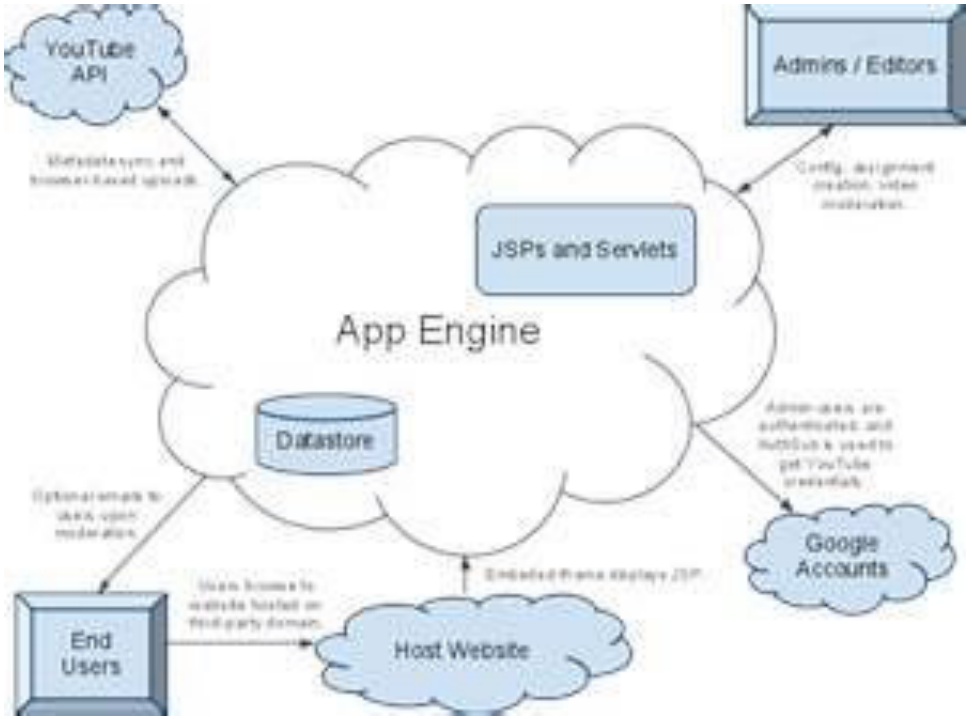
Introduction to Google App Engine

Google App Engine

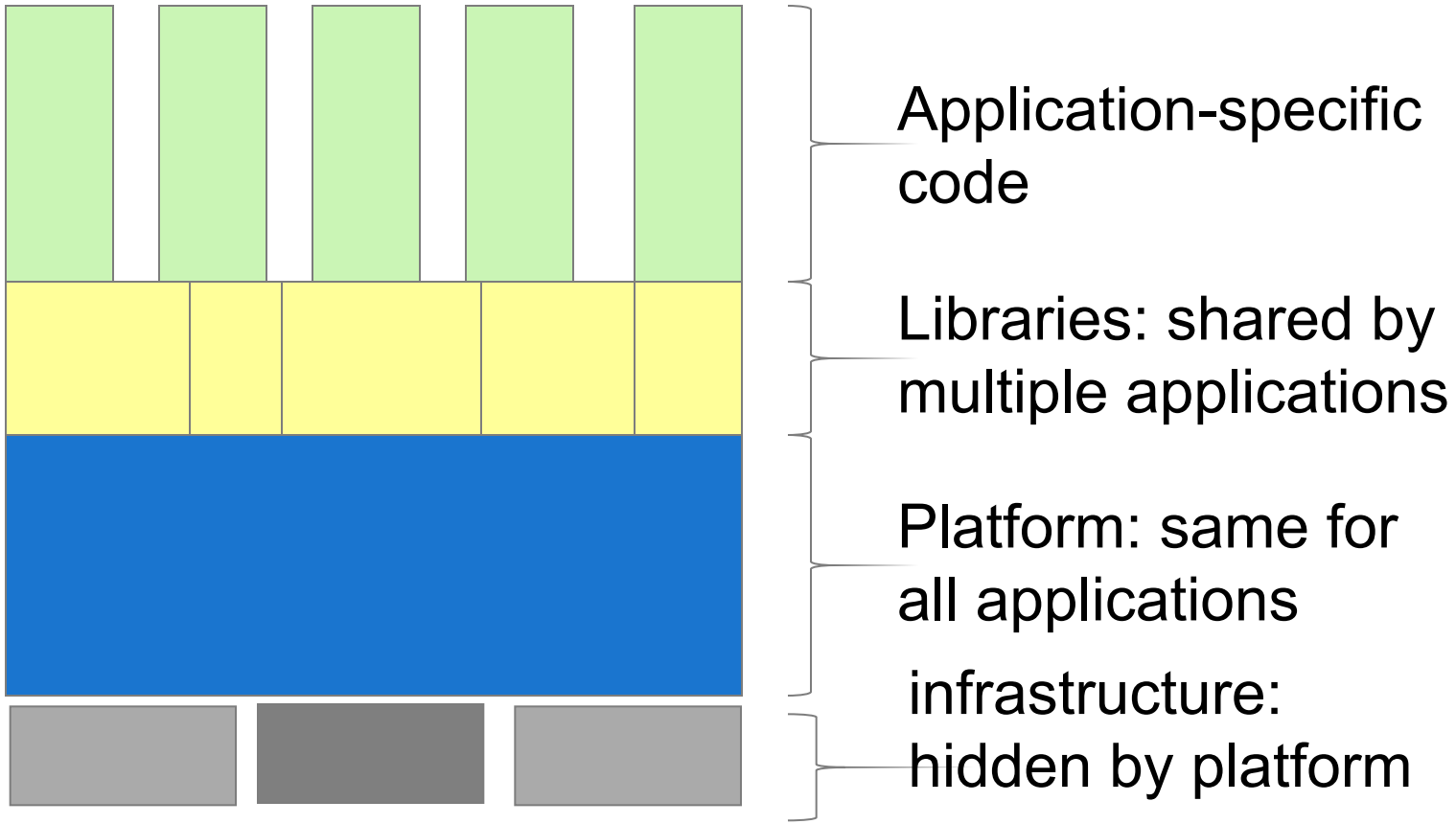
- Does one thing well: running web apps
