Mobile Solutions on Google Cloud Platform

Introduction

With over a billion people using smartphones and tablets in their daily lives, there is a huge opportunity and growing demand for mobile solutions. Users can choose from hundreds of thousands of applications, so in order for mobile developers to be successful, it is important to create compelling, engaging and connected user experiences. This typically requires a need for backend components or services to feed the application with relevant data, provide push notification, and allow interactions between users, and more.

With Google Cloud Platform you can easily build a backend for your mobile solution. You may be starting small and hoping to get a lot of users or you may already have a large customer base that you want to start serving through a new mobile experience. In either case, you can develop a backend that scales to meet the growing demand. By leveraging Google infrastructure, you can focus on implementing your application's scenarios and not have to worry about things such as managing machines or load balancing.

Requirements

Typical requirements for a compelling mobile solution include:

- Support for Android and iOS devices through native applications
- Storage, retrieval, and processing data outside of mobile devices
- Orchestrating push notification to Android and IOS devices
- Geo-location awareness and geo-proximity search
- User authentication
- High scalability

You can easily meet these requirements when you build your mobile solution on Google Cloud Platform.

Scope

This paper presents the key components of a mobile solution built on Google Cloud Platform. It includes both components that are common to most mobile solutions and optional components that might be useful in more specific contexts.

This paper is intended for architects and developers interested in building their own backend and designing the communications between mobile clients and the backend. Building the actual mobile clients is not covered in this document beyond information relevant to communicating with the backend, authenticating users and receiving push notification.

High-Level Overview

The key components in the proposed mobile solution architecture are:

- Your mobile clients - any combination of Android, iOS and HTML/JavaScript clients.
- Your mobile backend application which is responsible for serving requests from the clients.
- The communication layer between the mobile clients and the backend.
- Various cloud services for storing your application data, delivering push notification, etc.

This document walks you through the steps required for assembling a mobile solution step by step, leading to a complete architectural diagram presented at the end.

Your Mobile Backend in the Cloud [1]

The ability to have your own backend means that you can run your custom code on the backend and not be limited to only calling 3rd party services from your mobile client applications. By leveraging Google Cloud Platform, you can easily have your own backend in the cloud. Google App Engine is an ideal platform for running your code, especially code that can be called from your mobile client devices.

Primary role of your custom code in the backend

The approach for building the backend as presented in this document allows you to focus on developing your core scenarios. Your two main tasks are defining your domain specific Resources and implementing custom logic needed for specific operations on these Resources. Your mobile clients will be able to easily access these Resources and invoke these operations through your API that can be easily exposed from your mobile backend as described later.

Your mobile backend may retrieve some of Resources from external services without storing them within your application. But typically your mobile solution will have at least some Resources that are managed and persisted as part of your solution.

For example, if you are building an application that includes reviews, you can define Review as one of your Resources. If your application allows mobile shopping, you may include reviews of products in the inventory. If your application lists movies, you may include movie reviews. If you are developing a casual game you may define CompletedPlay as an entity with attributes such as score, date, user id, and more.

Then you can define operations on these Resources such as inserting a new review, adding a usefulness rating to a review, retrieving the reviews sorted by their usefulness rating or retrieving top scores for a day.

[1]
In this document, a backend refers broadly to the part of a mobile solution that runs outside of client devices. It should not be confused with Google App Engine backend instances, which represent a type of instances that have different characteristics than default App Engine instances. In many mobile solutions, default Google App Engine instances should be sufficient. However, if you have specific requirements, you can assemble a mobile solution using a "backend" composed of any combination of default App Engine instances and backend instances.

Depending on your scenario and requirements, the implementation of these operations and your custom code won't always be trivial. Your code may also do a lot of things besides synchronously handling simple API requests from the clients. You may want to do background processing, geo-proximity based searches, or push notification, all of which are possible when running your mobile backend on Google Cloud Platform. As a result, you can focus on writing your application-specific code and not infrastructure code.

