Distributed Systems: Google App Engine 2

Stefan Walraven, Wouter Joosen

iMinds-DistriNet, KU Leuven

Overview

- Recap: PaaS
- Google App Engine Services
  - Storage
  - Task Queues
  - Other

Recap: PaaS

- **Platform as a Service (PaaS)**
  "A cloud service model that provides the consumer the capability to **deploy** onto the cloud infrastructure **consumer-created or acquired *applications* created using programming languages, libraries, services, and tools supported by the provider**.[1] The **consumer does not manage or control the underlying cloud infrastructure** including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment."
  [The NIST Definition of Cloud Computing]

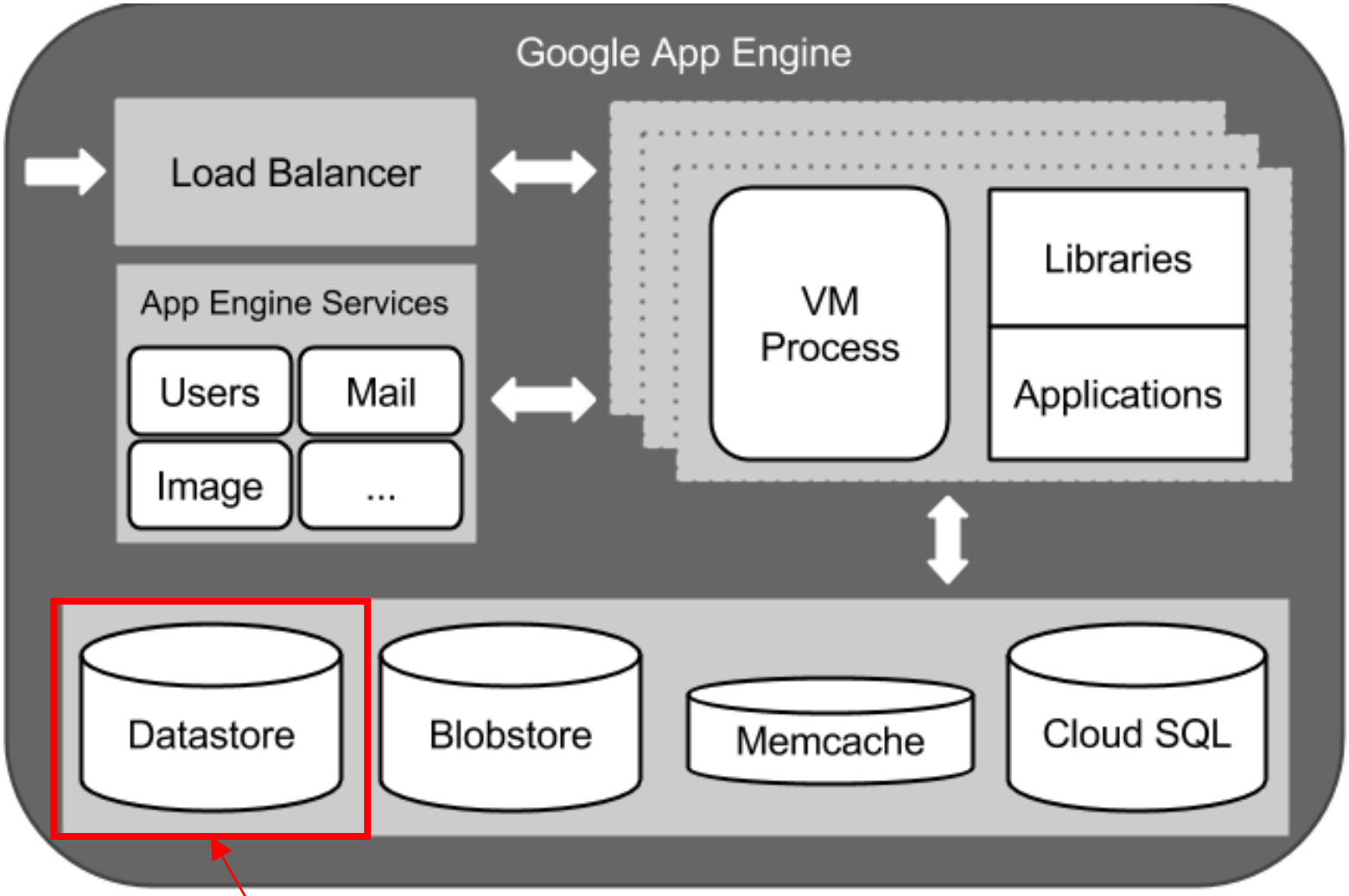
[1] This capability does **not necessarily** preclude the use of **compatible** programming languages, libraries, services, and tools from other sources.

