

Managing Large Flask Applications on Google App Engine (GAE)

Emmanuel Olowosulu & Yemisi Obielodan



Emmanuel Olowosulu

CTO, Formplus

 @seyisulu



- **Over 8 years experience consulting and building production apps with a variety of stacks and technologies.**

Yemisi Obielodan

Product Manager, Formplus

 @ye_misi



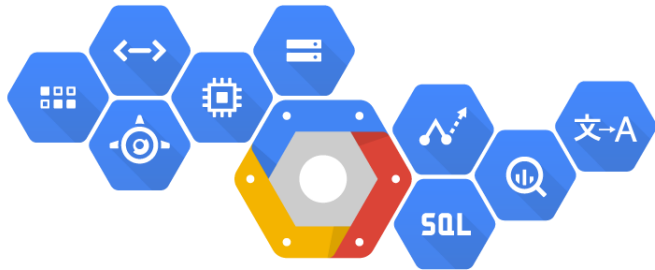
- **Also, software developer with experience building Flask applications and community builder.**
-



- **Formplus** is a cloud-based data collection software as a service. We help businesses gather data from their users who are online and offline, analyze the data and then help them make data-driven decisions from it.
- Also a fun place to work.



Our Stack



Google Cloud Platform

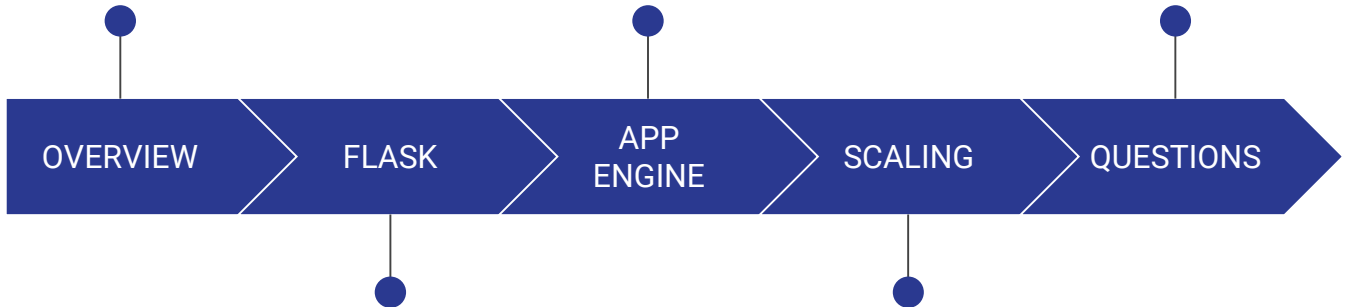
- **Python 2.7**
- **Flask**
- **Google App Engine**
- **Cloud Datastore**
- **Cloud Tasks**
- **AngularJS 1.5**
(don't judge 😊)

What We Will Be Covering

Large applications,
Flask and Google App
Engine

App Engine, Cloud
Datastore and the
Google Cloud Platform

?



Flask application
structure, blueprints
and request routing

Performance, effective
GCP utilization,
problems

Overview

Large applications, Flask and Google App Engine

What is a Large Application?



- **Large number of users**
- **High requests/second**
- **Distributed servers**
- **Large data volume**

Flask Framework



- **Flask is a micro web framework written in Python**
- **Supports extensions for additional functionality**
- **Compatible with Google App Engine**

Google App Engine



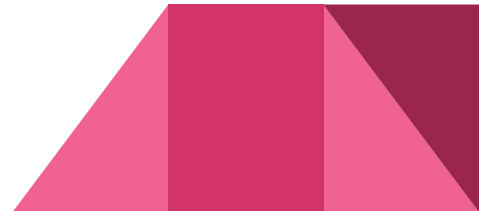
- **GAE is a web framework and cloud computing platform for developing and hosting web applications in Google-managed data centers**
 - **Applications are sandboxed and run across multiple servers**
-

Flask

Flask application structure, blueprints and request routing

Flask Application Structure

- Large Flask apps should be broken down into microservices. There should be at least two: the frontend handling user facing requests and backend handling longer running data and event processing.
- This ensures that the application is responsive.
- Also, there is nothing micro about microservices; the frontend and backend may be monoliths themselves and could benefit from refactoring.



