Applicazione Ipermediali
(Web e Multimedia)
2010/2011

Cloud computing
Politecnico di Milano

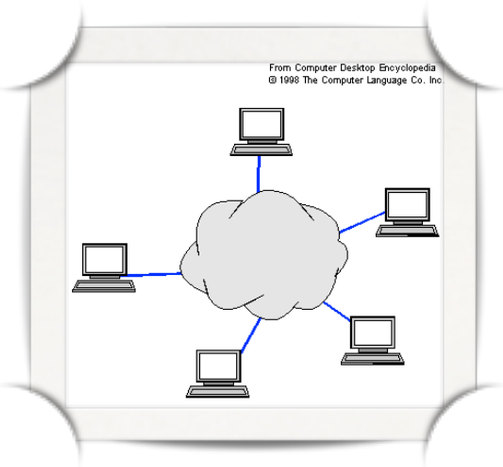
Agenda

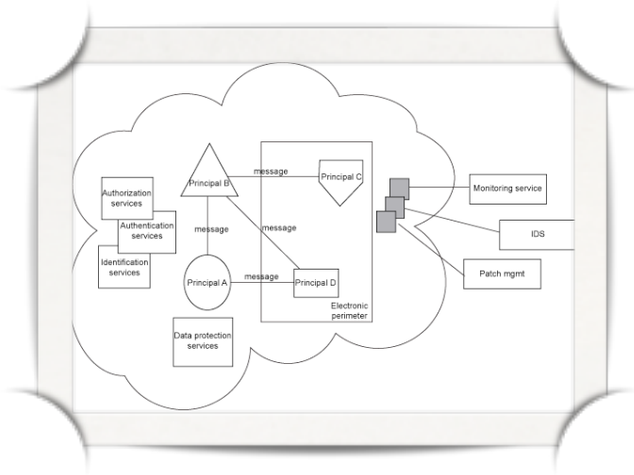
- **Cloud computing**
  - PaaS
  - IaaS
  - SaaS
- **NoSQL DB**
- **Google App Engine**
  - Environment (Features, Sandbox, Languages)
  - Datastore
  - Services
  - Limits
  - Development (workflow and tools)

Cloud computing …

The origins…

- "Cloud": well known concept in IT
  - *Networking*
  - *SOA (Service Oriented Architecture)*





- Focus on required services
  - *availability*
  - *performance*

- Not on "*where*" and "*how*"

... Cloud computing ...

What's cloud and which benefits?

Cloud computing is a Model that implies:

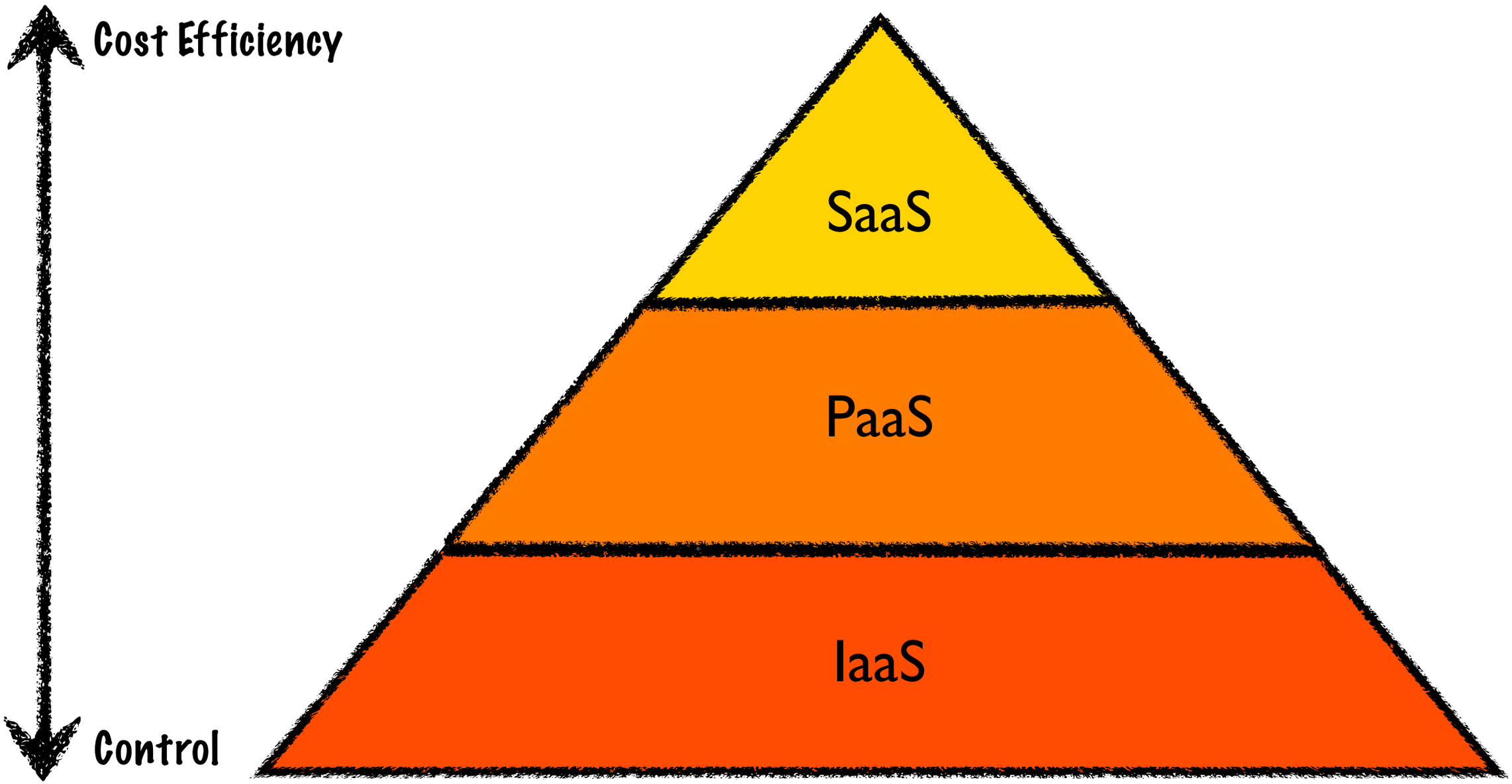
- *usage simplification of IT technologies*
- *modifications of IT service producer - consumer relationships*
- *elastic resources adoption: allocation/reallocation and scalability is now simpler*

