE

Google Cloud Platform mpcs51033-2017-autum... ▾

SEARCH

☰

! ? 📡 🔔 ⋮



App Engine	Quotas	VIEW USAGE HISTORY		
 Dashboard	The quota details for this application are grouped by API and are listed below. If your application exceeds 50% of any particular quota halfway through the day, it may exceed the quota before the day is over. To learn more about how quotas work, read Understanding Quotas and Why is My App Over Quota?			
 Services	Apply for higher quota			
 Versions	Quotas are reset every 24 hours. Next reset: 6 hours			
 Instances	Resource	Usage today	Daily quota	Per-minute quota
 Task queues	Requests	4	--	Standard
 Security scans	Outgoing Bandwidth	0.000003 GB	--	Standard
 Firewall rules	Incoming Bandwidth	0.0015 GB	--	Standard
 Quotas	Secure Requests	4	--	Standard
 Blobstore	Secure Outgoing Bandwidth	0.000003 GB	--	Standard
 Memcache	Secure Incoming Bandwidth	0.0015 GB	--	Standard
 Search	Frontend Instance Hours	0.06 Instance Hours	--	Standard
	Storage			

PHOTO TIMELINE

```
# Test your application from the command line using `curl`
```

```
curl -X GET https://mpcs51033-2017-autumn-photos.appspot.com
```

```
curl -X GET https://mpcs51033-2017-autumn-photos.appspot.com/user/default/json/
```

```
curl -X POST -H "Content-Type: multipart/form-data" -F caption='curl' -F  
"image=@kitten.jpg" https://mpcs51033-2017-autumn-photos.appspot.com/post/  
lolakitty/
```

DEPLOY TO APP ENGINE

Apple Developer Discover Design Develop Distribute Support

Documentation > Foundation > Archives and Seriali... > Using JSON with Cu... Language: Swift

Sample Code

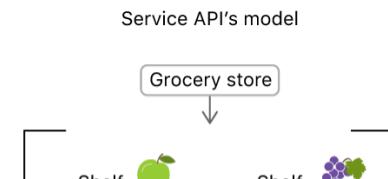
Using JSON with Custom Types

Demonstrates approaches for encoding and decoding different kinds of JSON in Swift.

Download

Overview

JSON data you send or receive from other apps, services, and files can come in many different shapes and structures. Use the techniques described in this sample to handle the differences between external JSON data and your app's model types.



DEVELOPMENT AND LOGGING

DEVELOPMENT AND LOGGING

- Different levels of logging
- Appear different in the console
- Used to filter logs

```
import logging

class LoggingHandler(webapp2.RequestHandler):
    """Demonstrate the different levels of logging"""

    def get(self):
        logging.debug('This is a debug message')
        logging.info('This is an info message')
        logging.warning('This is a warning message')
        logging.error('This is an error message')
        logging.critical('This is a critical message')

    try:
        raise ValueError('This is a sample value error.')
    except ValueError:
        logging.exception('A example exception log.')

    self.response.out.write('Logging example.')
```

DEVELOPMENT AND LOGGING

```
session2-cloud-photo-timeline — python  
TTP/1.1" 200 188  
INFO    2017-10-03 00:20:33,178 module.py  
/Users/tabinkowski/Google Drive/g-Teachin  
mpcs51033-2017-autumn-playground-private/session2-cloud-photo-timeline/README.md  
INFO    2017-10-03 00:39:24,441 main.py:153] This is an info message  
WARNING 2017-10-03 00:39:24,442 main.py:154] This is a warning message  
ERROR   2017-10-03 00:39:24,442 main.py:155] This is an error message  
CRITICAL 2017-10-03 00:39:24,442 main.py:156] This is a critical message  
ERROR   2017-10-03 00:39:24,442 main.py:161] A example exception log.  
Traceback (most recent call last):  
  File "/Users/tabinkowski/Google Drive/g-Teaching/uchicago.cloud/mpcs51033-2017-a  
utumn/mpcs51033-2017-autumn-playground-private/session2-cloud-photo-timeline/main.  
py", line 159, in get  
    raise ValueError('This is a sample value error.')  
ValueError: This is a sample value error.  
INFO    2017-10-03 00:39:24,474 module.py:821] default: "GET /logging/ HTTP/1.1"  
200 16
```

```
def get(self):  
    logging.debug('This is a debug message')  
    logging.info('This is an info message')  
    logging.warning('This is a warning message')  
    logging.error('This is an error message')  
    logging.critical('This is a critical message')  
  
try:  
    raise ValueError('This is a sample value error.')  
except ValueError:  
    logging.exception('A example exception log.')
```

DEVELOPMENT AND LOGGING

The screenshot shows the Google Cloud Platform interface with the following details:

- Header:** Google Cloud Platform, Project: mpcs51033-2017-autum..., Search bar, and various navigation icons.
- Left Sidebar:** Navigation menu with sections: STACKDRIVER (Monitoring, Debug, Trace), TOOLS (Logging, Error Reporting, Container Registry, Source Repositories).
- Current View:** The "Logs" tab is selected under the "Logging" section.
- Table Data:** A table showing resource usage statistics. The columns are: Resource, Usage today, Daily quota, and Per-minute quota.

Resource	Usage today	Daily quota	Per-minute quota
Requests	4	--	Standard
Logs	0.000003 GB	--	Standard
Logs-based metrics	0.0015 GB	--	Standard
Exports	4	--	Standard
Resource usage	0.000003 GB	--	Standard
Secure Incoming Bandwidth	0.0015 GB	--	Standard
Frontend Instance Hours	0.06 Instance Hours	--	Standard
Storage			

DEVELOPMENT AND LOGGING

Google Cloud Platform mpcs51033-2017-autum... ▾

Stackdriver Logging

Logs

Logs-based metrics

Exports

Resource usage

CREATE METRIC CREATE EXPORT ⏪ ⏩

Filter by label or text search

GAE Application request_log Any log level Jump to date

2017-10-02 CDT View Options

Time	Method	Path	Size	Latency	User Agent
19:57:35.158	GET	/user/andrew/json/	292 B	87 ms	Safari 11
19:57:34.764	POST	/post/default/	162 B	357 ms	Safari 11
19:56:59.516	GET	/favicon.ico	8.15 KB	1 ms	Safari 11
19:56:58.476	GET	/	341 B	740 ms	Safari 11

DEVELOPMENT AND LOGGING

The screenshot shows the Google Cloud Platform Logging interface. On the left, there are navigation links for 'Exports' and 'Resource usage'. The main area displays log entries for the date 2017-10-02 CDT. A yellow header bar at the top of the log list includes the date, a download arrow icon, and 'View Options' with a dropdown arrow.

The log entries are as follows:

- A log entry for a GET request at 20:03:04.439 with status 200, 121 bytes, 64 ms duration, from Safari 11, and the URL /logging/. It includes a long client IP and user agent string.
- An expand/collapse section starting with '...', with 'Expand all' and 'Collapse all' buttons.
- Multiple log entries from main.py:
 - 20:03:04.441: This is a debug message (file: /base/data/home/apps/s~mpcs51033-2017-autumn-photos/20171002t194803.404530012887445132/main.py:152)
 - 20:03:04.442: This is an info message (file: /base/data/home/apps/s~mpcs51033-2017-autumn-photos/20171002t194803.404530012887445132/main.py:153)
 - 20:03:04.442: This is a warning message (file: /base/data/home/apps/s~mpcs51033-2017-autumn-photos/20171002t194803.404530012887445132/main.py:154)
 - 20:03:04.442: This is an error message (file: /base/data/home/apps/s~mpcs51033-2017-autumn-photos/20171002t194803.404530012887445132/main.py:155)
 - 20:03:04.442: This is a critical message (file: /base/data/home/apps/s~mpcs51033-2017-autumn-photos/20171002t194803.404530012887445132/main.py:156)
- 20:03:04.442: An example exception log (file: /base/data/home/apps/s~mpcs51033-2017-autumn-photos/20171002t194803.404530012887445132/main.py:161). It includes a traceback and the following code snippet:

```
File "/base/data/home/apps/s~mpcs51033-2017-autumn-photos/20171002t194803.404530012887445132/main.py", line 159, in get
    raise ValueError('This is a sample value error.')
```

DEVELOPMENT AND LOGGING

```
# Download the logs from a running application  
% gcloud app logs read -s default
```

DATASTORE

DATASTORE

- NoSQL document database
- Highly structured mutable data
- Designed and implemented for scaling

Cloud Datastore Overview

[Python](#) | [Java](#) | [PHP](#) | [Go](#)

Google Cloud Datastore is a NoSQL document database built for automatic scaling, high performance, and ease of application development. Cloud Datastore features include:

- **Atomic transactions.** Cloud Datastore can execute a set of operations where either all succeed, or none occur.
- **High availability of reads and writes.** Cloud Datastore runs in Google data centers, which use redundancy to minimize impact from points of failure.
- **Massive scalability with high performance.** Cloud Datastore uses a distributed architecture to automatically manage scaling. Cloud Datastore uses a mix of indexes and query constraints so your queries scale with the size of your result set, not the size of your data set.
- **Flexible storage and querying of data.** Cloud Datastore maps naturally to object-oriented and scripting languages, and is exposed to applications through multiple clients. It also provides a SQL-like [query language](#).
- **Balance of strong and eventual consistency.** Cloud Datastore ensures that entity lookups by key and ancestor queries always receive strongly consistent data. All other queries are eventually consistent. The consistency models allow your application to deliver a great user experience while handling large amounts of data and users.
- **Encryption at rest.** Cloud Datastore automatically encrypts all data before it is written to disk and automatically decrypts the data when read by an authorized user. For more information, see [Server-Side Encryption](#).
- **Fully managed with no planned downtime.** Google handles the administration of the Cloud Datastore service so you can focus on your application. Your application can still use Cloud Datastore when the service receives a planned upgrade.