Recap: PaaS (2)

- **Short version:**
  PaaS provides a **computing platform and solution stack** upon which applications and services can be **developed and hosted** using programming concepts and tools supported by the provider.

- Examples:
  - Amazon Elastic Beanstalk
  - Cloud Foundry
  - Force.com
  - Google App Engine
  - Oracle PaaS
  - Red Hat OpenShift
  - Windows Azure

Recap: GAE



Focus of last session

Google App Engine Services

- Storage
  - Datastore (subject of GAE session 1)
  - Blobstore
  - Memcache
  - Cloud SQL
- Task Queues
  - Subject of GAE session 2
- Other:
  - Users
  - Scheduled tasks
  - Fetch URL
  - Namespaces
  - Mail
  - Modules

App Engine Storage

Datastore

- See GAE 1 slides

Blobstore

- Blobstore API
  - Cf. Amazon S3 and Windows Azure Blob Storage
  - **Blob** = binary large objects
    - Stored in GAE datastore
  - Goal: serve data objects >> datastore entities
  - created by uploading a file (not via application code)
  - E.g. video or image files

Memcache

- Memcache API
  - **distributed in-memory key-value storage** (not persistent!)
  - 2 purposes
    - To speed up common datastore queries (i.e. caching)
    - To store temporary data
  - Low-level API:

```
MemcacheService cache = MemcacheServiceFactory.getMemcacheService();
value = (<cast to appropriate type>) cache.get(key); // read from cache
if (value == null) {
    // get value from other source (e.g. datastore)
    // ...
    cache.put(key, value); // populate cache
}
```

  - Supports JCache (JSR 107) (in development)
    - Proposed interface standard for memory caches

Cloud SQL

- Google Cloud SQL
  - Instance == Relational database server
    - Can contain multiple databases
  - Compatible with MySQL
    - Supporting JDBC, JPA, JDO, Hibernate...
    - Configure it correctly! (e.g. connections)
  - Replicated to ensure availability and backup
  - Not free

Other storage services

- Google Cloud Storage (GCS)
  - Storing and service large files
  - Strongly consistent
  - Can also be accessed via Blobstore API
  - Difference with Blobstore unclear
    - GCS seems to be more programmable

App Engine task queues

Message Queues

- Asynchronous inter-process/-thread communication
- Use queue to pass control or content
- ⇔ request-reply
- Examples:
  - Java Message Service (JMS)
  - RabbitMQ
  - JBoss HornetQ

Message Queues (2)

- Often used in cloud computing:
  - to improve performance and scalability
  - by decoupling “bottleneck” activities to worker processes
    - Asynchronous execution
      - No delay at front end (user experience)
    - Opportunity to scale and parallelize execution
      - Increase number of workers
  - Examples:
    - Amazon Simple Queue Service (SQS)
    - Windows Azure Queues
    - **GAE Task Queues**

GAE Task Queues: Queue

- Perform background processing
  - Outside context of user request (asynchronously)
  - Steps:
    - Split up work into *tasks* (== small, discrete unit of work)
    - Insert tasks into *queue* to be executed later
- 2 queue configurations:
  - *Push queue* (default)
    - Automatic scaling of processing capacity to match queue configuration
    - Automatically deletes tasks after processing
  - *Pull queue*
    - More control over when to process tasks
    - Manual scaling and deleting (handled by application)
    - Supports integration with non-App-Engine code

GAE Task Queues: Queue (2)

- Default (push) queue
  - Available for all applications
  - No configuration required
  - Throughput rate: 5 task invocations per second

```
Queue queue = QueueFactory.getDefaultQueue();
```

- Custom queues (push & pull)
  - Define configuration in **queue.xml** (or **queue.yaml**)
    - name, processing rate, max concurrency, number of retries...
    - in WEB-INF directory inside the WAR
  - To group similar types of tasks in same queue

```
Queue queue = QueueFactory.getQueue("<queue name>");
```