Benefits:

- *control on costs (..."pay as you go"...)*
- *in-house IT infrastructure reduction (less maintenance and day by day costs, no more investments, but only operating costs)*
- *high flexibility and scalability (now it's possible to rapidly follow market requests and changes)*
- *availability at low costs*

... Cloud computing ...

Categories



Overlaps in categories is possible depending on who is doing the service classification (Manager, System Administrator, Developer and so on...)

... Cloud computing ...

IaaS

- IaaS *(Infrastructure as a Service)*:

  -IT infrastructures supplied as services (virtual servers, datastore e data-center space, network devices, firewalls, network bandwidth ...)

  -service costs are based on the real resource usage (aka "utility computing")

  -typical users of this kind of services have well-grounded IT expertise but no in-house infrastructures

- Example:

  -Amazon Elastic Compute Cloud (EC2)

... Cloud computing ...

PaaS

- PaaS *(Platform as a Service)*:

  -IT infrastructure and a whole environment based on it is supplied.

  -service user is able to develop his solution using environment API, services and features

  -no control on the infrastructure (HW, SO, network etc.), only some hooks available (for application scope)

  -minimal control on the environment in which owned applications are running

  -focus on solutions development based on supplied services integration

- Esempi:

  -Google App Engine
  -Microsoft Azure

... Cloud computing

SaaS

- Saas *(Software as a Service)*:
  - *top used solution in the market*
  - *cloud providers supply access only to the application*
  - *fewer customizations are possible (generally based on presentation layer)*
  - *costs based on time subscriptions or on real usage*

- Esempi:
  - *Google Apps*
  - *Google Docs*
  - *GMail*
  - *SalesForce on Demand*
  - *Trend Micro SaaS*

NoSQL DB ...

Story and definition

- "NoSQL": first time of this terminology in 1998 (by Carlo Strozzi)
  - *Lightweight DB without SQL interface*
- "Revamped" by Eric Evans in 2009
  - *"distributed open-source databases" debate organized by Johan Oskarsson (last.fm)*

- One possible definition (*by nosql-database.org*):

  "Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontal scalable ..."

  Summarizing, NoSQL =>
  - non relational
  - distributed
  - horizontally scalable

- NoSQL usually intended as *"Not only sql"*

... NoSQL DB ...

Features

Main features:

- Schema less:

  No fixed and predefined design based record structure. Number of fields for each tuple may vary (eg.: document-oriented database)

- Horizontal scalable:

  Computational and Data load can be distributed on several nodes belonging to the same "cluster".

  Horizontal scaling capability should be considered also like the simplicity to add several nodes to the cluster distributing the load.

- Simple API:

  The way data are exposed from the database is very important. NoSQL db often use simple JSON messages with REST API or tcp sockets, etc...

... NoSQL DB ...
ACID (relazionali)

- ACID (***A**tomicity, **C**onsistency, **I**solation, **D**urability*)

*-**A**tomicity: transaction is the minimal unit of work. Failure in a part of a transaction means the whole transaction fails("All or nothing rule")*

*-**C**onsistency: the condition in which DB integrity rules are all verified. Transactions cannot violate DB integrity.*
From a consistent state, the database can only go to another consistent ones(otherwise rollback to the previous state)

*-**I**solation: every transaction works like it is the only one running (separated working environment)*

*-**D**urability: successful notified modifications, made by transactions, become persistent and no system failure can violate that.*

... NoSQL DB ...