Comparison with traditional databases

While the Cloud Datastore interface has many of the same features as traditional databases, as a NoSQL database it differs from them in the way it describes relationships between data objects. Here's a high-level comparison of Cloud Datastore and relational database concepts:

Concept	Cloud Datastore	Relational database
Category of object	Kind	Table
One object	Entity	Row
Individual data for an object	Property	Field
Unique ID for an object	Key	Primary key

DATASTORE

- Remember just one of many options for store data
- Available via API through all GPC services

Cloud Datastore is ideal for applications that rely on highly available structured data at scale. You can use Cloud Datastore to store and query all of the following types of data:

- Product catalogs that provide real-time inventory and product details for a retailer.
- User profiles that deliver a customized experience based on the user's past activities and preferences.
- Transactions based on [ACID](#) properties, for example, transferring funds from one bank account to another.

Other storage options

Cloud Datastore is not ideal for every use case. For example, Cloud Datastore is not a relational database, and it is not an effective storage solution for analytic data.

Here are some common scenarios where you should probably consider an alternative to Cloud Datastore:

- If you need a relational database with full SQL support for an online transaction processing (OLTP) system, consider [Google Cloud SQL](#).
- If you don't require support for ACID transactions or if your data is not highly structured, consider [Cloud Bigtable](#).
- If you need interactive querying in an online analytical processing (OLAP) system, consider [Google BigQuery](#).
- If you need to store large immutable blobs, such as large images or movies, consider [Google Cloud Storage](#).

For more information about other storage options, see the [Choosing a Storage Option](#) guide.

Connecting to Cloud Datastore with App Engine

App Engine's Python standard runtime connects to Cloud Datastore using the [NDB Client Library](#). The NDB Client Library provides persistent storage in a schemaless object datastore. It supports automatic caching, sophisticated queries, and atomic transactions.

DATASTORE

- Data types available
- Some very specialized for applications

Property class	Value type	Sort order
<code>IntegerProperty</code>	<code>int</code> <code>long</code> (64 bits)	Numeric
<code>FloatProperty</code>	<code>float</code>	Numeric
<code>BooleanProperty</code>	<code>bool</code>	<code>False < True</code>
<code>StringProperty</code>	<code>str</code> unicode	Unicode (<code>str</code> is treated as ASCII)
<code>TextProperty</code>	<code>db.Text</code>	None
<code>ByteStringProperty</code>	<code>ByteString</code>	Byte order
<code>BlobProperty</code>	<code>db.Blob</code>	None
<code>DateProperty</code> <code>TimeProperty</code> <code>DateTimeProperty</code>	<code>datetime.date</code> <code>datetime.time</code> <code>datetime.datetime</code>	Chronological
<code>GeoPtProperty</code>	<code>db.GeoPt</code>	By latitude, then longitude
<code>PostalAddressProperty</code>	<code>db.PostalAddress</code>	Unicode
<code>PhoneNumberProperty</code>	<code>db.PhoneNumber</code>	Unicode
<code>EmailProperty</code>	<code>db.Email</code>	Unicode
<code>UserProperty</code>	<code>users.User</code>	Email address in Unicode order. Note that you should avoid using <code>UserProperty</code> , per the note under UserProperty class description .
<code>IMProperty</code>	<code>db.IM</code>	Unicode
<code>LinkProperty</code>	<code>db.Link</code>	Unicode
<code>CategoryProperty</code>	<code>db.Category</code>	Unicode
<code>RatingProperty</code>	<code>db.Rating</code>	Numeric
<code>ReferenceProperty</code> <code>SelfReferenceProperty</code>	<code>db.Key</code>	By path elements (kind, identifier, kind, identifier...)
<code>blobstore.BlobReferenceProperty</code>	<code>blobstore.BlobInfo</code>	Byte order
<code>ListProperty</code> <code>StringListProperty</code>	<code>list</code> of a supported type	If ascending, by least element; if descending, by greatest element

DATASTORE

- Photo model in our photo timeline app
- Objects with same ancestor have different behaviors

```
class Photo(ndb.Model):
    """Models a user uploaded photo entry"""

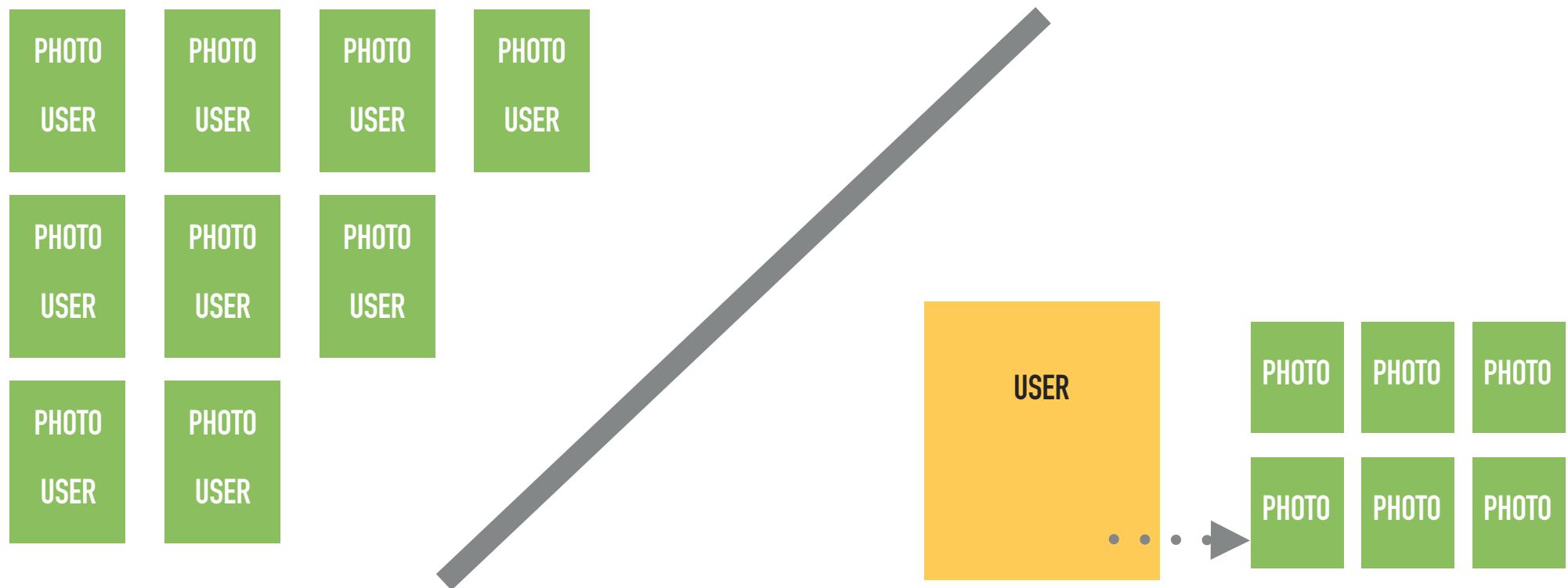
    user = ndb.StringProperty()
    image = ndb.BlobProperty()
    caption = ndb.StringProperty()
    date = ndb.DateTimeProperty(auto_now_add=True)

    @classmethod
    def query_user(cls, ancestor_key):
        """Return all photos for a given user"""
        return cls.query(ancestor=ancestor_key).order(-cls.date)

    @classmethod
    def query_user_alternate(cls, ancestor_key):
        """Return all photos for a given user using the gql syntax.
        It returns the same as the above method.
        """
        return ndb.gql('SELECT * '
                      'FROM Photo '
                      'WHERE ANCESTOR IS :1 '
                      'ORDER BY date DESC LIMIT 10',
                      ancestor_key)
```