GAE Task Queues: Task

- Task
  - Unit of work to be performed by the application
  - Object of TaskOptions class
  - Contains an endpoint
    - With request handler for task (i.e. worker)
    - with data payload to parameterize the task (optional)
  - Has a name (optional)
    - Generally unique
    - To delete specific tasks
- 2 kinds of workers:
  - **Default using URL (e.g. to Java Servlet)**
  - Backends (no restrictions, but extra cost per hour per instance)

GAE Task Queues: Task (2)

- Creating and processing tasks:
  - Steps:
    1. Create task and insert in queue
    2. Workers (i.e. task consumer) lease tasks:
       - When task is leased, then unavailable for other workers until lease expires
    3. Workers process leased tasks
       - If success => delete task from queue
       - If lease expires => stop task
       - If failure before lease expires => retry (max. # retries in config)
  - Different depending on queue type:
    - Push queues:
      - automatically by App Engine (lease = 10 minutes)
    - Pull queues:
      - programmatically=> pull queue when worker is available

GAE Task Queues: Push

- Push queues
  - Create task:

```
// call request handler at URL /path_to_worker with parameter key
queue.add(withUrl("/path_to_worker").param("key", <value (byte[]/String)>));
                                    OR          .payload(<value (byte[]/String)>));
// OR
// call default request handler at URL /_ah/queue/queue_name
queue.add();
// OR
// asynchronous call to queue
queue.addAsync()  /  addAsync(<TaskOptions>);
```

  - Process task

GAE Task Queues: Push (2)

- Push queues
  - Create task
  - Process task:
    - Create endpoint for URL (e.g. Java Servlet)

```
public class Worker extends HttpServlet {
        // default handler method is POST
        public void doPost(<request>, <response>) {
                String param = <request>.getParameter("key");
                OR      InputStream payload = <request>.getInputStream();
                // unmarshall payload
                // execute task
        }
}
```

    - Specify endpoint in **web.xml**
      - Map class name of Worker to a url pattern path_to_worker (cf. other servlets)

GAE Task Queues: Push (3)

- For each distinct task:
  - Create handler
  - Serialize and deserialize complex arguments
    - Only string or byte[] as payload
  - Cumbersome, especially for many diverse and small tasks
- Solution: DeferredTask
  - Interface to define a task as a single method
    - run() method will be called when received (implements Runnable)
    - Uses Java serialization (implements Serializable)
  - Return == success
  - Exception == failure

GAE Task Queues: Pull

- Pull Queues
  - Create task:

```
queue.add(withMethod(PULL).payload(<payload(byte[]/String)>)
                                   .taskName("<name>"));
```

  - Lease task (worker inside GAE):

```
List<TaskHandle> tasks =
    queue.leaseTasks(<# lease units>, <TimeUnit of  lease>,
                                  <max # tasks to lease>);
```

  - Lease task (worker outside GAE): REST API
  - Beta: lease tasks based on tag (i.e. filter tasks)
  - Delete task:

```
queue.deleteTask("<name>");
```

GAE Task Queues: Varia

- Securing URLs for Tasks:
  - Prevent users from accessing URLs of workers
    - Restrict access to administrator accounts via **web.xml**
- Push Queues and Development Server
  - Development server
    - Does not respect <rate> and <bucket-size>
    - Does not retry tasks
    - Does not preserve queue state across server restarts
  - Examine and manipulate tasks from developer console:
    - http://localhost:8888/_ah/admin/taskqueue
    - Works only after one task has been inserted

Other App Engine Services

Users

- Authentication (Users service API) of users with:
  - Google Account
  - Account on Google Apps Domain
  - OpenID identifier (deprecated)

```
UserService userService = UserServiceFactory.getUserService();
User user = userService.getCurrentUser();
```

Scheduled Tasks

- Scheduled tasks
  - Cron jobs
    - For example, send report email or update cached data
    - 20 scheduled tasks for free
  - At defined times or regular intervals
  - Automatically triggered by GAE Cron Service
  - **cron.xml** OR **cron.yaml** in WEB-INF folder within the WAR package
    - Must define *url* and *schedule*

Fetch URL

- Enables GAE applications to
  - Communicate with other applications
  - Access other resources on the web
- Issue HTTP and HTTPS requests and receive responses
  - Using java.net.URLConnection Java standard library
  - Using low-level API
- Actually *the only built-in support for integration* with other applications
  - Other integration support should be implemented by the developer or added via third-party libraries
    - Web services and RESTful web services