Flask Structure w/Blueprints

- In the structure shown, each directory represents a microservice
- In each microservice, group related functionality using Blueprints: the blog microservice has frontend and backend Blueprints
- App Engine handles requests using rules contained in dispatch.yaml

```
app/  
├─ admin/  
│   └─ %<--  
├─ tasks/  
│   └─ %<--  
├─ blog/  
│   ├── backend/  
│   │   └─ %<--  
│   ├── frontend/  
│   │   ├── assets/  
│   │   ├── static/  
│   │   ├── templates/  
│   │   ├── __init__.py  
│   │   ├── controllers.py  
│   │   ├── models.py  
│   │   └─ %<--  
│   ├── __init__.py  
│   ├── app.yaml  
│   ├── appengine_config.yaml  
│   └─ main.py  
├─ site/  
│   └─ %<--  
├─ dispatch.yaml  
└─ %<--
```

Flask Blueprints

Flask app

Frontend
blueprint

App Engine
dispatch file

```
# blog/main.py - Using Blueprints
from backend.controllers import back_app
from frontend.controllers import front_app

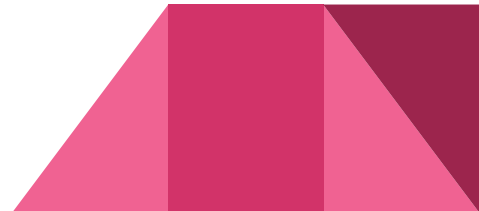
app = Flask(__name__)
app.config.update(SECRET_KEY=config.SECRET_KEY)
app.register_blueprint(back_app, url_prefix='/blog/admin')
app.register_blueprint(front_app, url_prefix='/blog')
# %<--

# blog/frontend/controllers.py
front_app = Blueprint(
    'front', __name__, template_folder='templates')
# %<--

# dispatch.yaml
dispatch:
  - url: "*/admin*"
    service: admin
  - url: "*/blog*"
    service: blog
  - url: "*/tasks*"
    service: tasks
```