- Simple app configuration
- Scalable
- Secure



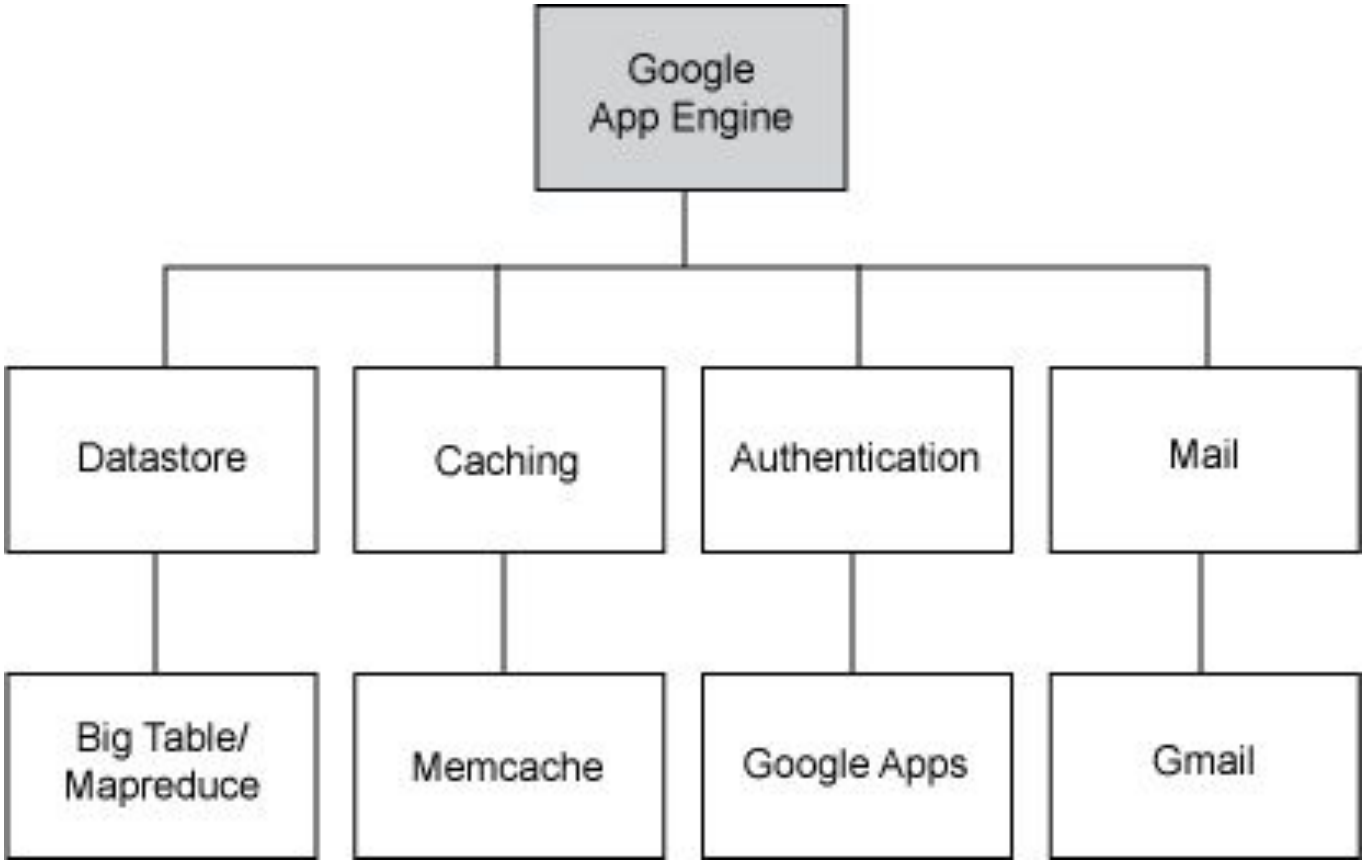
GAE is part of Google Cloud and is Platform As A Service cloud (PAAS)