Relational DB limits

- Scalability:
  - *-predefined data schema => relational databases maximize vertical scalability (e.g.: adding records to a table)*
  - *-performance increments related to HW upgrades (e.g: more CPU, more memory, high speed disks, SSD, etc...)*
  - *-performance are limited by available technologies*
- Availability:
  - *-redundancy techniques and systems at medium/high costs*
  - *-many servers usage (e.g.: active/passive, active/active o cluster)*
  - *-date sharing (external storage, redundant disks, redundancy also on controller, etc...)*
- Complexity:
  - *-data are organized in tables and records (not so flexible structure)*
  - *-records of a group have the same structure (=table structure)*
  - *-new data maybe added (e.g.: for a record) in two ways:*
    - *new column added to an existing table (wasting space...)*
    - *new table for only new data (adding more complexity...)*

... NoSQL DB ...
BASE (ACID alternative)

Approccio diametralmente opposto ad "ACID"

- *"ACID" => pessimistic approach, enforce consistency after each operation*
- *"BASE" => optimistic approach, partial loss of consistency is acceptable in some situation (consistency is granted eventually, at the end of flow of operations). Partial loss of availability is possible, but avoid total failure (using many nodes)*

- BASE
  - ***B**asically **A**vailable:*
    - *the system is generally available (partial failure managed, avoid total ones)*
  - ***S**oft State:*
    - *consistency not granted every time during the lifecycle*
  - ***E**ventually Consistent:*
    - *consistency at the end of an operations flow*

... NoSQL DB ...

NoSQL advantages

NoSQL databases may have several advantages.
The greater one is:

- Horizontal Scalability, may be achieved in two different ways

  -*Functional Scaling*:

  - data are organized in *functional groups* (usually through functions)
  - *functional groups* are distributed on different nodes
  - a functional group may itself be divided in *subgroups* ("*subfunctionals groups*")

    Example: *(users grp => male grp | female grp)*

-*Sharding:*

- a *functional group* is distributed in form of "*chunks*" on many cluster nodes. Chunks are so called "*shards*".
- advantages: *high availability, more write bandwidth, high throughput*

... NoSQL DB ...

CAP Theorem

- CAP "Theorem"

  Eric Brew (University of California, Berkley) concept expressed at PODC2000 (Principle of Distributed Computing symposium), about Web Services and then reused with databases related to modern web applications.

It's not possible to grant, at the same time, these three properties:

- ***C**onsistency*
- ***A**vailability*
- ***P**artition-Tolerance*

Only two of the above maybe achieved at the same time:

- *Usually designer choice is about consistency-availability tradeoff*
  - *Cassandra NoSQL DB for example has four consistency configuration that implies more or less restrictions on availability*
- *In case of NoSQL, partition tolerance is considered the default because every Horizontal scaling strategy is data-partitioning based*

... NoSQL DB ...

NoSQL disadvantages

- Reliability:
  - *many NoSQL DBs are not "reliable" at 100%*

- Consistency
  - NoSQL DBs may be not completely consistent in some situations

- No-Joins
  - *many NoSQL DBs don't have join features (Relations between objects must be maintained by the particular application)*

… NoSQL DB
Main Categories

- Key-Value Stores

  Amazon's SimpleDB, Redis, …

  -*Key-Value maps*
  -*key => is the index*
  -*value => is the value associated to the index (the value may also be a "set")*

- Column-oriented databases

  Google BigTable, Apache HBase, Cassandra, …

  -*Informations are stored by columns (closely related data)*
  -*Advantages: one single access to retrieve all data related to a column*
  -*Disadvantages: multiple accesses required to write a tuple*

- Document-based databases

  Apache's CouchDB, MongoDB, …

  -*data organized as document collections (no table concept exists)*
  -*each document may be different in structure (also in the same collection)*
  -*Advantages: No limits to vertical and horizontal scalability*
  -*Disadvantages: in each document there are also attribute name and values (expressed as key-value couple) … more space consumption.*

Google App Engine ...

What is?

- Google cloud solution

- PaaS cloud type:
  - *environment abstraction for application development*
  - *no fine-grained control over the IT infrastructure*

- Scalability granted in several ways:
  - *datastore capacity can grow (distributed data storage service)*
  - *CPU usage can grow related to requests increment (CPU is an elastic resource)*

- Free usage under some conditions:
  - *500MB storage*
  - *CPU and bandwidth enough to support an application serving around 5 million page views a month*

... Google App Engine ...

Features

- Features:

  -*dynamic web serving (common technologies supported...)*

  -*persistent transactional storage service (based on NoSQL db "Big Table")*

  -*authentication: using Google accounts or federated accounts (experimental)*

  -*a simulated local environment (useful to test all features in development phase)*

  -*task queues: to perform work in separate "threads" (outside request scope)*

  -*cron tasks: schedule based on specific time or regular intervals*

  -*auto-scaling and load balancing*

  -*xmpp and email services (using Google accounts)*

... Google App Engine ...

Sandbox

- Applications runs in protected environment
  - *Limited access to underlying operating system (to maximize scalability)*
  - *Hardware independent*
  - *Operating System independent*
  - *Physical location unaware*

- Limits imposed by this secure and scalable environment
  - *Access to resource outside of owned application only by Google provided services (URL fetch, email and other)*
  - *No access to the filesystem. Only use memcache, datastore and other services to persist data between requests*
  - *Code execution allowed only in response to a web, cron or task queue request*
  - *Response in 30 seconds (otherwise process stopped)*

... Google App Engine ...