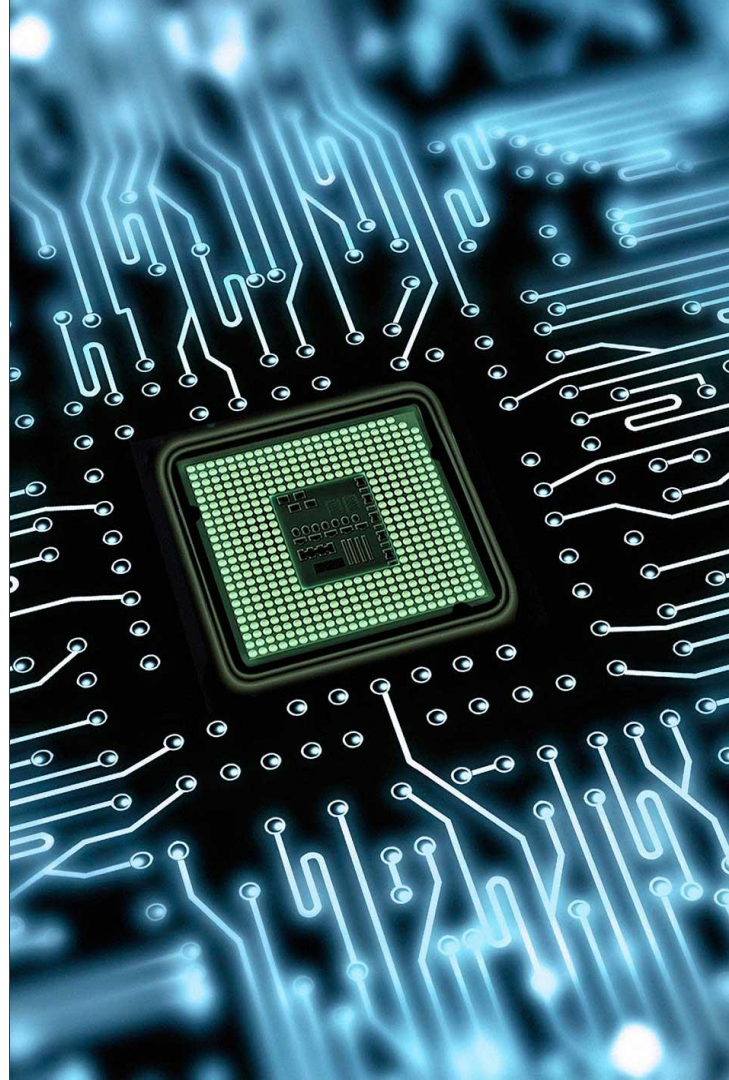
Monorepo vs Multirepo

- **Microservice purist may insist on having different repositories for each microservice but we have had success with a Monorepo.**
- **There are pros and cons associated with both approaches and a Multirepo is better suited for large teams each working on distinct microservices.**
- **Also, Conway's Law: software ends up "shaped like" the organizational structure it's designed in.**



Conway's Law

organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations.



Request Routing

In Flask there are two ways to create routes: class based and function based. A lot of people are familiar with the latter but using the class based method can lead to better code organization as you can use inheritance to share functionality between routes.

```
from flask import jsonify
from flask.views import MethodView

blue = Blueprint('front', __name__)

# Class
class Users(MethodView):
    decorators = [your_decorator]
    def get(self):
        posts = BlogPost.query().order(
            -BlogPost.date).q.fetch(9)
        return jsonify(posts=posts)

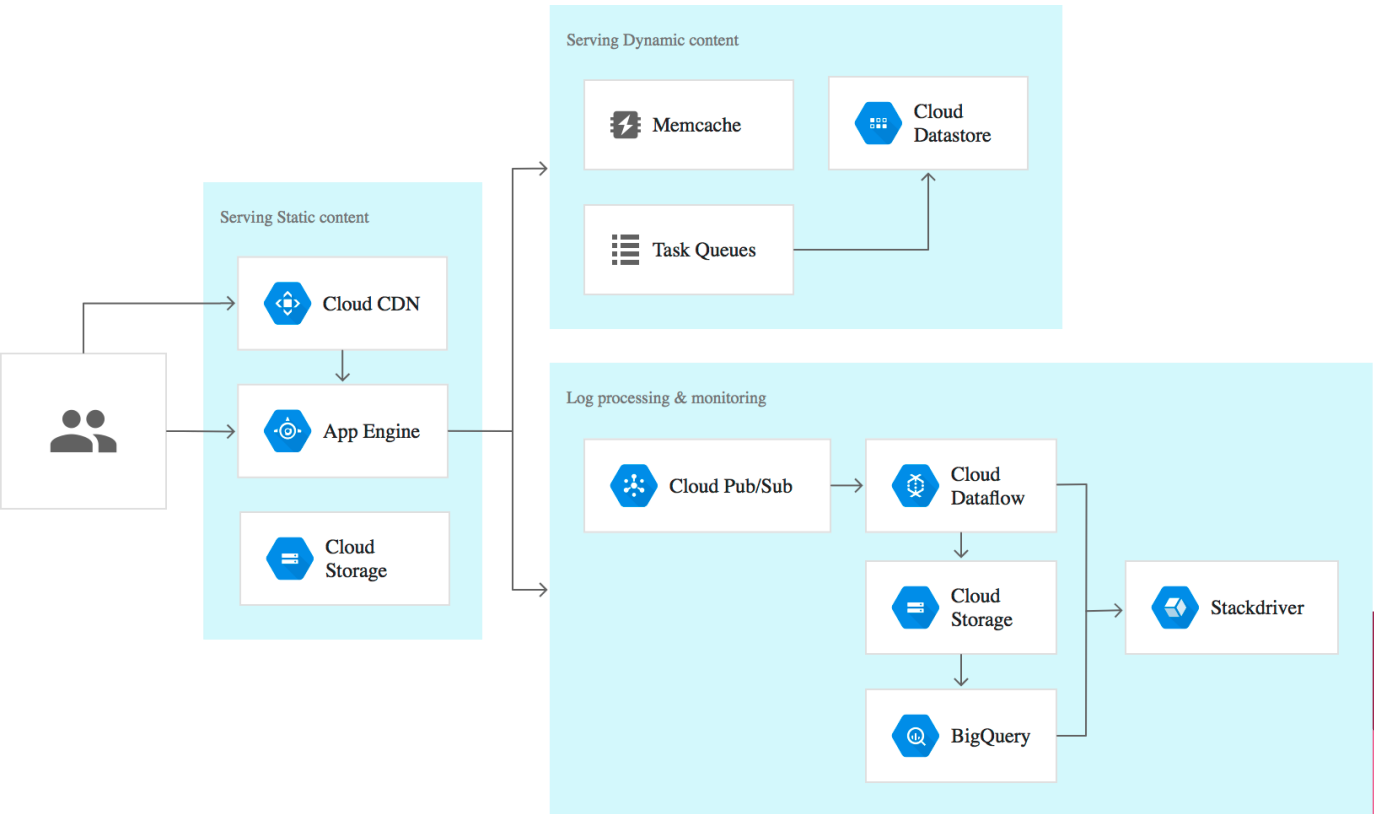
blue.add_url_rule(
    '/posts',
    view_func=Users.as_view('users'))

# Function
@your_decorator
@blue.route("/users", methods=["POST"])
def get_users():
    posts = BlogPost.query().order(
        -BlogPost.date).q.fetch(9)
    return jsonify(posts=posts)
```