Namespaces

- Namespaces API
  - *Easy partitioning of data*
    - Each set of data == namespace
      - Should be set manually (e.g. at the start of a request)
    - All namespace-enabled APIs (datastore, memcache and task queues) use the current namespace by default
  - Useful to achieve *multi-tenancy* at the application level
    - **Application-level multi-tenancy**: Architectural style that enables a *single application instance* on top of shared hardware and software to serve end users from different *tenants* (i.e. client organizations) simultaneously
      - Key enabler for economies of scale
    - Namespace = data partition for one tenant

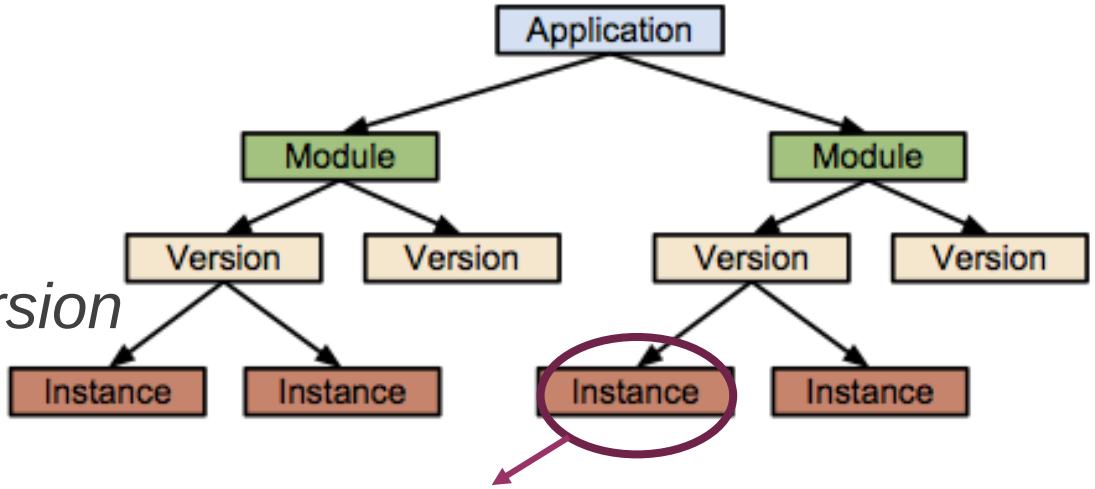
Mail

- Send email messages on behalf of
  - the application's administrators
  - users with Google Accounts (and are signed in)
    - Retrieve via Users API
- Receive emails at various addresses

- Via JavaMail API (javax.mail)
  - Uses low-level Mail service API
  - No SMTP configuration required/possible

Modules

- Factor large applications into logical components
  - that share stateful services (e.g. Memcache, Datastore, Task Queues)
  - and communicate in secure fashion
  - Cf. *microservices* design pattern
- Each module:
  - Source code
  - Config files
  - Represents a *version*



Modules (2)

- Application using modules is organized as EAR directory structure (cf. Java EE):
  - META-INF directory
    - appengine-application.xml (general info)
    - application.xml (list of modules)
  - Separate WAR subdirectory per module
    - appengine-web.xml (configuration of module)
    - web.xml
- Communication between modules:

```
ModulesService modulesAPI = ModulesServicesFactory.getModulesService();
String currentModuleName = modulesApi.getCurrentModule();
int currentInstance = modulesApi.getCurrentInstance();
URL url = new URL("http://" + modulesApi.getVersionHostname("my-backend-
                                module","v1") + "/<web page or application name>");
```

References

- GAE Developer's Guide
  https://cloud.google.com/appengine/docs/
- Java Service APIs
  https://cloud.google.com/appengine/docs/java/apis
- Task Queue Java API Overview
  https://cloud.google.com/appengine/docs/java/taskqueue/
- Using Push Queues
  https://cloud.google.com/appengine/docs/java/taskqueue/overview-push
- Java Task Queue Configuration
  https://cloud.google.com/appengine/docs/java/config/queue
- Java Task Queue Javadoc
  https://cloud.google.com/appengine/docs/java/javadoc/com/google/appengine/api/taskqueue/package-summary