- Using Googles Infrastructure to host and build your Web Applications
- Free Account --- for limited bandwidth
  ....see http://code.google.com/appengine/ for details

infrastructure vs. platform - What is “The Platform”?



What does GAE Provide



GAE provides ---why not on your own?

- how many developers or architects have experience and the mindset to build applications that support 100s of thousands of concurrent users up all the time?
- Scaling Big is Really Hard
- "Commoditization of Software Architecture and Scaling Skills"
- Horizontal scaling model
  - this is not a model that most web application developers have experience with.
  - instead of using more capable hardware (vertical scaling), you use more instances of less-capable hardware, each handling a slice of the work, often doing the same function (e.g. sliced between groups of users).
  - intent is to reduce centralization of resources
  - ultimate goal is to simply be able to add more instances of the hardware *without limit* to meet increased scale requirements.

App Engine Does One Thing Well

- App Engine handles HTTP(S) requests, nothing else
  - Think RPC: request in, processing, response out
  - Works well for the web and AJAX; also for other services

- App configuration is dead simple
  - No performance tuning needed

- Everything is built to scale
  - “infinite” number of apps, requests/sec, storage capacity
  - APIs are simple

GAE has limitations with free account

- What is Free and What is NOT
- **FREE:** All applications have a default quota configuration, the "free quotas", which should allow for roughly 5 million pageviews a month for an efficient application. You can read more about system quotas in the quota documentation.
- **PAY FOR MORE:** As your application grows, it may need a higher resource allocation than the default quota configuration provides. You can purchase additional computing resources by enabling billing As your application grows, it may need a higher resource allocation than the default quota configuration provides. You can purchase additional computing resources by enabling billing for your application. Billing enables developers to raise the limits on all system resources and pay for even higher limits on CPU, bandwidth, storage, and email usage