What you don't have to worry much about
Running your mobile backend code on Google App Engine, according to the recommended design, offers you the following benefits of Google Infrastructure:

- Automatic scalability
- Automatic load balancing across instances of your mobile backend
- Static content serving and caching
- Reliability, performance, and security features
- Monitoring and management
- Configurable DoS protection
- Integration with Google Accounts for OAuth2 authentication (for both Android and iOS clients)

Communication between your mobile clients and your mobile backend

There are many possible patterns for communication between mobile clients and the backend. However, over the last decade REST-based model has emerged as a predominant one.

Google Cloud Platform offers a powerful new technology called Google Cloud Endpoints [2] that simplifies, not only exposing REST API from your mobile backend, but also consuming REST APIs from Android, iOS, and JavaScript clients. It also provides OAuth2-based authentication, so your mobile backend code can know the identity of the caller. Cloud Endpoints service leverages Google infrastructure so you can count on it to scale to meet the demand. You can also use tooling support to have necessary boilerplate code auto-generated, so you can focus on writing code specific to your application scenarios.

[2]
As of February 2013, Google Cloud Endpoints is an experimental feature.

Please follow Google Cloud Endpoints documentation for information on how to annotate your application code to have your API exposed through Google Cloud Endpoints, how to configure OAuth2 authentication, and how to generate strongly typed Android and iOS clients as well as lightweight JavaScript clients. If you use Eclipse, you can benefit from additional integrated tooling support provided by Google Plugin for Eclipse.

Although you can build your mobile backend without using Google Cloud Endpoints (and many parts of this document would still be applicable), the proposed solution assumes the usage of Google Cloud Endpoints.

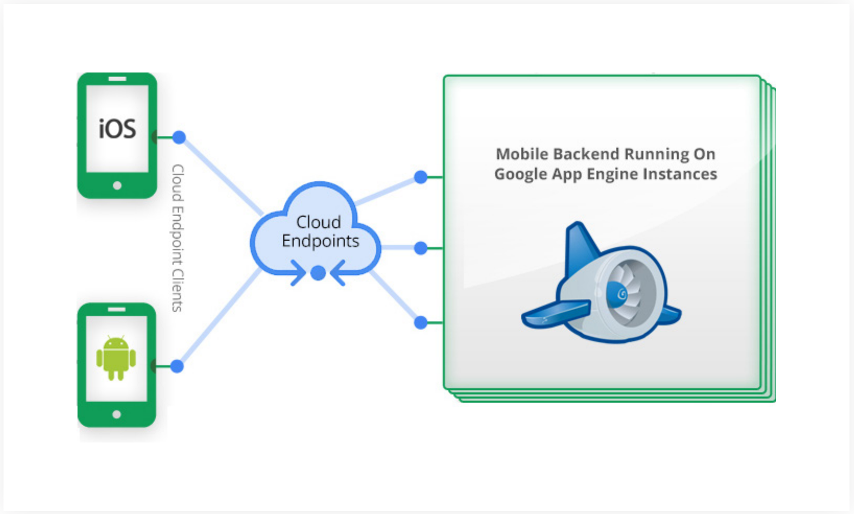
Assembling Your Mobile Solution

The basics

As mentioned in previous sections, the essential components (Fig. 1) in the proposed mobile solution architecture are:

1. Android and/or iOS mobile clients.
2. Google Cloud Endpoints used for communications between the clients and the backend over REST API with optional OAuth2 authentication.
3. Your mobile backend application code running on Google App Engine and responsible for serving requests from the clients.

Figure 1.
Essential components in proposed Mobile Solutions architecture



Storing data

A typical requirement for a mobile solution with a backend is to store data outside of client devices [3]. This data can be categorized into two groups (Fig. 2):

[3]
Section 4.2 of Google App Engine Terms of Service describes your obligation to protect the privacy and legal rights of your End Users, including the requirement to provide a legally adequate privacy policy about the use of data you collect and to obtain all necessary consents from your end users to the collection, use, monitoring or disclosure of such data.

1. Large, and typically binary, objects such as images

These objects can be a part of your solution, for example, product images for a dynamic inventory of gifts offered by your mobile application. Or, you may also want to allow your users to upload their pictures, avatars, and other images. In both cases, for storing this kind of data you can use Google Cloud Storage, a service that stores objects and files up to terabytes in size. See also "Serving and processing images" below.

2. Fine grained properties and entities

These properties and entities might include information about the last completed level in a mobile game, information about users, user's device registration for push notification, records of in-app purchases, or other similar application-specific data.