DATASTORE

PHOTO TIMELINE DATA MODEL



DATASTORE

PHOTO TIMELINE DATA MODEL

1 to many relationships is Datastore?

```
class Photo(ndb.Model):  
    image = ndb.BlobProperty()
```

```
class User(ndb.Model):  
    name = ndb.StringProperty()
```

```
photos = ndb.KeyProperty(kind='Photo', repeated=True)  
# You could use Key or Structured property  
photos = nab.StructuredProperty(kind='Photo',  
                                 repeated=True)
```

repeated
means list
of type

DATASTORE

PHOTO TIMELINE DATA MODEL

- Structured Property

- Although the Photo instances are defined using the same syntax as for model classes they are not full-fledged entities
- Don't have their own keys
- Cannot be retrieved independently of the User entity to which they belong
- Can't have "top rated"

```
# 1 to many relationships is  
  
class Photo(ndb.Model):  
    image = nab.BlobProperty()  
  
class User(ndb.Model):  
    name = ndb.StringProperty()  
  
    photos = ndb.KeyProperty(k:  
# or #  
    photos = nab.StructuredProp
```

DATASTORE

- Cloud Datastore 101: Overview of Google's scalable NoSQL document database (Google Cloud Next '17a)
 - <https://www.youtube.com/watch?v=uZDk0NZGqHs>
- Building scalable apps with Cloud Datastore (Google Cloud Next '17) - YouTube
 - <https://www.youtube.com/watch?v=0ElqacNVuAo&t=11s>

The image shows a screenshot of a YouTube video player. The video title is "Cloud Datastore 101: Overview of Google's scalable NoSQL document database (Google Cloud Next '17a)". The video has 2,234 views and was published on March 10, 2017. The video player interface includes a play button, volume control, and a timestamp of 0:00 / 42:33. Below the video, there is a description: "You can run that on App Engine? (Google Cloud Next '17)". The channel information shows "Google Cloud" with 51K subscribers. To the right of the video player, there is a sidebar titled "Up next" showing thumbnails for other videos from the same event, such as "Google Cloud Next" and "Google Cloud Next".

MEMCACHE

MEMCACHE

- App Engine memcache service provides a fast in-memory cache of data
 - Speed up common datastore queries (for repeated queries)
 - Store temporary values

Memcache Overview

Contents ▾

[When to use a memory cache](#)

[Service levels](#)

[Limits](#)

[How cached data expires](#)

...

[Python](#) | [Java](#) | [PHP](#) | [Go](#)

This page provides an overview of the App Engine memcache service. High performance scalable web applications often use a distributed in-memory data cache in front of or in place of robust persistent storage for some tasks. App Engine includes a memory cache service for this purpose. To learn how to configure, monitor, and use the memcache service, read [Using Memcache](#).



Note: The cache is global and is shared across the application's frontend, backend, and all of its services and versions.

MEMCACHE

- Memcache is key/value pairs
 - In memory at any time
 - Change as items are written and retrieved from the cache
 - Value evicted when memory runs low
 - Expire policy

Memcache Overview

Contents ▾

[When to use a memory cache](#)
[Service levels](#)
[Limits](#)
[How cached data expires](#)
...

[Python](#) | [Java](#) | [PHP](#) | [Go](#)

This page provides an overview of the App Engine memcache service. High performance scalable web applications often use a distributed in-memory data cache in front of or in place of robust persistent storage for some tasks. App Engine includes a memory cache service for this purpose. To learn how to configure, monitor, and use the memcache service, read [Using Memcache](#).



Note: The cache is global and is shared across the application's frontend, backend, and all of its services and versions.

MEMCACHE

- The cache is global and is shared across the application's frontend, backend, and all of its services and versions.

Memcache Overview

Contents ▾

[When to use a memory cache](#)
[Service levels](#)
[Limits](#)
[How cached data expires](#)
...

[Python](#) | [Java](#) | [PHP](#) | [Go](#)

This page provides an overview of the App Engine memcache service. High performance scalable web applications often use a distributed in-memory data cache in front of or in place of robust persistent storage for some tasks. App Engine includes a memory cache service for this purpose. To learn how to configure, monitor, and use the memcache service, read [Using Memcache](#).



Note: The cache is global and is shared across the application's frontend, backend, and all of its services and versions.



MEMCACHE

- Memcache pattern is similar to other cache patterns
- Fallback to generating the data

```
def get_data():
    data = memcache.get('key')
    if data is not None:
        return data
    else:
        data = query_for_data()
        memcache.add('key', data, 60)
    return data
```

MEMCACHE

```
# Add a value if it doesn't exist in the cache
# with a cache expiration of 1 hour.
memcache.add(key="weather_USA_98105", value="raining", time=3600)

# Set several values, overwriting any existing values for these keys.
memcache.set_multi(
    {"USA_98115": "cloudy", "USA_94105": "foggy", "USA_94043": "sunny"} ,
    key_prefix="weather_" ,
    time=3600
)

# Atomically increment an integer value.
memcache.set(key="counter", value=0)
memcache.incr("counter")
memcache.incr("counter")
memcache.incr("counter")
```

MEMCACHE

- Best practices
 - Handle memcache API failures gracefully
 - Use the batching capability of the API
 - Distribute load across your memcache keyspace
 - Having a single or small set of memcache items represent a disproportionate amount of traffic will hinder your app from scaling

MEMCACHE

```
def get_data(user):
    """Get data from the datastore only if we don't have it cached"""
    key = user + "_photos"
    data = memcache.get(key)
    if data is not None:
        logging.info("Found in cache")
        return data
    else:
        logging.info("Cache miss")
        ancestor_key = ndb.Key("User", user)
        data = Photo.query_user(ancestor_key).fetch(100)
        if not memcache.add(key, data, 3600):
            logging.info("Memcache failed")
    return data
```

```
# Clear the cache (the cached version is going to be outdated)
key = user + "_photos"
memcache.delete(key)
```

MEMCACHE

```
def get_data(user):
    """Get data from the datastore only if we don't have it cached"""
    key = user + "_photos"
    data = memcache.get(key)
    if data is not None:
        logging.info("Found in cache")
        return data
    else:
        logging.info("Cache miss")
        ancestor_key = ndb.Key("User", user)
        data = Photo.query_user(ancestor_key).fetch(100)
        if not memcache.add(key, data, 3600):
            logging.info("Memcache failed")
    return data
```

NDB uses memcache internally but it is not transparent

POTENTIAL PROBLEMS WITH THE CURRENT ARCHITECTURE

PHOTO TIMELINE

FORESEEABLE PROBLEMS

- This solution will not scale very well once we have more users
- All the data is in a single entity, Photos, has to be searched to retrieve the photos for a given user name
 - This could be very costly

Entities			
Query by kind		Query by GQL	
Kind			
<input type="button" value="Photo"/>		<input type="button" value="Filter entities"/>	
<input type="checkbox"/> Name/ID	Parent <small>?</small>		<small>caption</small>
<input type="checkbox"/> id=5629499534213120	Key(User, 'andrew')		This is my fi
<input type="checkbox"/> id=5629499534213120	Key(User, 'lolakitty')		curl

PHOTO TIMELINE

FORESEEABLE PROBLEMS

- No security
 - Anyone can post or retrieve with just a username
 - No way to uniquely identify a user



PHOTO TIMELINE

FORESEEABLE PROBLEMS

- No API to delete embarrassing photos



PHOTO TIMELINE

FORESEEABLE PROBLEMS

- No API to delete embarrassing photos
- Storing images in Datastore

Google Cloud Platform

Why Google Products Solutions Launcher Pricing Customers Documentation Support Partners