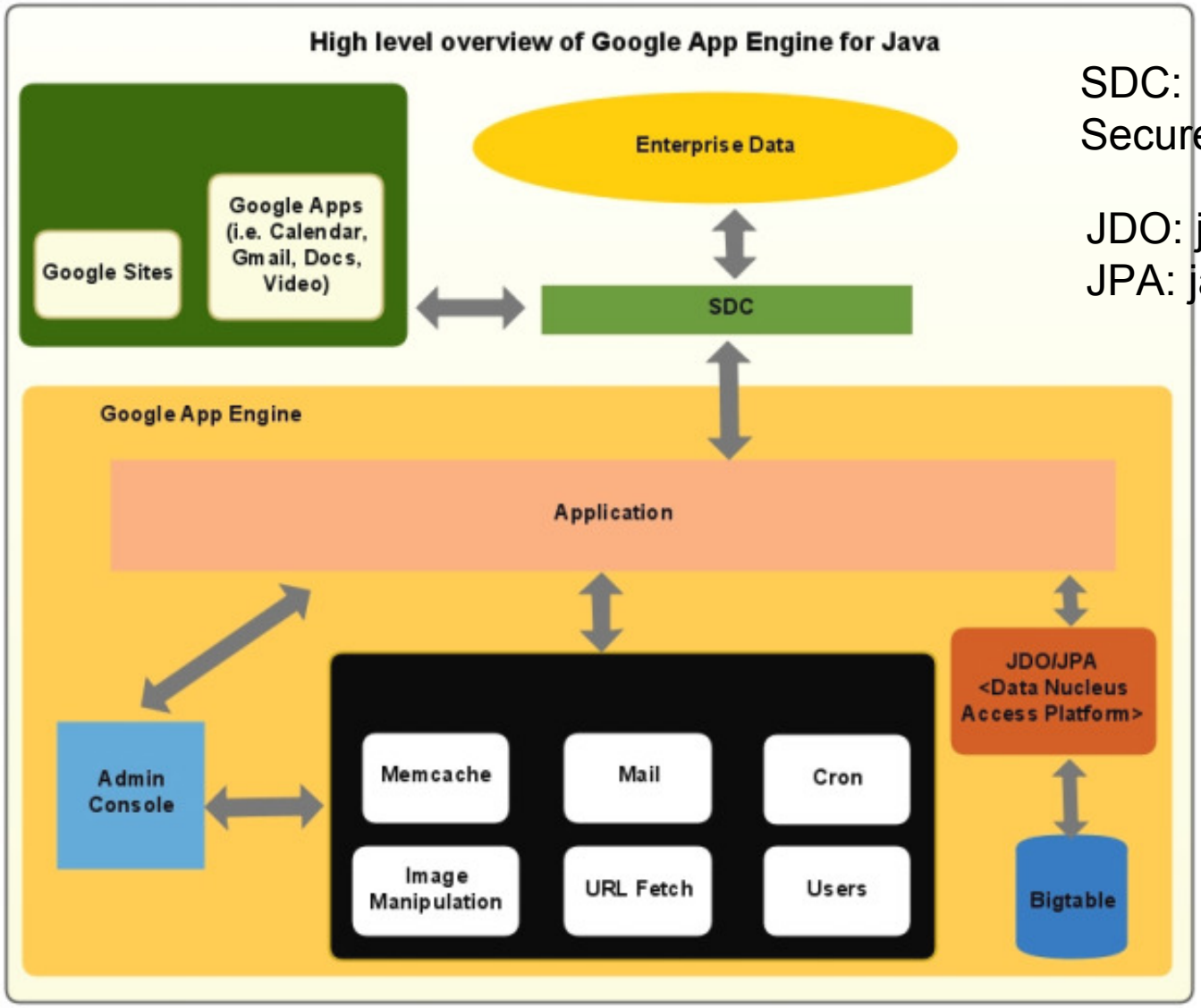
Services

- URLFetch – fetch web resources/services
- Images – manipulate images: resize, rotate, flip, crop
- Google Accounts
- Mail
- XMPP – instant messages
- Task Queue – message queue; allow integration with non-GAPPs
- Datastore – managing data objects
- Blobstore – large files, much larger than objects in datastore, use <key, object> to access