Runtime environments

- Java
  - *Java SE Runtime Environment uses Java 6*
  - *Any library or bytecode respecting security limitations admitted (otherwise runtime exception thrown...for instance in case of opening a socket or writing to file)*
  - *Sandbox restrictions implemented directly in JVM*
  - *Standard persistence API: JPA and JDO*
  - *Standard Mail API: Java Mail*
  - *App Engine URL fetch service: access through standard java.net HTTP APIs*
  - *Low level APIs: memcache, images Google Accounts, etc ...*
- JVM based languages
  - *Ruby, Scala, Groovy*
- Python
  - *Environment base on 2.5.2 version (Python 3 for the future)*
  - *Python standard library (but following sandbox limitations),*
  rich data modeling API, URL fetch, email services, google accounts,
  web framework

... Google App Engine ...

Datastore (I)

- Distributed data storage service

-grows with data (over 500MB limit you have to activate business account)

-datastore is not relational

BigTable: sparse, distributed, persistent multi-dimensional sorted map;indexed by a row key, column key, and a timestamp

-transactional (optimistic concurrency control)

-datastore is schemaless

-based on entities (a record is an entity) and "entity groups" (entities organized in hierarchical way)

-transactions are at "entity groups" level: entities are manipulated within a single group

... Google App Engine ...
Datastore (2)

- Two types of datastore
  - master/slave
    - *default for new applications*
    - *asynchronous data replication to another data center (after write)*
    - *only one data center is the master for writing at any given time*
    - *strong consistency (on reads and queries)*
    - *data may be unavailable during data center planned/unplanned downtime*
    - *lowest CPU and storage costs*
  - high replication
    - *Highly available and reliable solution*
    - *Data always available, also during planned downtime*
    - *Resilient to failures*
    - *Data replicated across data centers*
    - *Higher availability (read and write) at higher cost (in write about 3x)*
    - *Entity groups are unit of consistency*

... Google App Engine ...

Datastore (3)

- Entity
  - *each datastore "record" is an entity*
  - *is usually represented with an object in the application code*
  - *has a "key" that uniquely identifies it across all entities*
- Key has several components
  - *kind: each entity is of a particular kind (a name specified bye the application)*
  - *name: it's an identifier assigned by the application or by the datastore*
  - *path: identifies entity relationships (useful to define an entity group)*

```
Entity carHolder = new Entity("Person"); //Key-kind-> Person, Key-name-> datastore assigned (e.g.: Person:3245)
datastore.put(carHolder);

Entity address = new Entity("Address", carHolder.getKey()); //Key-kind-> Address,
datastore.put(address);                                     //Key-name-> datastore assigned (e.g.: Address:75),
                                                            //Key-path-> carHolder

Entity suvCar = new Entity("Car", "Hummer_12654", carHolder.getKey()); //Key-kind-> Car,
datastore.put(suvCar);                                                 //Key-name-> Application assigned,
                                                                       //Key-path-> carHolder

The complete key for "address" can be represented as: Person:3245/Address:45
The complete key for "suvCar" can be represented as: Person:3245/Car:Hummer_12654
```

... Google App Engine ...

Services

- A variety of services are provided by App Engine
  - *Memcache: distributed in-memory data cache to improve performance (JCache JSR107)*
  - *Mail: app can send email using Mail service (JavaMail API) or receive mail messages as HTTP requests posted to the app by App Engine*
  - *Image Manipulation: provides image manipulation capabilities (rotate, resize, flip, crop and other transformations) via Java API*
  - *URL Fetch: used to fetch resources or communicate over internet using http or https*
  - *Scheduled Tasks: regularly scheduled tasks that trigger events at defined times or regular intervals*
  - *Task Queues: allow to perform background processing pushing task into a queue*
  - *XMPP: exchange instant messages with users of services compatible with XMPP (e.g.: GoogleTalk)*
  - *Blobstore: allows to store and serve data objects (usually files) larger than the maximum size allowed for Datastore service.*

... Google App Engine ...

Limits

- Limits of App Engine free account:
  - *Up to 10 applications per developer account*
  - *Up to 500 MB of storage per app*
  - *Up to 5 millions page views a month*
  - *Response to a web request is expected in 30 seconds. This timeout is dynamic (shortened for processes that often exceeds limit)*
  - *Process that exceeds timeout will be terminated (with a runtime exception)*
- Enabling billing can increase some limits
  - *usually rising the "Daily Limit" and/or the "Maximum Rate" (expressed in calls per minute)*

... Google App Engine ...
Development (I)

- App Engine SDK

  -*Emulates App Engine on local machine using a web server (some Services are not emulated, like cron service)*

  -*Includes a tool to upload applications to App Engine (newly created or updated versions)*

  -*SDK runs on Java 5 or Java 6 (the last one is preferred)*

- Google Plugin for Eclipse

  -*Allows to create, test and upload applications from within eclipse IDE*

  -*For Eclipse 3.6 (Helios) install the plugin using*

  `http://dl.google.com/eclipse/plugin/3.6`

  in "Help menu > Install New Software"

... Google App Engine ...
Development (2)

- First project with App Engine:

  -*Create new App Engine project*

  ```
  New Project > Google > Web Application Project
  ```

  -*App name and package*

  ```
  Project Name: Garage | Package name: it.aip.samples.garage
  ```