Search CONSOLE

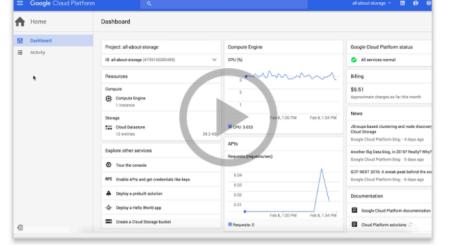
CHOOSING A STORAGE OPTION

Managed storage and databases to satisfy all your needs

VIEW DOCUMENTATION VIEW CONSOLE

Choosing a storage option

Different applications and workloads require different storage and database solutions. We offer a full suite of industry-leading storage services that are price performant and meet your needs for structured, unstructured, transactional, and relational data. This page helps you identify the solutions that fit your scenarios, whether they are mobile applications, hosting commercial software, data pipelines, or storing backups.



```
graph TD; Start(( )) -- NO --> Structured{Is your data structured?}; Start -- YES --> Mobile{Do you need Mobile SDKs?}; Structured -- YES --> Analytics{Is your workload analytics?}; Structured -- NO --> Relational{Is your data relational?}; Mobile -- YES --> Firebase[Cloud Storage for Firebase]; Mobile -- NO --> Standard[Cloud Storage]; Analytics -- YES --> BigTable[Bigtable]; Analytics -- NO --> Relational{Is your data relational?}; Relational -- YES --> Datastore[Datastore]; Relational -- NO --> HorizontalScalability{Do you need horizontal scalability?}; Relational -- NO --> MultiRegion{Do you need multi-region?}; HorizontalScalability -- YES --> CloudStorageHorizontal[Cloud Storage]; HorizontalScalability -- NO --> MultiRegion{Do you need multi-region?}; MultiRegion -- YES --> CloudStorageMultiRegion[Cloud Storage]; MultiRegion -- NO --> Updates{Do you need updates or low latency?};
```

[HTTPS://CLOUD.GOOGLE.COM/STORAGE-OPTIONS/](https://cloud.google.com/storage-options/)

BREAK TIME



SENDING EMAIL WITH APP ENGINE

SENDING EMAIL

- Two APIs for sending an email message in App Engine environment
 - mail.send_mail()
 - EmailMessage class



SENDING EMAIL

- Who can send email?
 - The Gmail or Google Apps Account of the user who is currently signed in
 - Any email address of the form anything@[APP_NAME].appspotmail.com or anything@[APP_ALIAS].appspotmail.com
 - Any email address listed in the Cloud Platform Console under Email API Authorized Senders



SENDING EMAIL

- Mail will bounce back to the senders email
 - App Engine doesn't know
- There are special rules if you are sending out bulk emails



SENDING EMAIL

```
from google.appengine.api import mail

mail.send_mail(sender="me@uchicago-mobi-photo-
                      timeline.appspotmail.com",
               to="abinkowski@uchicago.edu",
               subject="New Photo!",
               body="Hi!")
```

SENDING EMAIL

```
mail.send_mail(sender=sender_address,  
              to="Albert Johnson  
<Albert.Johnson@example.com>",  
              subject="Your account has been approved",  
              body=""""\nDear Albert:
```

Your example.com account has been approved. You can now visit
<http://www.example.com/> and sign in using your Google Account to
access new features.

Please let us know if you have any questions.

The example.com Team
""")

SENDING EMAIL

```
message = mail.EmailMessage(  
    sender=sender_address,  
    subject="Your account has been approved")  
message.to = "Albert Johnson <Albert.Johnson@example.com>"  
message.body = "Hi"  
message.send()
```

SENDING EMAIL

Limit	Amount
Maximum size of outgoing mail messages, including attachments	31.5 MB
Maximum size of incoming mail messages, including attachments	31.5 MB
Maximum size of message when an administrator is a recipient	16 KB
Maximum number of authorized senders	50

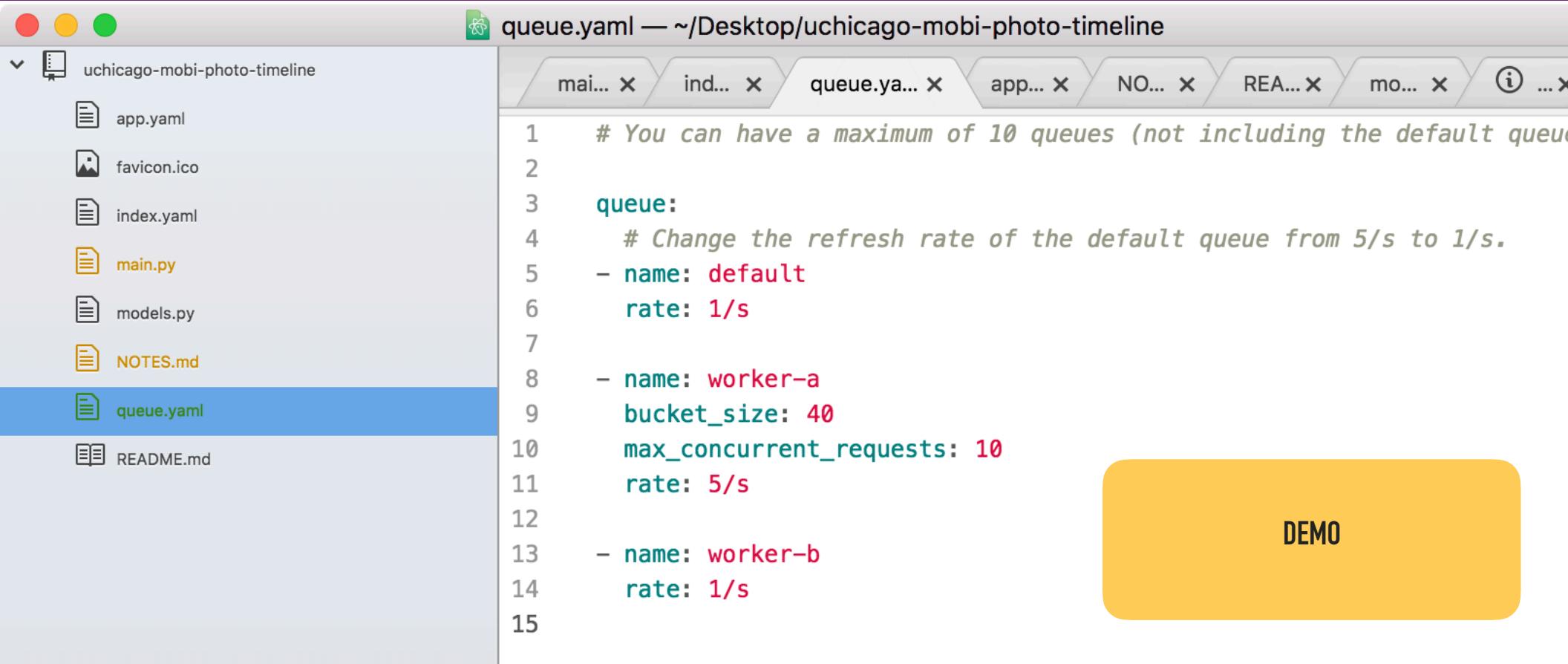
SENDING EMAIL

```
try:  
    mail.SendMessage(to='test@example.com',  
                     from='admin@example.com',  
                     subject='Test Email',  
                     body='Testing')  
except apiproxy_errors.OverQuotaError, message:  
    # Log the error.  
    logging.error(message)  
    # Display an informative message to the user.  
    self.response.out.write('The email could not be sent. '  
                           'Please try again later.')  
    self.error(500)
```

SENDING EMAIL

```
try:  
    mail.SendMessage(to='test@example.com',  
                     from='admin@example.com',  
                     subject='Test Email',  
                     body='Testing')  
except apiproxy_errors.OverQuotaError, message:  
    # Log the error.  
    logging.error(message)  
    # Display an informative message to the user.  
    self.response.out.write('The email could not be sent. '  
                           'Please try again later.')  
    self.error(500)
```

SENDING EMAIL



```
queue.yaml — ~/Desktop/uchicago-mobi-photo-timeline
mai... × ind... × queue.ya... × app... × NO... × REA... × mo... × ⓘ ... ×
∨ uchicago-mobi-photo-timeline
  app.yaml
  favicon.ico
  index.yaml
  main.py
  models.py
  NOTES.md
  queue.yaml
  README.md

1  # You can have a maximum of 10 queues (not including the default queue)
2
3  queue:
4      # Change the refresh rate of the default queue from 5/s to 1/s.
5      - name: default
6          rate: 1/s
7
8      - name: worker-a
9          bucket_size: 40
10         max_concurrent_requests: 10
11         rate: 5/s
12
13     - name: worker-b
14         rate: 1/s
15
```

DEMO

TASK QUEUE

TASK QUEUE

- Task Queue API lets applications perform work, called tasks, asynchronously outside of a user request
- If an app needs to execute work in the background, it adds tasks to task queues
- The tasks are executed later, by scalable App Engine worker services in your application

Task Queue Overview

Contents

[Push queues and pull queues](#)

[Use cases](#)

[Push queues](#)

[Pull queues](#)

[What's next](#)

[Python](#) | [Java](#) | [PHP](#)

This page describes what task queues are, and when and how to use them. The Task API lets applications perform work, called *tasks*, asynchronously outside of a user request. If an app needs to execute work in the background, it adds tasks to *task queues*. The tasks are executed later, by scalable App Engine worker services in your application.