App Engine

App Engine, Cloud Datastore and the Google Cloud Platform

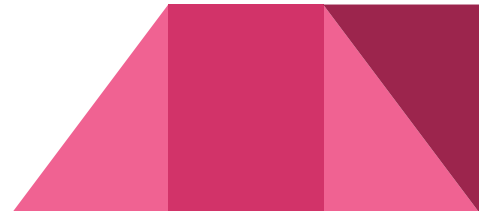
App Engine & Google Cloud Platform



Cloud Storage



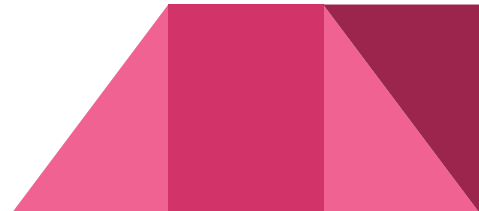
- **Google Cloud Storage** is file storage solution intended for developers and system admins and has APIs allowing apps to store and retrieve objects. It features global edge-caching, versioning, OAuth and granular access controls, configurable security, etc.



Cloud CDN



- **Google Cloud CDN (Content Delivery Network) uses Google's servers worldwide to load balanced content close to your users. This provides faster delivery of content to your users and reduces serving costs.**
- **App Engine also automatically pushes application assets to Cloud CDN on deployment.**



Cloud Tasks

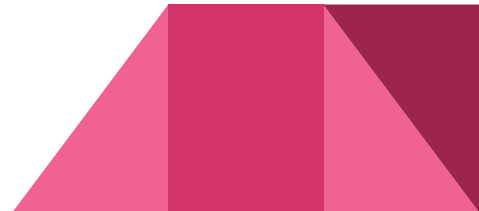


- **Task queues let applications do work, asynchronously outside of a user request. If an app needs to execute work in the background, it adds tasks to task queues. The tasks are executed later, by worker services. Task queues come in two flavors, push and pull.**
- **Push queues run tasks by delivering HTTP requests to App Engine worker services.**
- **Pull queues rely on other worker services to fetch tasks from the queue on their own.**

Cloud Pub/Sub



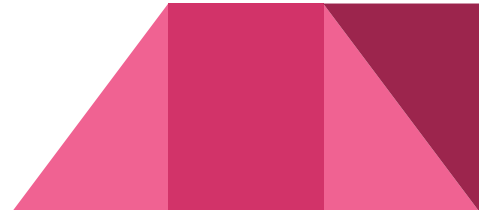
- **Cloud Pub/Sub is a fully-managed real-time messaging service that allows you to send and receive messages between independent applications.**
- **Pub/Sub broadcasts messages from publishers to subscribers via channels. Many subscribers can subscribe to the same channel.**
- **Allows the decoupling of background, long running data and event processing from user-facing requests.**



Cloud Dataflow



- **Cloud Dataflow is a fully-managed service for transforming and enriching data in stream (real time) and batch (historical) modes. Cloud Dataflow seamlessly integrates with GCP services for streaming events ingestion (Cloud Pub/Sub), data warehousing (BigQuery), and more.**