  -*Uncheck "Use Google Web Toolkit"*

  -*Submit with "Finish" button and base project will be created*

  -*The new project has a demo servlet (GarageServlet.java) and is JDO enabled (src/META-INF/jdo-config.xml)*

... Google App Engine ...
Development (3)

- Create new Servlet (DemoEntityServlet.java)

>Garage/src/it/aip/samples/garage/DemoEntityServlet.java

```
package it.aip.samples.garage;
...
public class DemoEntityServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    }
}
```

- Declare and map servlet in the "*web.xml*"

>Garage/war/WEB-INF/web.xml

```
...
<servlet>
    <servlet-name>DemoEntity</servlet-name>
    <servlet-class>it.aip.samples.garage.DemoEntityServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>DemoEntity</servlet-name>
    <url-pattern>/demoEntity</url-pattern>
</servlet-mapping>
...
```

... Google App Engine ...
Development (4)

- Add another entity related with a parent (“*entity group*”)

>Garage/src/it/aip/samples/garage/DemoEntityServlet.java

```
...
Entity address = new Entity("Address", "Address_6387", johnCarHolder.getKey());
address.setProperty("city", "New York");
datastore.put(address);
...
```

“address” entity key has a path property set to parent entity “johnCarHolder”

- Let’s try a query over created entity

>Garage/src/it/aip/samples/garage/DemoEntityServlet.java

```
...
Query overThirty = new Query("Person").addFilter("age", FilterOperator.GREATER_THAN, 30);
PreparedQuery prepQry = datastore.prepare(overThirty);
Iterator<Entity> overThirtyEntities = prepQry.asIterator();
...
```

Although “Person” entities have different properties, is possible to execute a query based on the “uncommon” property

... Google App Engine ...
Development (5)

- Printing out query result set and entity keys

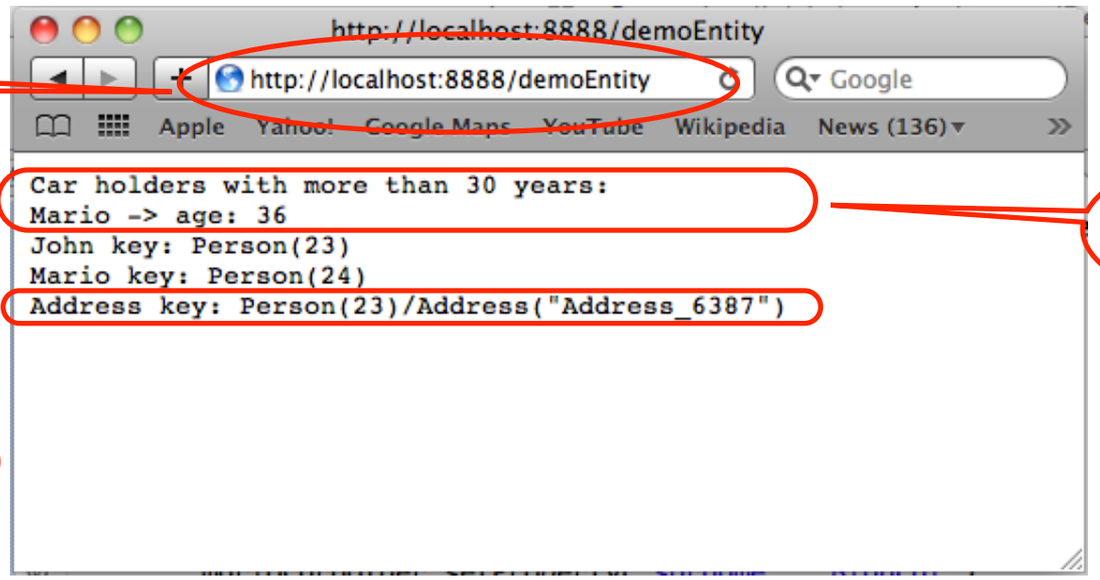
>Garage/src/it/aip/samples/garage/DemoEntityServlet.java

```
...
resp.setContentType("text/plain");
resp.getWriter().println("Car holders with more than 30 years: ");

while (overThirtyEntities.hasNext()){
    Entity entity = overThirtyEntities.next();
    resp.getWriter().println(entity.getProperty("name") + " -> age: " +
entity.getProperty("age"));
}

resp.getWriter().println("John key: " + johnCarHolder.getKey().toString());
resp.getWriter().println("Mario key: " + marioCarHolder.getKey().toString());
resp.getWriter().println("Address key: " + address.getKey().toString());
...
```

Run local server



Only Mario has age property and is <30

Address is part of an Entity Group

... Google App Engine ...