Push queues and pull queues

Task queues come in two flavors, *push* and *pull*. The manner in which the Task Queue service dispatches task requests to worker services is different for the different queue types.

Push queues dispatch requests at a reliable, steady rate. They guarantee reliable task execution. Because you can control the rate at which tasks are sent from the queue, you can control the workers' scaling behavior and hence your costs.

Because tasks are executed as App Engine requests targeted at services, they are subject to the same stringent deadlines. Tasks handled by automatic scaling services must finish in ten minutes. Tasks handled by basic and manual scaling services can run for up to 24 hours.

TASK QUEUE

- Task queues come in two flavors, push and pull
- The manner in which the Task Queue service dispatches task requests to worker services is different for the different queues

Task Queue Overview

Contents

[Push queues and pull queues](#)

[Use cases](#)

[Push queues](#)

[Pull queues](#)

[What's next](#)

[Python](#) | [Java](#) | [P](#)

This page describes what task queues are, and when and how to use them. The Task Queue API lets applications perform work, called *tasks*, asynchronously outside of a user request. When your application needs to perform work in the background, or when an app needs to execute work in the background, it adds tasks to *task queues*. The tasks are then executed later, by scalable App Engine worker services in your application.

Push queues and pull queues

Task queues come in two flavors, *push* and *pull*. The manner in which the Task Queue service dispatches task requests to worker services is different for the different queue types.

Push queues dispatch requests at a reliable, steady rate. They guarantee reliable task execution. Because you can control the rate at which tasks are sent from the queue, you can control the workers' scaling behavior and hence your costs.

Because tasks are executed as App Engine requests targeted at services, they are subject to the same stringent deadlines. Tasks handled by automatic scaling services must finish in ten minutes. Tasks handled by basic and manual scaling services can run for up to 24 hours.

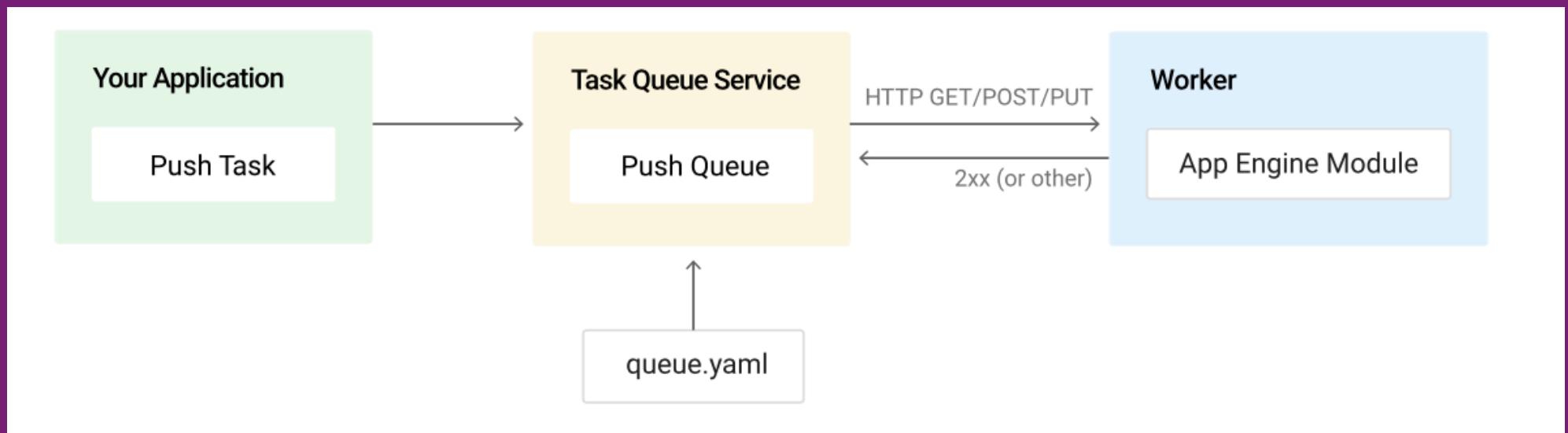
TASK QUEUE

USE CASES

- Push queue
 - Long running operations
 - Scheduled tasks
- Pull queue
 - Tasks that are interdependent
 - Related tasks that can be batched for efficiency

PUSH QUEUE

PUSH QUEUES



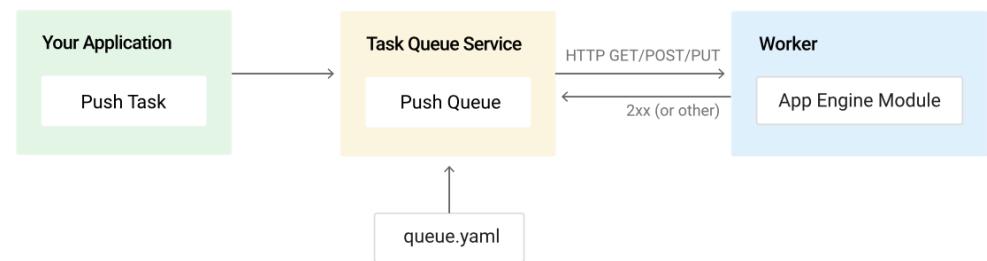
- Push queues run tasks by delivering HTTP requests to App Engine worker services

TASK QUEUE

- Push queue
 - Dispatch requests at a reliable, steady rate
 - They guarantee reliable task execution
 - You can control the workers' scaling behavior (and hence your costs)
 - Tasks handled by automatic scaling services must finish in ten minutes
 - Tasks handled by basic and manual scaling services can run for up to 24 hours

PUSH QUEUES

- Requests are delivered at a constant rate
- If a task fails, the service will retry the task, sending another request
- An HTTP response code between 200–299 indicates success
 - All other values indicate the task failed



PUSH QUEUES

- You must write a handler for every kind of task you use
- A single service can have multiple handlers for different kinds of tasks,
 - You can use different services for different task types

PUSH QUEUES

- Requirements
 - Define a queue (or be content with default)
 - Write a handler to process a task request
 - Create tasks and add them to the queue

PUSH QUEUES

CREATE A QUEUE

- You can have a maximum of 10 queues (not including the default queue)
- Optional parameters help to control quotas

queue:

Change the refresh rate of the default queue from 5/s to 1/s.

- `name: default`
`rate: 1/s`
- `name: worker-a`
`bucket_size: 40`
`max_concurrent_requests: 10`
`rate: 5/s`
- `name: worker-b`
`rate: 1/s`

PUSH QUEUES

CREATE A TASK

- Create and add a task to the "worker" queue

```
task = taskqueue.add(  
    url='/count_pictures',  
    target='worker',  
    params={'amount': amount})
```

PUSH QUEUES

CREATE A TASK

```
taskqueue.add(method=GET, url='/update-counter?key=blue', target='worker')
taskqueue.add(url='/update-counter', params={'key': 'blue'}, target='worker')
taskqueue.add(url='/update-counter', payload="{'key': 'blue'}", target='worker')
```

- Different ways of passing parameters to the task
- No difference in behavior

PUSH QUEUES

CREATE A TASK HANDLER

- Create a class to handle the task and a handler to route it

```
class EmailTaskHandler(webapp2.RequestHandler):  
    """Handler for task queue emails"""  
  
    def post(self):  
        message = self.request.get('message', default_value='default')  
        logging.info('This is an info message')  
        mail.send_mail(sender="me@uchicago-mobi-photo-timeline.appspotmail.com",  
                      to="abinkowski@uchicago.edu",  
                      subject="New Photo!",  
                      body="A new photo has been uploaded to your account.")  
  
#####  
#  
#####  
app = webapp2.WSGIApplication([  
    ('/', HomeHandler),  
    ('/email_task/', EmailTaskHandler),  
    webapp2.Route('/logging/', handler=LoggingHandler),  
    webapp2.Route('/image/<key>/', handler=ImageHandler),  
    webapp2.Route('/post/<user>/', handler=PostHandler),  
    webapp2.Route('/user/<user>/<type>/', handler=UserHandler)  
],  
    debug=True)
```

PUSH QUEUES

CREATE A TASK HANDLER

- Putting it all together
- You can test by directly accessing the URL

```
task = taskqueue.add(
    url='/email_task/',
    params={'message': 'hi'}
)
logging.debug('Task {} enqueue, ETA {}'.format(task.name, task.eta))

# Redirect to print out JSON
self.redirect('/user/%s/json/' % user)

class LoggingHandler(webapp2.RequestHandler):=

class EmailTaskHandler(webapp2.RequestHandler):
    """Handler for task queue emails"""

    def post(self):
        message = self.request.get('message', default_value='default')
        logging.info('This is an info message')
        mail.send_mail(sender="me@uchicago-mobi-photo-timeline.appspotmail.com",
                      to="abinkowski@uchicago.edu",
                      subject="New Photo!",
                      body="A new photo has been uploaded to your account.")

#####
#
#####
app = webapp2.WSGIApplication([
    ('/', HomeHandler),
    ('/email_task/', EmailTaskHandler),
    webapp2.Route('/logging/', handler=LoggingHandler),
    webapp2.Route('/image/<key>/', handler=ImageHandler),
    webapp2.Route('/post/<user>/', handler=PostHandler),
    webapp2.Route('/user/<user>/<type>/', handler=UserHandler)
],
    debug=True)
```