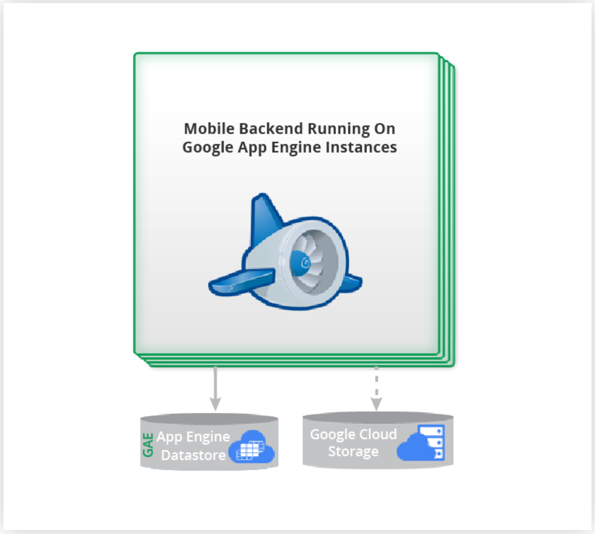
The properties of these entities can also include a reference, for example, object name and optionally bucket name or URL, to objects stored in Google Cloud Storage.

With your mobile backend hosted on Google App Engine, a natural place to store this kind of data is App Engine Datastore. It provides a NoSQL [4] schemaless object data store, with a query engine and atomic

[4]
If you prefer to use relational SQL database you can choose to use Google Cloud SQL either instead of Datastore or complementary to Datastore.

transactions. These entities will often map to the Resources exposed over Google Cloud Endpoints API. When using appropriate annotations you can leverage Google Plugin for Eclipse to generate skeleton code for typical list/get/insert/update/remove operations using App Engine Datastore.

Figure 2.
Storing Data

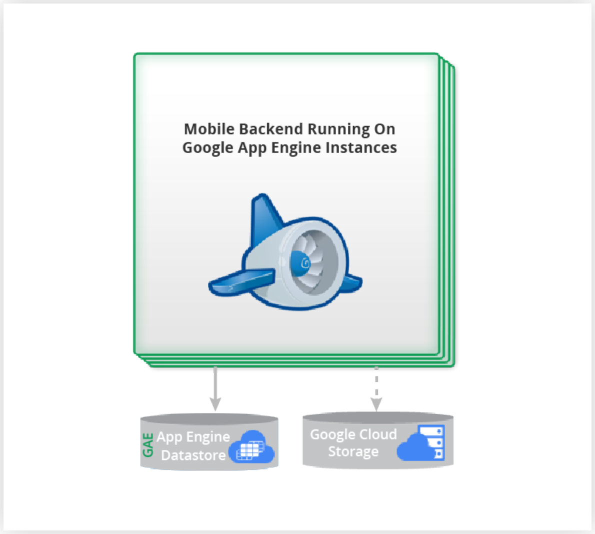


Optimizing data access with Memcache

If your mobile clients are making frequent requests to the backend to retrieve the same data, it wouldn't be optimal to retrieve this data from persistent storage for every request. For example, if you are building a mobile application that shows movie reviews, it is likely that most users would be retrieving reviews about one of a handful movies that are currently in theaters and there will be significantly fewer requests for reviews of the remaining thousands of movies that the backend stores in a persistent storage.

A backend running on Google App Engine can use a distributed in-memory data cache service called Memcache to store and retrieve frequently accessed data (Fig. 3). Using Memcache not only allows your mobile solution to have higher performance and to scale better, but also reduces your cost of accessing App Engine Datastore as your application will be sending fewer requests to App Engine Datastore.

Figure 3.
Optimizing data access with Memcache



Asynchronous processing with Task Queues

Sometimes it is not possible, or desirable, to do all the work associated with a request sent from a mobile client before the response needs to be sent back to the user's device. For example, let's say you are developing a social application that notifies user's friends when the user checks into a restaurant. You may want to record the check-in occurrence synchronously, together with recording the fact that the notification needs to be processed, and then send the acknowledgment response to the client immediately without waiting until all notifications are processed.