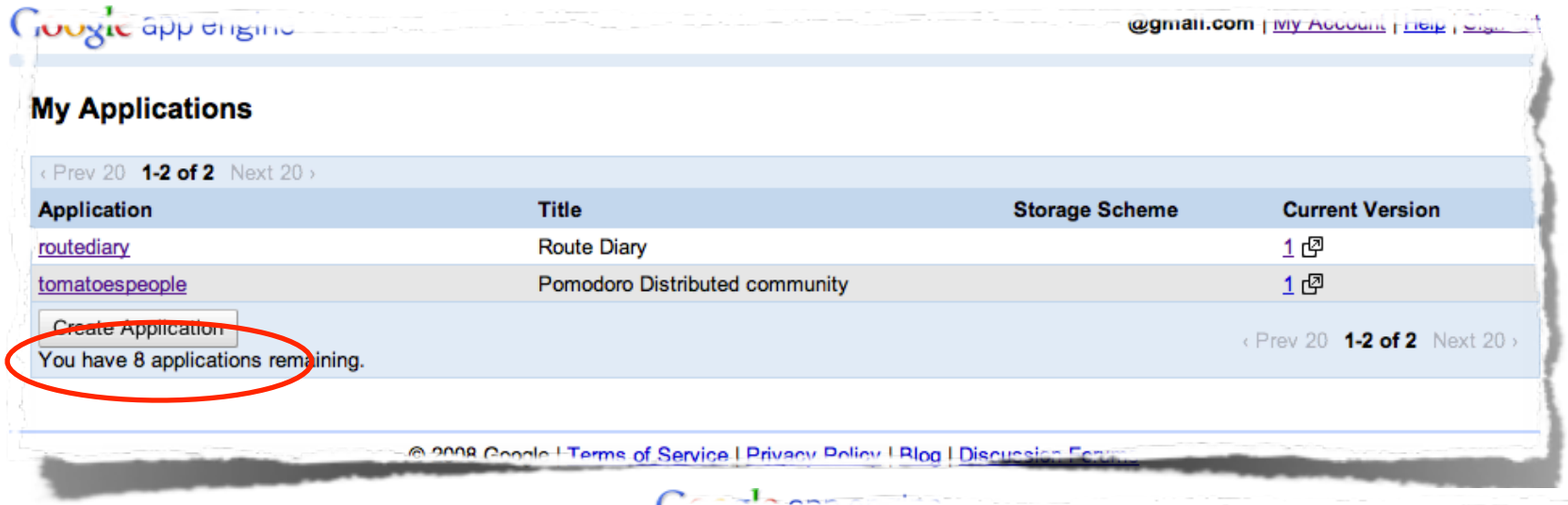
Development (6)

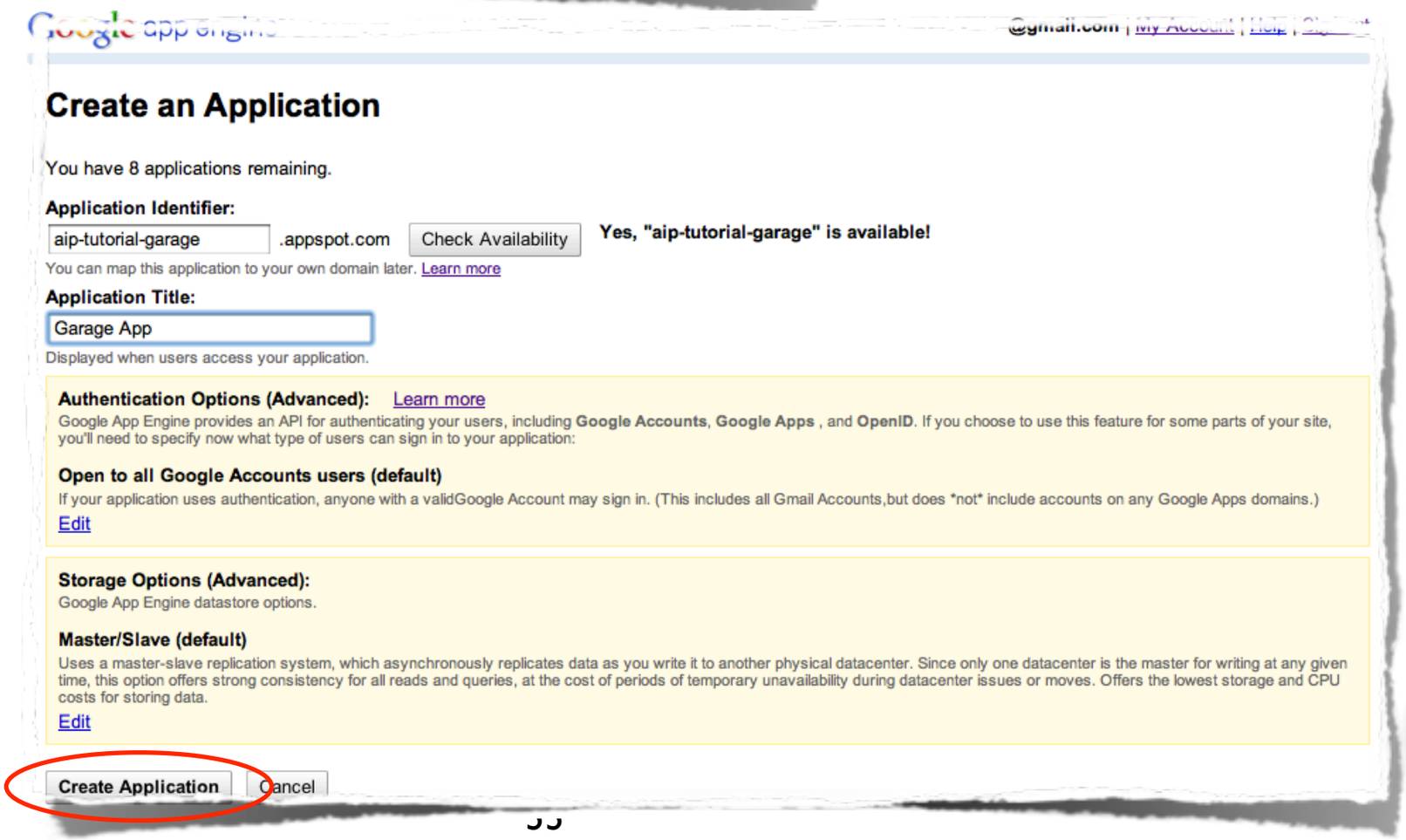
- A first look at the local admin page

... Google App Engine ...