PUSH QUEUES

CREATE A TASK HANDLER

```
task = taskqueue.add(  
    url='/email_task/',  
    params={'message': 'hi'}  
)  
logging.debug('Task {} enqueued, ETA {}'.format(task.name, task.eta))  
  
# Redirect to print out JSON  
self.redirect('/user/%s/json/' % user)  
  
class LoggingHandler(webapp2.RequestHandler):  
  
class EmailTaskHandler(webapp2.RequestHandler):  
    """Handler for task queue emails"""  
  
    def post(self):  
        message = self.request.get('message', default_value='default')
```

DEMO

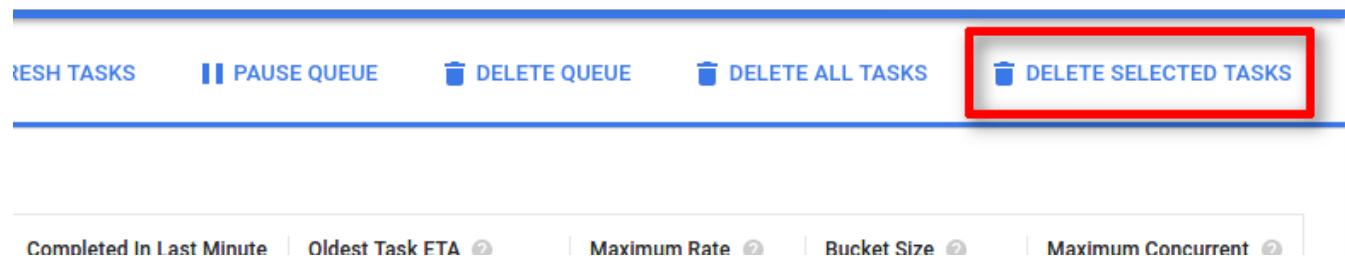
PUSH QUEUES

DELETE TASK

```
from google.appengine.api import taskqueue

# Delete an individual task...
q = taskqueue.Queue('queue1')
q.delete_tasks(taskqueue.Task(name='foo'))
```

- Delete a task from a queue



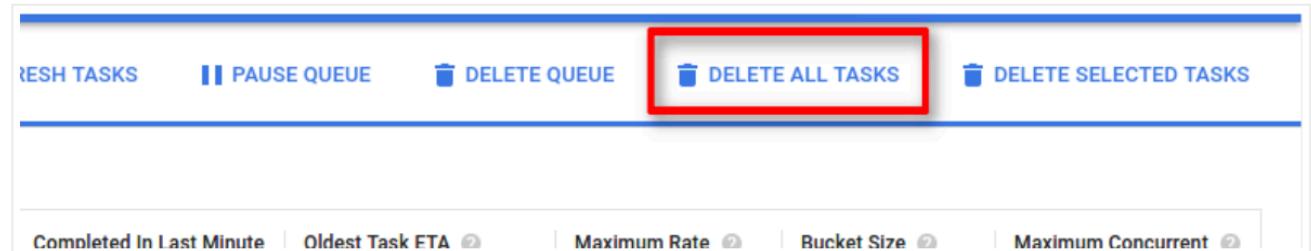
PUSH QUEUES

DELETE TASK

```
from google.appengine.api import taskqueue  
  
# Purge entire queue...  
q = taskqueue.Queue('queue1')  
q.purge()
```

- Delete all tasks from a queue

Select the task that you want to delete and click **Delete All Tasks**.



PUSH QUEUES

DEFERRED TASK

- Submit ad-hoc tasks with deferred
- No difference in how its run compared to push queue

Background work with the deferred library

Contents ▾

- Introduction
- Example: A datastore mapper
- Using the mapper
- Deferred tips and tricks

...

Nick Johnson

October 15, 2009

Introduction

Thanks to the [Task Queue API](#) released in SDK 1.2.3, it's easier than ever to do work 'offline', separate from user serving requests. In some cases, however, setting up a handler for each distinct task you want to run can be cumbersome, as can serializing and deserializing complex arguments for the task - particularly if you have many diverse but small tasks that you want to run on the queue.

Fortunately, a new library in release 1.2.5 of the SDK makes these ad-hoc tasks much easier



PUSH QUEUES

DEFERRED TASKS

```
from google.appengine.ext import deferred

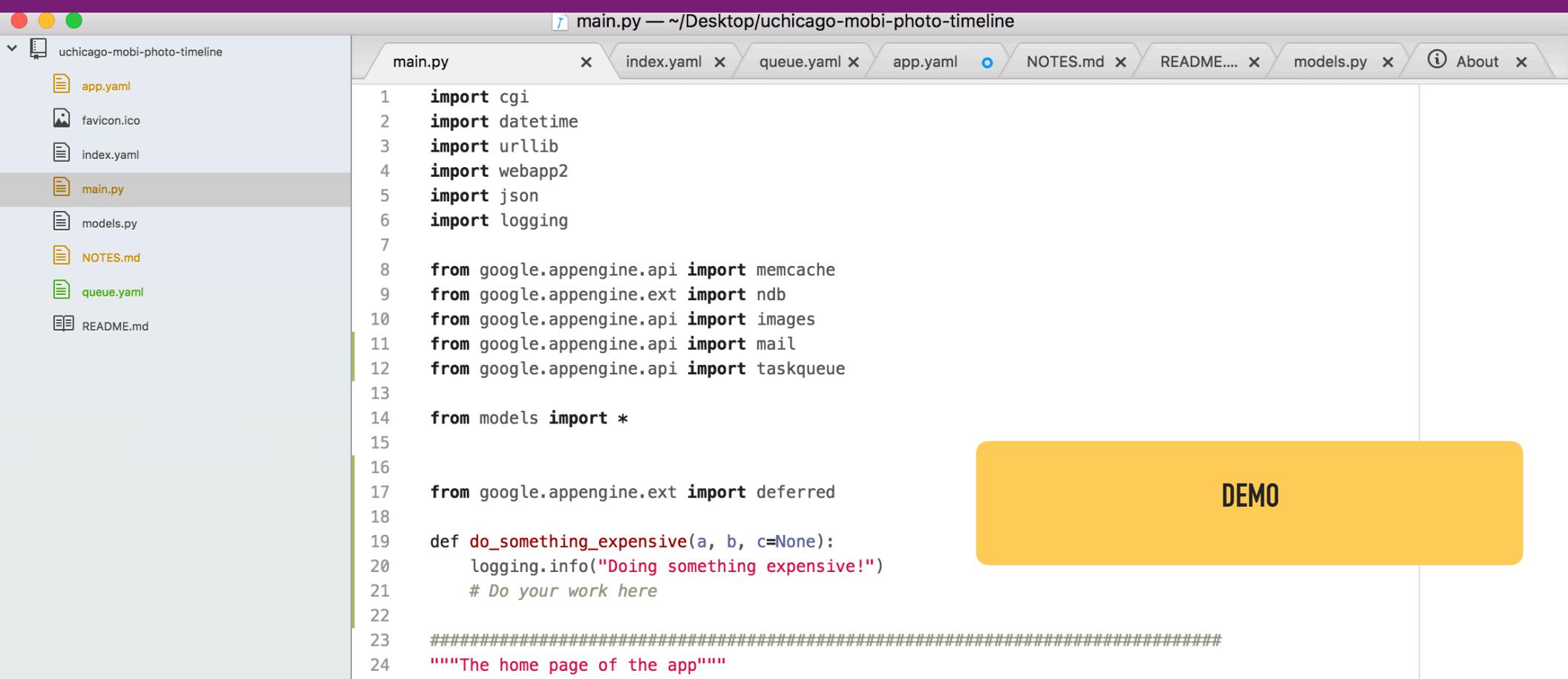
def do_something_expensive(a, b, c=None):
    logging.info("Doing something expensive!")
    # Do your work here

# Somewhere else
deferred.defer(do_something_expensive, "Hello, world!", 42, c=True)
```

- Run in the default queue

PUSH QUEUES

DEFERRED TASK



The screenshot shows a code editor window with the title "main.py — ~/Desktop/uchicago-mobi-photo-timeline". The main.py file is selected in the tabs. The code editor displays the following Python code:

```
1 import cgi
2 import datetime
3 import urllib
4 import webapp2
5 import json
6 import logging
7
8 from google.appengine.api import memcache
9 from google.appengine.ext import ndb
10 from google.appengine.api import images
11 from google.appengine.api import mail
12 from google.appengine.api import taskqueue
13
14 from models import *
15
16
17 from google.appengine.ext import deferred
18
19 def do_something_expensive(a, b, c=None):
20     logging.info("Doing something expensive!")
21     # Do your work here
22
23 #####
24 """The home page of the app"""
25
```

A yellow callout bubble with the word "DEMO" is positioned to the right of the code editor.