What kind of “Apps” can Google App Engine support

- What languages are supported: **python, java, php and Go**

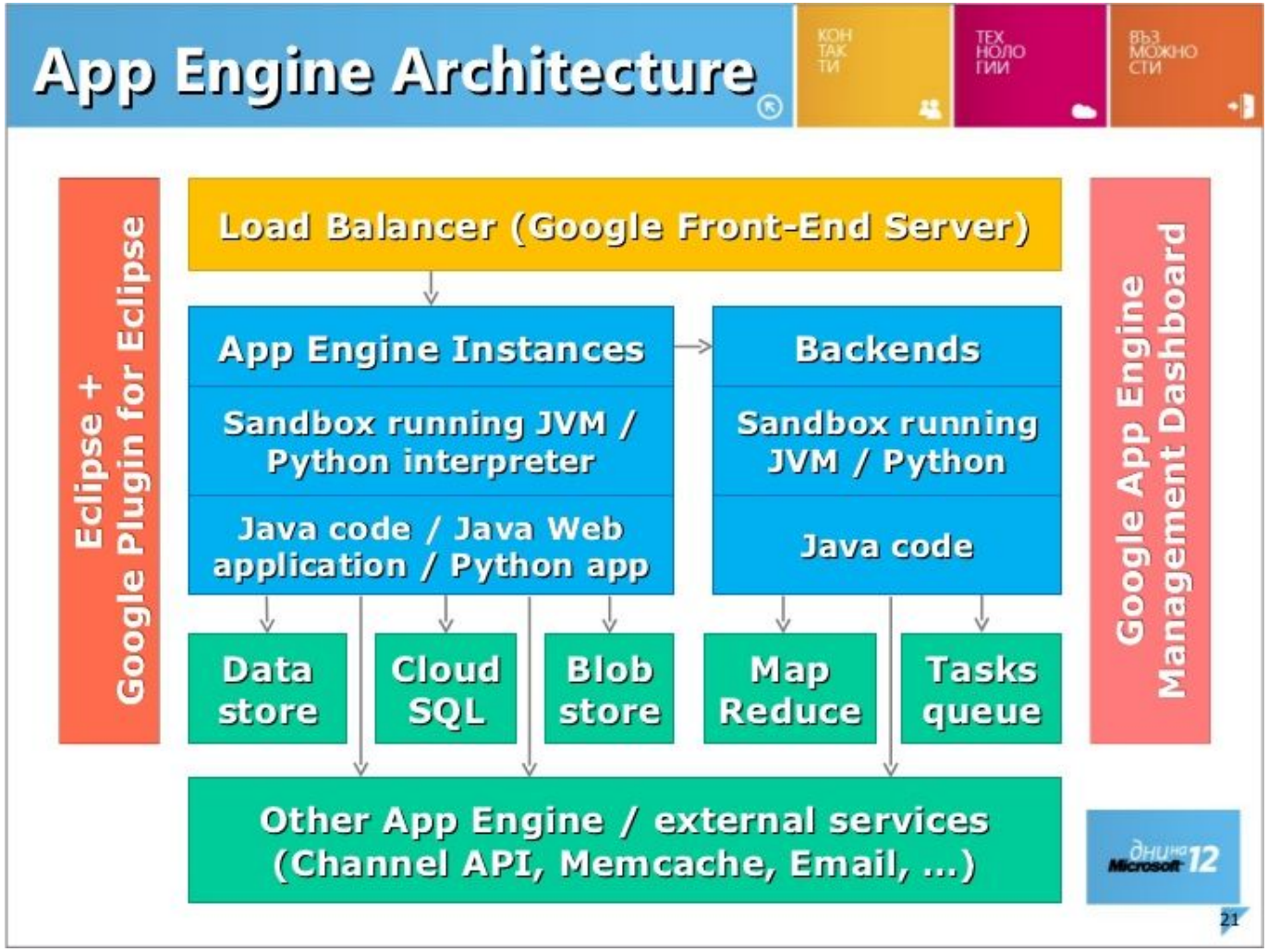
App Engine Architecture (java)



SDC:
Secure data connector

JDO: java data object
JPA: java persistent API

GAE architecture featuring java



GAE: Java or python? Or how about php or Go

- First Question: What do you or your developers know?
- Benefit of Python: powerful python syntax, library, possibly shorter code
- Benefit of Java: rich language that is mature, many packages, can use JDO/JPA
  - Better portability if you need to use Bigtable to store data

Java on GAE

- Java Servlets, JSPs on Google AppEngine
- **You provide your app's servlet classes, JavaServer Pages (JSPs), static files and data files, along with the deployment descriptor (the web.xml file) and other configuration files, in a standard WAR directory structure.**
- **App Engine serves requests by invoking servlets according to the deployment descriptor.**

SCALING AND RESULTING APP LIMITATIONS

Scaling in GAE how is it achieved ---leads us to some limitations in coding

- Low-usage apps: many apps per physical host
- High-usage apps: multiple physical hosts per app

- Stateless APIs are trivial to replicate

- Datastore built on top of Bigtable; designed to scale well
  - Abstraction *on top of* Bigtable
  - API influenced by scalability
    - No joins
    - Recommendations: *denormalize* schema; precompute joins

Restrictions on Java in GAE – a new way of thinking to get scalability

- To allow App Engine to distribute requests for applications across multiple web servers, and to prevent one application from interfering with another, the application runs in a restricted "sandbox" environment.
- The JVM runs in a secured "sandbox" environment to isolate your application for service and security.
- The sandbox ensures that apps can only perform actions that do not interfere with the performance and scalability of other apps.