Development (7)

- Create "aipTutorialGarage" app on App Engine

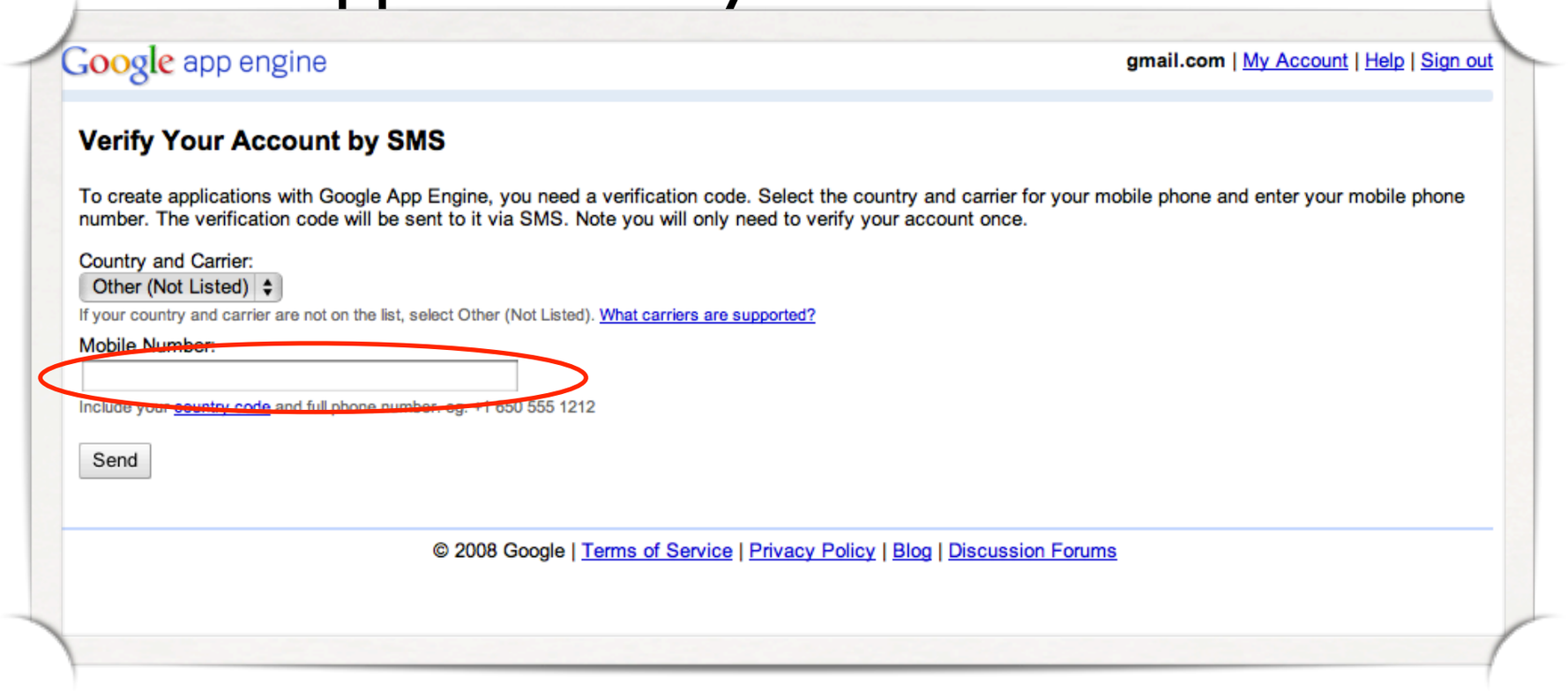




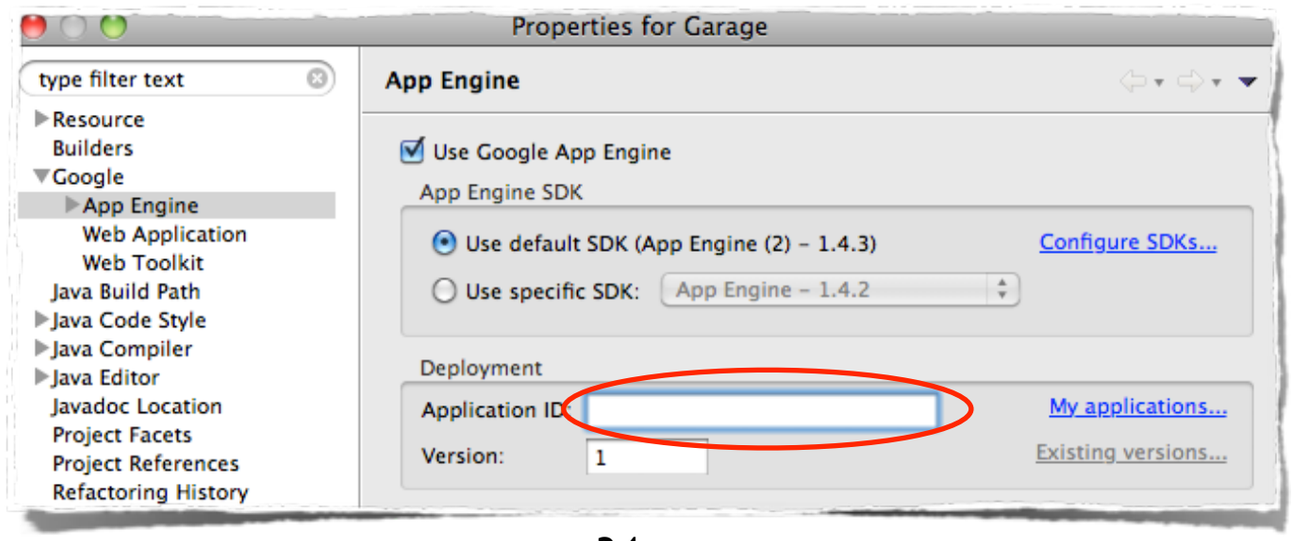
... Google App Engine ...

Development (8)