PULL QUEUE

PULL QUEUE

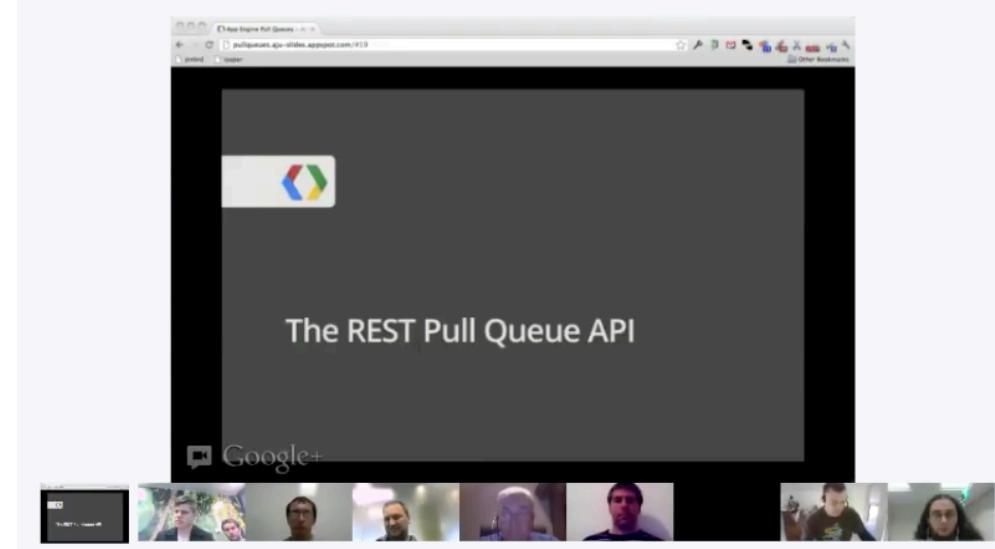
- Pull queues allow a task consumer to process tasks outside of App Engine's default task processing system
 - More customizable
- If the task consumer is a part of your App Engine app, you can manipulate tasks using simple API calls from the `google.appengine.api.taskqueue` module
- Task consumers outside of App Engine can pull tasks using the Task Queue REST API
 - More interoperability with outside services

PULL QUEUE

- Pull queue
 - Do not dispatch tasks at all
 - They depend on other worker services to "lease" tasks from the queue on their own initiative
 - Pull queues give you more power and flexibility over when and where tasks are processed, but they also require you to do more queue management
 - Tasks must complete in the specified deadline or it will be expire

TASK QUEUE

- Google Hangout video with developer relations
 - <https://cloud.google.com/appengine/docs/standard/python/taskqueue/overview-pull#>



TASK QUEUE

- In practice, start with push queues and consider pull queues when you need more control

Using Pull Queues in Python

[SEND FEEDBACK](#)

Contents ▾

- Pull queue overview
- Pulling tasks within App Engine
 - Defining pull queues
 - Adding tasks to a pull queue
- ...

Pull queues allow you to design your own system to consume App Engine tasks. The task consumer can be part of your App Engine app, such as a [service](#), or a system outside of App Engine by using the [Task Queue REST API](#). The task consumer leases a specific number of tasks for a specific duration, then processes and deletes them before the lease ends.

Using pull queues requires your application to handle some functions that are automated in push queues:

- Your application needs to scale the number of workers based on processing volume. If your application does not handle scaling, you risk wasting computing resources if there are no tasks to process; you also risk latency if you have too many tasks to process.
- Your application also needs to explicitly delete tasks after processing. In push queues, App Engine deletes the tasks for you. If your application does not delete pull queue tasks after processing, another worker might re-process the task. This wastes computing resources and risks errors if tasks are not [idempotent](#).

Pull queues require a specific configuration in `queue.yaml`. For more information, see [Defining Pull Queues](#).



DEPLOY QUEUES

DEPLOY TASK QUEUES

- Cron.yaml file

```
# cron.yaml

cron:
  - description: daily summary job
    url: /tasks/summary_email/
    schedule: every 1 minutes

  - description: weekly summary job
    url: /logging/
    schedule: every 5 minutes
```

DEPLOY TASK QUEUES

```
session2-cloud-photo-timeline — -bash — 103x17
...ion2-cloud-photo-timeline — python -m dev_appserver.py app.yaml ...
...-playground-private/session2-cloud-photo-timeline — -bash
binkowski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.cloud/mpcs51033-2017-autumn/mpcs51033-2017-autumn-playground-private/session2-cloud-photo-timeline (email_tasks)
6 % gcloud app deploy queue.yaml --project mpcs51033-2017-autumn-photos
nfigurations to update:

scriptor:      [/Users/tabinkowski/Google Drive/g-Teaching/uchicago.cloud/mpcs51033-2017-autumn/mpcs51033-2017-autumn-playground-private/session2-cloud-photo-timeline/queue.yaml]
pe:           [task queues]
rget project: [mpcs51033-2017-autumn-photos]

you want to continue (Y/n)? Y
Deploy task queue
dating config [queue]...done.

sk queues have been updated.
```

DEPLOY TASK QUEUES

≡ Google Cloud Platform mpcs51033-2017-autum... ▾

Task queues  REFRESH

Push Queues Pull Queues Cron Jobs

A cron job is a scheduled task that runs at a specific time or at regular intervals.

Cron job ^	Description	Frequency	Last run	Status	Log
/logging/	weekly summary job	every 5 minutes (GMT)		Has not run yet	View Run now
/tasks/summary_email/	daily summary job	every 1 minutes (GMT)		Has not run yet	View Run now

Task queues



SCHEDULING TASKS WITH CRON

SCHEDULING TASKS WITH CRON

- Cron Service allows you to configure regularly scheduled tasks that operate at defined times or regular intervals
 - "cron" jobs in unix
- Jobs are automatically triggered by the App Engine Cron Service

Scheduling Tasks With Cron for Python

Contents ▾

- Creating a cron job
- Testing cron jobs in the development server
- Uploading cron jobs
- Deleting all cron jobs

...

Pyth

The App Engine Cron Service allows you to configure regularly scheduled tasks that operate at regular intervals. These tasks are commonly known as *cron jobs*. These cron jobs are automatically triggered by the App Engine Cron Service. For instance, you might use a cron job to send out an email every day, or to update some cached data every 10 minutes, or refresh summary information once a week.

A cron job invokes a URL, using an HTTP `GET` request, at a given time of day. A cron job runs under the same limits as those for [push task queues](#).

Creating a cron job

1. Create the `cron.yaml` file in the root directory of your application (alongside `app.yaml`).
2. Add one or more `<cron>` entries to your file and define the necessary elements for each required `<url>` and `<schedule>` elements.

The following example creates a basic cron job that runs daily:

```
crontab:  
- description: daily summary job
```

SCHEDULING TASKS WITH CRON

- Use cases
 - Send out an email report on a daily basis
 - Update cached data every 10 minutes
 - Refresh summary information once an hour

Scheduling Tasks With Cron for Python

Contents ▾

- Creating a cron job
- Testing cron jobs in the development server
- Uploading cron jobs
- Deleting all cron jobs

...

Pyth

The App Engine Cron Service allows you to configure regularly scheduled tasks that operate at regular intervals. These tasks are commonly known as *cron jobs*. These cron jobs are automatically managed by the App Engine Cron Service. For instance, you might use a cron job to send out an email message or to update some cached data every 10 minutes, or refresh summary information once a day.

A cron job invokes a URL, using an HTTP `GET` request, at a given time of day. A cron job runs under the same limits as those for [push task queues](#).

Creating a cron job

1. Create the `cron.yaml` file in the root directory of your application (alongside `app.yaml`).
2. Add one or more `<cron>` entries to your file and define the necessary elements for each required `<url>` and `<schedule>` elements.