Google BigQuery






- **BigQuery is Google's serverless, highly scalable, enterprise cloud data warehouse. It is fast, cost-effective, fully managed and can be used for analytics. It also has built-in machine learning (beta).**
- **BigQuery is used for Online Analytical Processing (OLAP): generating reports, time series and trend analysis views and can be paired with Business Intelligence (BI) tools like Metabase or Google Data Studio (beta).**

Cloud Datastore



- **Cloud Datastore is a highly-scalable NoSQL database for your web and mobile applications.**
- **Datastore is to App Engine what MongoDB is to the MEAN stack and to App Engine what MySQL is to the LAMP stack.**
- **It is a highly-scalable NoSQL database for your applications. Cloud Datastore automatically handles sharding and replication, providing you with a highly available and durable database that scales automatically.**

Datastore vs. MongoDB vs. MySQL

Datastore		MongoDB	MySQL
<ul style="list-style-type: none">○ Category of object: Kind○ One object: Entity○ Individual object data: Property○ Unique ID for object: Key		<ul style="list-style-type: none">○ Category of object Collection○ One object Document○ Individual object data Field○ Unique ID for object Key	<ul style="list-style-type: none">○ Category of object: Table○ One object: Row○ Individual object data: Column○ Unique ID for object: Primary Key
			
NoSQL			SQL

Merits of Using App Engine

App Engine is similar to Heroku

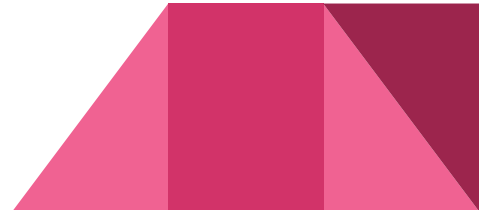


- **Platform as a Service (PaaS)**
 - **Provide abstracted machine instances to run your code**
 - **Prevent you from worrying about ops: infrastructure upgrades, software updates and security patches**
 - **Automatic load balancing, scaling and fault tolerance**
-

Google App Engine



- **Task queues let applications do work, 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 worker services. Task queues come in two flavors, push and pull.**
- **Push queues run tasks by delivering HTTP requests to App Engine worker services.**
- **Pull queues rely on other worker services to fetch tasks from the queue on their own.**



Scaling

Performance, effective GCP utilization, problems

He who thinks the
soup is too much for the
eba cannot handle
greatness scale.

- Pete Edochie



Problem 1: Out of Memory Errors

Background

Some tasks are unable to be completed because they ran out of memory. One of what we've found causes this error is communicating with Sheets.

Cause

A library used consumed ~30MB per instance when created and instances cannot be reused across requests. This resulted in OOM Errors when serving a lot of requests to update Google Sheets.

Solution

Cloud Functions

We recently refactored the part of our code that updates users spreadsheets to use a Cloud Function. This means the additional memory used does not affect subsequent requests.

There are only two hard things in Computer Science: cache invalidation and naming things.

- Phil Karlton

```
use_y = False
mod.use_z = False
tion == "MIRROR_Y":
mod.use_x = False
mod.use_y = True
mod.use_z = False
tion == "MIRROR_Z":
mod.use_x = False
mod.use_y = False
mod.use_z = True

ion at the end -add
elect= 1
.select=1
.scene.objects.activ
```


Problem 2: Cache Invalidation

Background

Users not having their monthly submissions quota reset

Cause

When updating a large number of entities in Cloud Datastore, the recommended approach is to use Datastore Migrations Pipelines. Batching Datastore writes for efficiency bypassed Memcache and that stale data could potentially overwrite the Datastore update.

Solution

Skip the in-process cache but update Memcache and Datastore. Use transactions when modifying entities likely to be updated in the datastore but outdated in the Memcache.

Datastore Contention



Problem 3: Datastore Contention

Background

Updating form submission count for high traffic forms

Cause

A job was created on the submissions task queue for each submission and they all tried to update the same entity at the same time.

Solution

When the submission comes in, a task is created named with the form ID and the time of submission and then we schedule it to run in future.

We combine this with an entity in datastore that holds the submission status for that task to batch updates.

Questions

???