Your code running on Google App Engine can use a service called Task Queues to enqueue information about work that needs to be done asynchronously (i.e., "tasks"). In the example above, your application could enqueue a task to notify Joe's friends that Joe checked into Trendy Perks restaurant at 7:35 pm.

Typically, you would configure Task Queue service to send the enqueued tasks (push model) to your code running on Google App Engine as HTTP requests to a chosen URL. Your application can process these requests through regular web request handlers. In the example above, your request handler could retrieve the task information from the HTTP request, look up Joe's friends, and then notify each of the friends by sending a push notification. You can configure the tasks to be processed by the same Google App Engine instance(s) that are processing requests from mobile clients or by dedicated instance(s).

Alternatively, your code can pull the tasks from the queue(s). This can be done by either your mobile backend code running on Google App Engine, or by your code running outside of App Engine, which can be used to process tasks by Google Compute Engine instances.

In either case, processing tasks is typically the second most common responsibility your custom code running on Google App Engine would have, in addition to processing requests from mobile clients over Google Cloud Endpoints, as mentioned in previous sections.

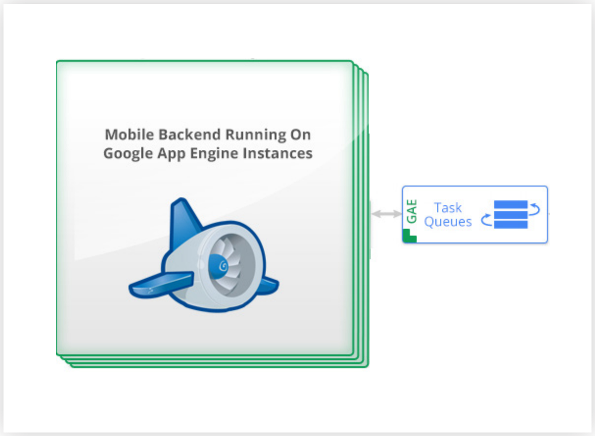


Figure 4. Asynchronous processing with Task Queues

Push notification

Push notification is very important to the overall user experience on mobile devices. Depending on the client platform, your mobile solution can use this technology to display "toasts" and other notifications on users' devices, even if your mobile application has not been running. It can also be used to keep feeding the mobile client application with relevant data.

With your mobile backend code running on Google App Engine, you can orchestrate sending push notifications to your users by leveraging the following technologies:

- Google Cloud Messaging (GCM) - a Google service that allows you to send data to your users that have your client application installed on their Android-powered devices. If you use Eclipse you can have skeleton code generated by Google Plugin for Eclipse.
- Sockets API [5] to send push notification using Apple® Push Notification Service to your iOS users.

[5]
As of early January 2013 Sockets API is a feature available for Trusted Testers only and doesn't have production support.

An example of this might be to send push notifications to the users of your mobile solution when their friend checks into a restaurant or posts a review (Fig. 5).

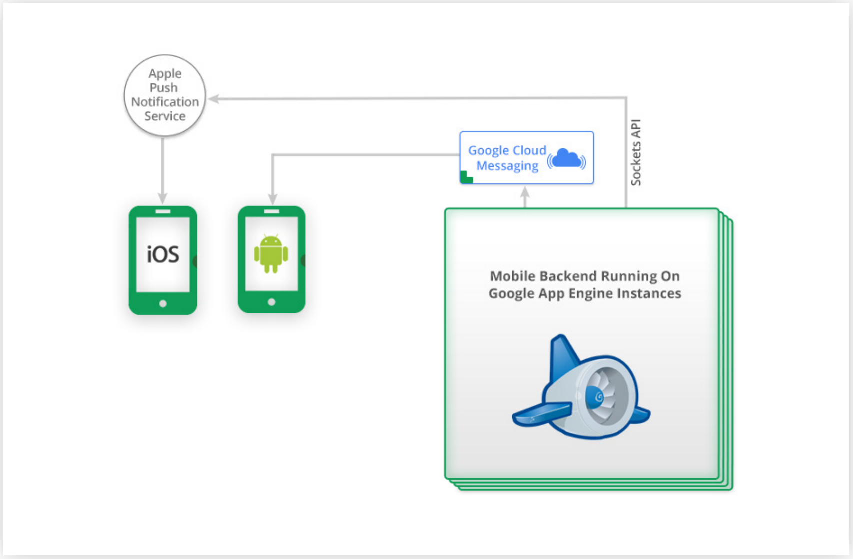

Apple Push Notification Service
Google Cloud Messaging
Sockets API
iOS
Mobile Backend Running On Google App Engine Instances