Restrictions on Java in GAE – a new way of thinking to get scalability

- **An app cannot spawn threads, write data to the local file system or make arbitrary network connections. (cant allow to store to local file system when things are distributed ---could be problems...you need to use the datastore instead)**
  - **HOWEVER--Apps use the URL FetchHOWEVER--Apps use the URL Fetch service to access resources over the web, and to communicate with other hosts using the HTTP and HTTPS protocols. Java apps can simply usejava.net.URLConnection and related classes from the Java standard library to access this service.**

- **app also cannot use JNI or other native code.**

Again What you CAN NOT DO!!!

- **write to the filesystem**. SOLUTION --- Applications must use the App Engine datastore for storing persistent data. Reading from the filesystem is allowed, and all application files uploaded with the application are available.
- **open a socket or access another host directly.** SOLTUION --- An application can use the App Engine URL fetch service to make HTTP and HTTPS requests to other hosts on ports 80 and 443, respectively.
- **spawn a sub-process or thread.** CAVEAT A web request to an application must be handled in a single process within a few seconds. Processes that take a very long time to respond are terminated to avoid overloading the web server.
- **make other kinds of system calls.**

MORE ON SCALING

Automatic Scaling to Application Needs

- You don’t need to configure your resource needs
- One CPU can handle many requests per second
- Apps are hashed onto CPUs:
  - One process per app, many apps per CPU
  - Creating a new process is a matter of cloning a generic “model” process and then loading the application code (in fact the clones are pre-created and sit in a queue)
  - The process hangs around to handle more requests (reuse)
  - Eventually old processes are killed (recycle)
- Busy apps (many QPS query per sec) get assigned to multiple CPUs
  - This automatically adapts to the need
    - as long as CPUs are available

Preserving Fairness Through Quotas

- Everything an app does is limited by quotas, for example:
  - request count, bandwidth used, CPU usage, datastore call count, disk space used, emails sent, even errors!
- If you run out of quota that particular operation is blocked (raising an exception) for a while (~10 min) until replenished
- Free quotas are tuned so that a well-written app (light CPU/datastore use) can survive a moderate “slashdotting”
- **The point of quotas is to be able to support a very large number of small apps (analogy: baggage limit in air travel)**
- Large apps need raised quotas
  - currently this is a manual process (search FAQ for “quota”)
  - in the future you can buy more resources

WHAT DO YOU MEAN CAN'T SPAWN NEW THREADS???

No creating new threads!!!

- **A Java application cannot create a new java.lang.ThreadGroup nor a new java.lang.Thread. These restrictions also apply to JRE classes that make use of threads. For example, an application cannot create a new java.util.concurrent.ThreadPoolExecutor, or a java.util.Timer. An application *can*perform operations against the current thread, such as Thread.currentThread().dumpStack().**
-
- **SOLUTIONS -- here it is to rethink the use of threads**
  - **make the separate threads web services (other apps) you call and invoke and get back results**
  - **Task queues+ Task options**

The solutions to no new threads

- **SOLUTIONS -- here it is to rethink the use of threads**
  - **Solution 1: make the separate threads web services (other apps) you call and invoke and get back results**
  - **Solution 2: Task queues+ Task options**
    - **If an app needs to execute some background work, it can use the Task Queue API to organize that work into small, discrete units, called tasksIf an app needs to execute some background work, it can use the Task Queue API to organize that work into small, discrete units, called tasks. The app adds tasks to task queuesto be executed later.**
- **(PYTHON: http://code.google.com/appengine/docs/python/taskqueue/)**
- **(JAVA: http://code.google.com/appengine/docs/java/taskqueue/overview.html)**

ANOTHER LIMITATION ---YOUR APP MUST RESPOND IN A CERTAIN TIME

If your app fails to respond in a certain time you will have problems –your app will be timed out by GAE

- All requests (including tasks) in app engine have a time limit of XXX seconds (30 seconds --but, see current limits on google documentation). If your calculations will take longer than that, you will need to figure out a way to break them down into smaller chunks. App engine's sweet spot is web apps, not number crunching.
- WHY are they doing this? They want to serve the most web apps they can on their platform
- Solution: again task queues for longer processing needs\