The following example creates a basic cron job that runs daily:

```
crontab:  
- description: daily summary job
```

SCHEDULING TASKS WITH CRON

- A cron job invokes a URL, using an HTTP GET request, at a given time of day
- Job request is subject to the same limits as those for push task queues

Scheduling Tasks With Cron for Python

Contents ▾

- Creating a cron job
- Testing cron jobs in the development server
- Uploading cron jobs
- Deleting all cron jobs

...

Pyth

The App Engine Cron Service allows you to configure regularly scheduled tasks that operate at regular intervals. These tasks are commonly known as *cron jobs*. These cron jobs are automatically managed by the App Engine Cron Service. For instance, you might use a cron job to send out an email once a day or to update some cached data every 10 minutes, or refresh summary information once a week.

A cron job invokes a URL, using an HTTP GET request, at a given time of day. A cron job runs under the same limits as those for [push task queues](#).

Creating a cron job

1. Create the `cron.yaml` file in the root directory of your application (alongside `app.yaml`).
2. Add one or more `<cron>` entries to your file and define the necessary elements for each entry, including required `<url>` and `<schedule>` elements.

The following example creates a basic cron job that runs daily:

```
crontab:  
- description: daily summary job
```

SCHEDULING TASKS WITH CRON

- Create a cron job
 - Add a cron.yaml file
 - Add entries
 - Create a handler for cron jobs URL

```
cron:  
  - description: daily summary job  
    url: /tasks/summary  
    target: beta  
    schedule: every 24 hours
```

SCHEDULING TASKS WITH CRON

`cron:`

`- description: daily summary job`

`url: /tasks/summary`

`#target: beta`

`schedule: every 1 minutes`

SHOWS IN THE CONSOLE

HANDLER

RUNS ON DEFAULT BY DEFAULT.
OTHERWISE SPECIFY A TARGET OR
VERSION

SCHEDULING TASKS WITH CRON

every 12 hours

every 5 minutes from 10:00 to 14:00

every day 00:00

every monday 09:00

2nd,third mon,wed,thu of march 17:00

1st monday of sep,oct,nov 17:00

1 of jan,april,july,oct 00:00

- Schedule format

SCHEDULING TASKS WITH CRON

cron:

- description: "daily summary job"
url: /tasks/summary
schedule: every **24** hours
- description: "monday morning mailout"
url: /mail/weekly
schedule: every monday **09:00**
timezone: Australia/NSW
- description: "new daily summary job"
url: /tasks/summary
schedule: every **24** hours
target: beta

RUNS ON DEFAULT BY DEFAULT.
OTHERWISE SPECIFY A TARGET OR
VERSION

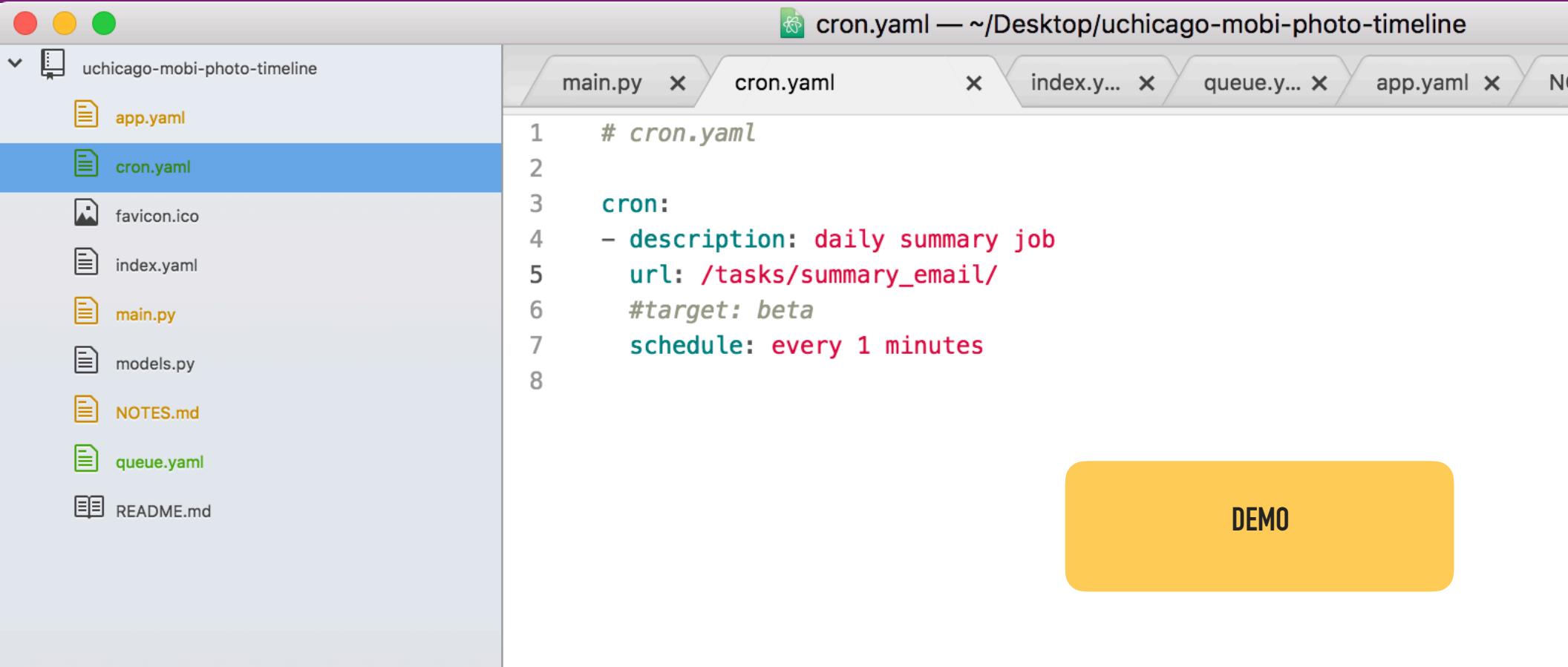
SCHEDULING TASKS WITH CRON

- The handler should execute any tasks that you want scheduled
- The handler should respond with an HTTP status code between 200 and 299 (inclusive) to indicate success
- Other status codes can be returned and can be used to trigger retrying the job

cron:

```
- description: daily summary job
  url: /tasks/summary
  target: beta
  schedule: every 24 hours
```

SCHEDULING TASKS WITH CRON



The screenshot shows a terminal window with the title "cron.yaml — ~/Desktop/uchicago-mobi-photo-timeline". The window displays the contents of the "cron.yaml" file, which defines a cron job for daily summaries:

```
# cron.yaml
cron:
- description: daily summary job
  url: /tasks/summary_email/
  #target: beta
  schedule: every 1 minutes
```

The file is located in a directory named "uchicago-mobi-photo-timeline" which contains other files like "app.yaml", "index.yaml", and "main.py". A yellow button labeled "DEMO" is overlaid on the bottom right of the terminal window.

DEPLOY

PUSH QUEUES

- Cron.yaml file

```
# cron.yaml

cron:
  - description: daily summary job
    url: /tasks/summary_email/
    schedule: every 1 minutes

  - description: weekly summary job
    url: /logging/
    schedule: every 5 minutes
```

PUSH QUEUES

```
tabinkowski@Ts-MacBook-Pro ~/Google Drive/g-Teaching/uchicago.cloud/mpcs51033-2017-autumn-playground-private/session2-cloud-photo-timeline (email_tasks)
[513 % gcloud app deploy cron.yaml --project mpcs51033-2017-autumn-photos
Configurations to update:
```

```
descriptor:      [/Users/tabinkowski/Google Drive/g-Teaching/uchicago.cloud/mpcs51033-2017-autumn-playground-private/session2-cloud-photo-timeline/cron.yaml]
type:           [cron jobs]
target project: [mpcs51033-2017-autumn-photos]
```

**Deploy cron
jobs for project**

Do you want to continue (Y/n)? █

PUSH QUEUES

≡ Google Cloud Platform mpcs51033-2017-autum... ▾

Task queues REFRESH

Push Queues Pull Queues Cron Jobs

A cron job is a scheduled task that runs at a specific time or at regular intervals.

Cron job ^	Description	Frequency	Last run	Status	Log
/logging/	weekly summary job	every 5 minutes (GMT)		Has not run yet	View Run now
/tasks/summary_email/	daily summary job	every 1 minutes (GMT)		Has not run yet	View Run now

Cron jobs



ASSIGNMENT 2

Assignment 2

As discussed in class, there are some potential issues with our photo timeline backend. In the first part of the assignment you will update our existing code base ([available here on Github](#)) with the following changes.

Update the Data Model



THE UNIVERSITY OF
CHICAGO



MPCS 51033 • AUTUMN 2017 • SESSION 2

BACKENDS FOR MOBILE APPLICATIONS