Figure 5.
Push Notification

Serving static content

In some cases, you may also benefit from packaging static content as part of your Google App Engine application and then having it served by Google App Engine infrastructure using dedicated servers and caches without having your mobile backend code involved at all. This is typically more relevant to HTML/JavaScript clients than to native iOS or Android applications and is applicable to resources such as images, CSS style sheets, or browser JavaScript code.

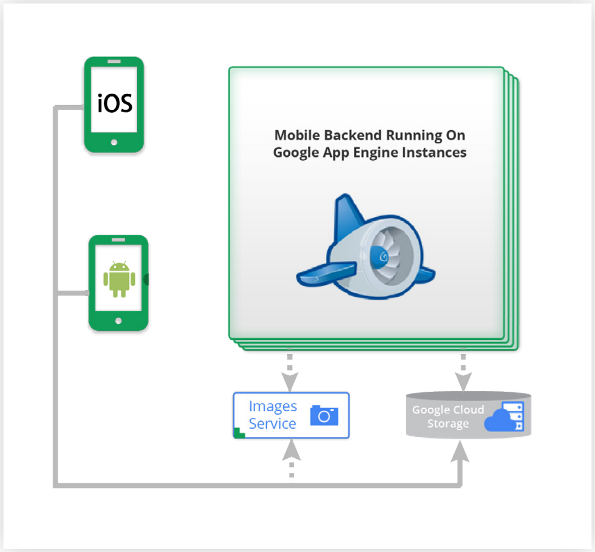
Serving and processing images

If your mobile solution includes a manageable number of static images, you can package them (often in multiple resolutions) as part of your native Android and iOS applications. This applies, for example, to various icons that are part of your application and are used to build the UI of your client applications.

Other category of images are images that are not part of your applications, but are, rather, related to the data that your application processes, such as thumbnails of movies in your movie related application, pictures of products that can be gifted through your application, or avatars in your game or social applications. If you can store such images in resolutions and formats optimized for your mobile clients then you can upload them to Google Cloud Storage and then provide direct URLs to these images to the mobile clients. This approach is very scalable as it allows mobile clients to retrieve these images directly from the scalable cloud storage without involving your mobile backend code.

In some cases, you may want to manipulate and process images in addition to storing them. For example, you may want to serve the images in dynamically adjusted sizes depending on the resolution of your client devices. In such scenarios you can use Google App Engine Images Service to resize, rotate, flip, and crop images, as well as to enhance photographs using a predefined algorithm. These operations can be performed on images from, among other sources, Google Cloud Storage. The modified images can then be served to the clients directly from the dynamic image serving infrastructure provided by Google without involving your mobile backend code. Your code is responsible just for using the Images Service API to obtain the appropriate image URL and for providing this URL to the mobile clients.

Figure 6.
Serving and processing images



Full text and geo-proximity (location based) search

If you want to give your users full text search capability over some of the structured data managed by your mobile solution, you can leverage Google App Engine Search API [6]. For example, if your application provides access to a large and dynamic inventory of items, your users may benefit from being able to find items by entering words that are part of an item's name or description, even if you provide a convenient touch UI with categorization, paging, etc.

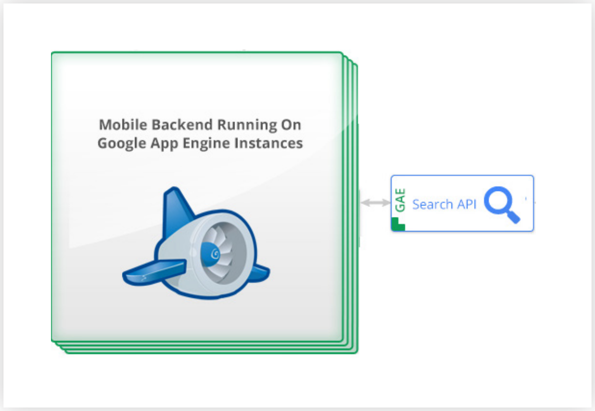
[6]
As of early January 2013, App Engine Search API is an experimental feature with limited free quotas.