WHAT ABOUT DATA

GAE Datastore (storage organization)

- Data model
  - Property, entity, entity group
  - Schemeless: properties can have different types/meanings for different objects
  - Allow (1) object query (2) SQL-like query
- Transaction
- Can be applied to a group of operations
- Persistent store (check BigTable paper)
  - Strongly consistent
  - Not relational database
  - Index built-in
- Memcache
  - Caches objects from bigtable, to improve performance

Hierarchical Datastore

- *Entities* have a *Kind,* a *Key,* and *Properties*
  - Entity ~~ Record ~~ Python dict ~~ Python class instance
  - Key ~~ structured foreign key; includes Kind
  - Kind ~~ Table ~~ Python class
  - Property ~~ Column or Field; has a type
- Dynamically typed: Property types are recorded per Entity
- Key has either *id* or *name*
  - the id is auto-assigned; alternatively, the name is set by app
  - A key can be a *path* including the parent key, and so on
- Paths define *entity groups* which limit *transactions*
  - A transaction locks the *root entity* (parentless ancestor key)
  - Recall the chubby lock service in bigtable paper

Indexes

- Properties are automatically indexed by type+value
  - There is an index for each Kind / property name combo
  - Whenever an entity is written all relevant indexes are updated
  - However Blob and Text properties are never indexed
- This supports basic queries: AND on property equality
- For more advanced query needs, create *composite indexes*
  - SDK auto-updates index.yaml based on queries executed
  - These support inequalities (<, <=, >, >=) and result ordering
  - Index building has to scan *all* entities due to parent keys

Free tier

- First 5GB
- Daily limits see Online for CURRENT QUOTAS

Resource	Free Default Limit		Billing Enabled Default Limit	
	Daily Limit	Maximum Rate	Daily Limit	Maximum Rate
Channel API Calls	657,000 calls	3,000 calls/minute	91,995,495 calls	32,000 calls/minute
Channels Created	100 channels	6 creations/minute	Based on your budget	60 creations/minute
Channel Hours Requested	200 hours	12 hours requested/minute	Based on your budget	180 hours requested/minute
Channel Data Sent	Up to the Outgoing Bandwidth quota	22 MB/minute	1 TB	740 MB/minute

What happens when you exceed your budget (free or what you set)

- When an application consumes all of an allocated resource, the resource becomes unavailable until the quota is replenished.
- This may mean that your application will not work until the quota is replenished.

What happens when you exceed your budget (free or what you set)

- For resources that are required to initiate a request, when the resource is depleted, App Engine by default returns an HTTP 403 Forbidden status code for the request instead of calling a request handler. The following resources have this behavior:
  - Bandwidth, incoming and outgoing

- For all other resources, when the resource is depleted, an attempt in the application to consume the resource results in an exception. This exception can be caught by the application and handled, such as by displaying a friendly error message to the user. In the Python API, this exception is apiproxy_errors.OverQuotaError. In the Java API, this exception is com.google.apphosting.api.ApiProxy.OverQuotaException.

Pricing

- The part exceeding the free quota
- User defined budget
- Look On line for CURRENT PRICES

Resource	Unit	Unit cost (in US $)
Instances*	Instance hours	$0.05
Outgoing Network Traffic	Gigabytes	$0.12
Incoming Network Traffic	Gigabytes	Free
Datastore Storage	Gigabytes per month	$0.18
Blobstore, Logs, and Task Queue Stored Data	Gigabytes per month	$0.026
Dedicated Memcache	Gigabytes per hour	$0.06
Logs API	Gigabytes	$0.12
SSL Virtual IPs** (VIPs)	Virtual IP per month	$39.00
Sending Email, Shared Memcache, Pagespeed, Cron, APIs (URLFetch, Task Queues, Image, Sockets, Files, and Users)		No Additional Charge

Security

- Constrain direct OS functionality
  - no processes, threads, dynamic library loading
  - no sockets (use urlfetch API instead)
  - can’t write files (use datastore)
  - disallow unsafe Python extensions (e.g. ctypes)

- Limit resource usage
  - Hard time limit of 30 seconds per request
  - Most requests must use less than 300 msec CPU time
  - Hard limit of 1MB on request/response size, API call size, etc.
  - Quota system for number of requests, API calls, emails sent, etc
  - Free use for 500MB data and 5M requests per month
  - 10 applications per account