To use Search API, you can construct "documents" that contain searchable data and then add these "documents" to an index. Then, your mobile backend can construct search queries, including full-text search queries, and run them against the "documents" in the index. In the example above, each item in the inventory could be mapped to a "document" in the index with properties such as name, description, etc.

Your mobile solution may also perform geo-proximity search if, for instance, you wanted to retrieve the list of nearby stores. You can avoid the complexity of building such functionality yourself and simply leverage Search API for geo-proximity search [7]. The basic flow is very similar to the process described above for full text search. You can simply add geo-location as one of document properties and then include geo-proximity condition in your search queries.

[7]
As of December 2012, geo-proximity search works in applications deployed to App Engine. However, it doesn't work on local dev server.

Figure 7.
Full text and geo-proximity search

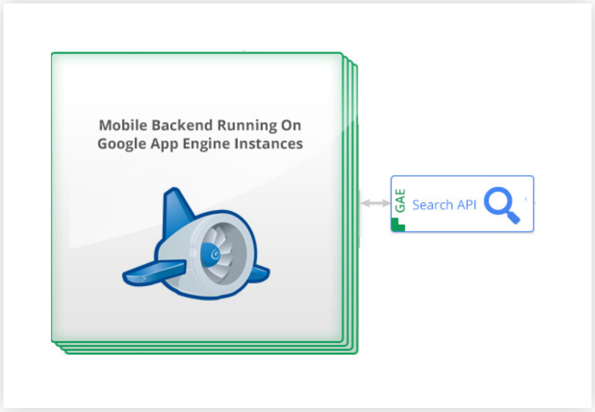


Custom maintenance and other scheduled jobs

Besides processing requests from the mobile clients synchronously through Google Cloud Endpoints and asynchronously using Task Queues, your mobile backend code may need to do some processing at various regular or semi-regular intervals. For example, you may want your mobile solution to send push notification to your mobile users every morning with a relevant local offer or you may want your solution to do various application specific maintenance jobs daily or few times a day. For example, your application may offer premium subscriptions. When the subscriptions expire, or about to expire, you may want to do various related processing, for example, send an email using Mail API, deactivate user accounts, etc. You may also want your mobile backend to update some data cached in Memcache every 10 minutes or update statistics of top players every hour.

With your mobile backend code running on Google App Engine, configuring such scheduled jobs is straightforward with Cron Service. You can configure it to invoke some URLs in your mobile backend at the requested intervals or specific times.

Figure 7.
Full text and geo-proximity search



Storing and Analyzing Application Logs

Application logs can provide a lot of insight about how your users are using your mobile solution and how your mobile backend behaves. With your mobile backend code running on Google App Engine, you can use Logs API to retrieve the application logs. Then, you can upload them to Google Cloud Storage and ingest them into Google BigQuery. If your mobile client application allows users to make a lot of interactions within the applications without sending regular requests to the backend, you may want to augment the backend logs with logs that the client application can collect and periodically send to your mobile backend through your API exposed over Google Cloud Endpoints.

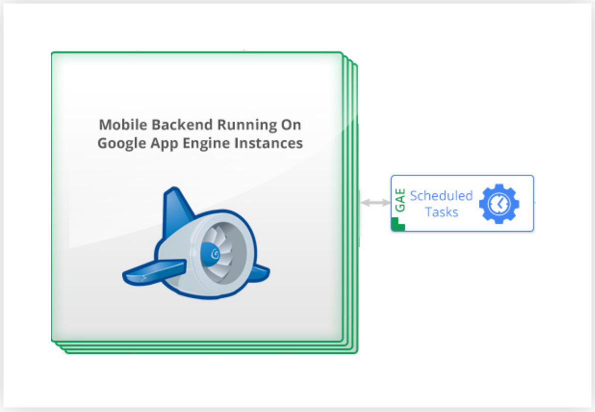


Figure 8.
Scheduled jobs with Cron Service

Running code outside of App Engine sandbox

If you can build your complete mobile backend using runtimes supported by Google App Engine [8], you will be able to avoid having to deal with infrastructural issues such as auto scaling, load-balancing, failover, etc. However, you may have requirements that are not suitable for running in App Engine sandbox. For example, you may want to use some C/C++ libraries, custom binaries, or Perl programs as part of asynchronous request processing or batch processing.

In such scenarios, you can use Google Compute Engine [9] to run the parts of your mobile backend not suitable for hosting on Google App Engine.

When code running on Compute Engine is used for batch processing or asynchronous request processing [10], your code that runs on Google App Engine can enqueue tasks into a pull queue (or multiple queues) and your code that runs on Google Compute Engine can pull these tasks using Task Queue REST API [11].

[8]
As of December 2012, App Engine supports Java 5 and 6, Python 2.7.3 and 2.5.2, as well as Go runtime (the last one as experimental feature).

[9]
As of December 2012, Google Compute Engine is available to selected users in limited preview.

[10]
Google Compute Engine can be used also in other mobile solutions scenarios, but such usage is beyond the scope of this document.

[11]
As of December 2012, Task Queue REST API is an experimental feature.

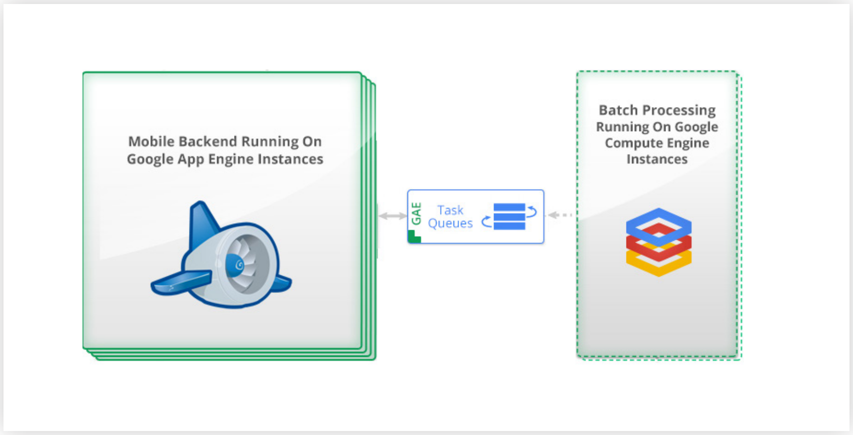

Mobile Backend Running On Google App Engine Instances
GAE
Task Queues
Batch Processing Running On Google Compute Engine Instances

Figure 9.
Running code outside of App Engine sandbox

Other relevant services and APIs

Google offers many other services and APIs that can be used in mobile solutions. These include: AdMob, Analytics, and various Maps APIs, such as Places API, Android Maps, and iOS Maps SDKs. The usage of these services and APIs is beyond the scope of this document.

Architecture diagram

The following architecture diagram (Fig. 10) illustrates the components in the solution:

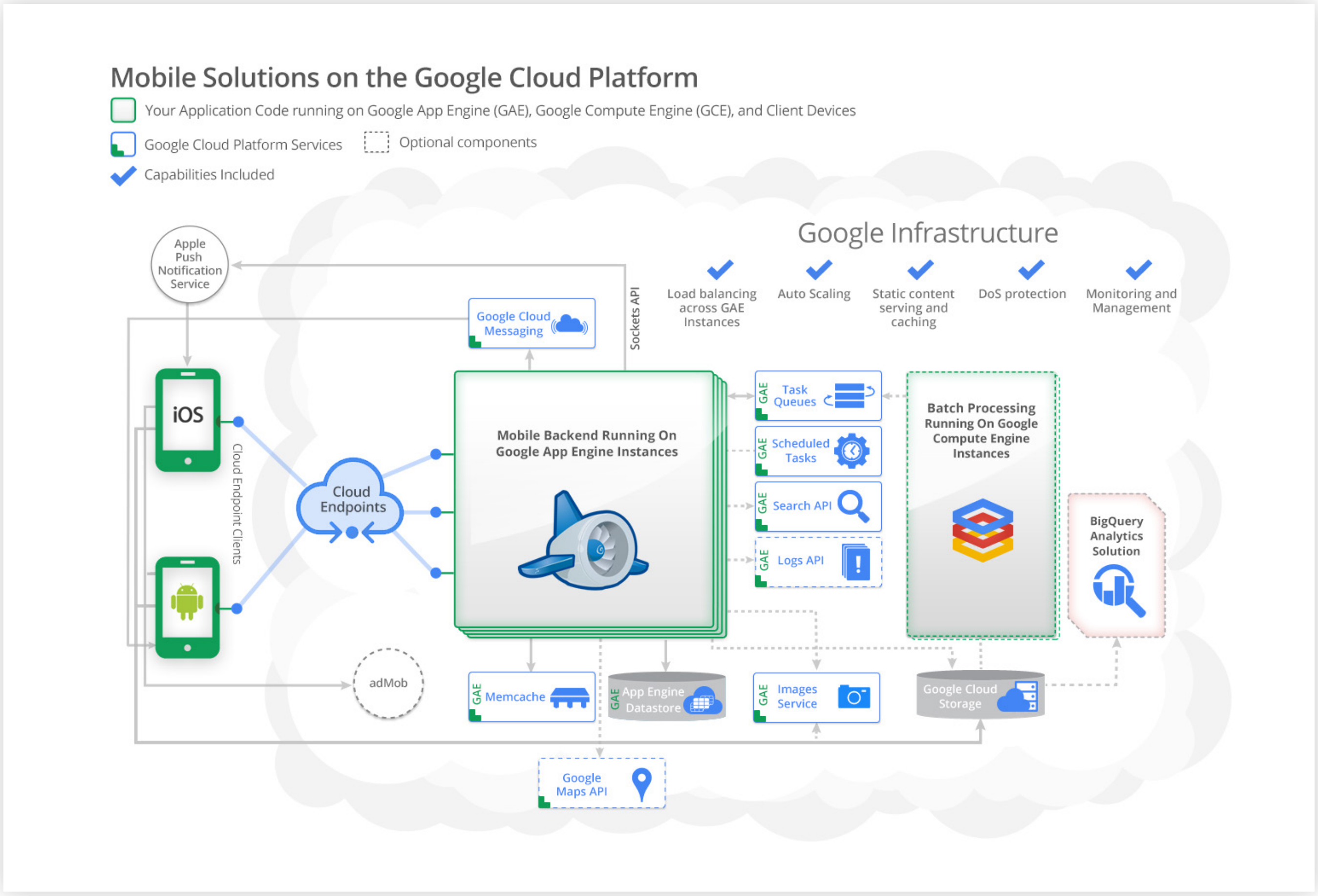

Mobile Solutions on the Google Cloud Platform
Your Application Code running on Google App Engine (GAE), Google Compute Engine (GCE), and Client Devices
Google Cloud Platform Services
Optional components
Capabilities Included
Google Infrastructure
Load balancing across GAE Instances
Auto Scaling
Static content serving and caching
DoS protection
Monitoring and Management
Apple Push Notification Service
Google Cloud Messaging
Sockets API
iOS
Cloud Endpoint Clients
Cloud Endpoints
Mobile Backend Running On Google App Engine Instances
GAE Task Queues
GAE Scheduled Tasks
GAE Search API
GAE Logs API
Batch Processing Running On Google Compute Engine Instances
BigQuery Analytics Solution
adMob
GAE Memcache
GAE App Engine Datastore
GAE Images Service
Google Cloud Storage
Google Maps API

Figure 10. Architecture Diagram of the proposed Mobile Solution. View full size

Conclusion

By leveraging Google Cloud Platform, you can easily build a backend for your mobile solution. You don't have to worry about plumbing and infrastructure. Your solution can be highly scalable and meet requirements for building compelling applications. Among other things, it can use integrated user authentication, support Android and iOS native applications, store, retrieve and process data outside of mobile devices, as well as orchestrate push notification and use geo-proximity search.

Sample Application

You can download the Mobile Shopping Assistant sample application (App Engine backend and Android client) and the corresponding walkthrough document to see how the key parts of the architecture (presented in this document) have been implemented for a particular scenario. The sample application features REST API exposed through Google Cloud Endpoints with integrated OAuth-based client authentication, geo-proximity search, push notification, asynchronous request processing, scheduled tasks, and serving images from Google Cloud Storage.