- For the first App creation you have to confirm with an sms pin



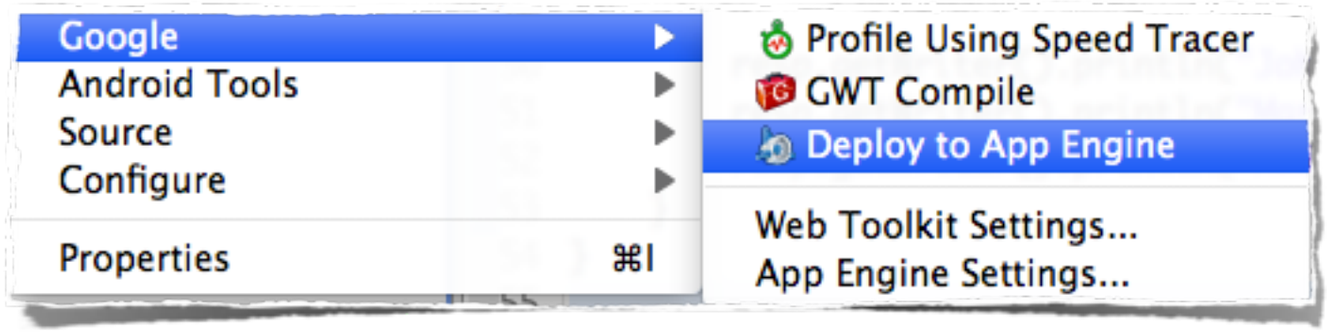
- Then set Eclipse to deploy your App

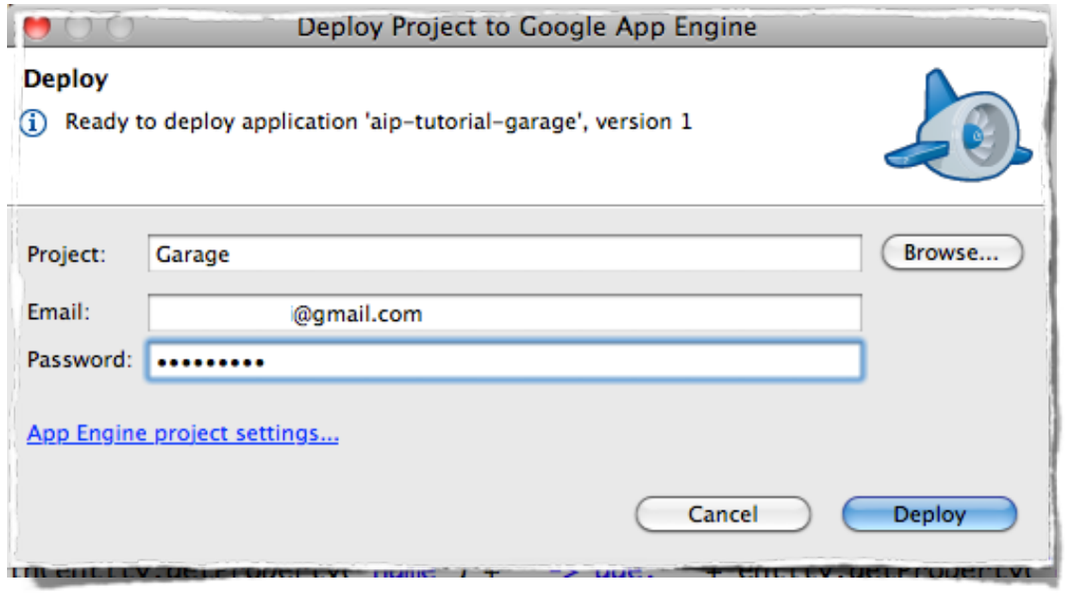


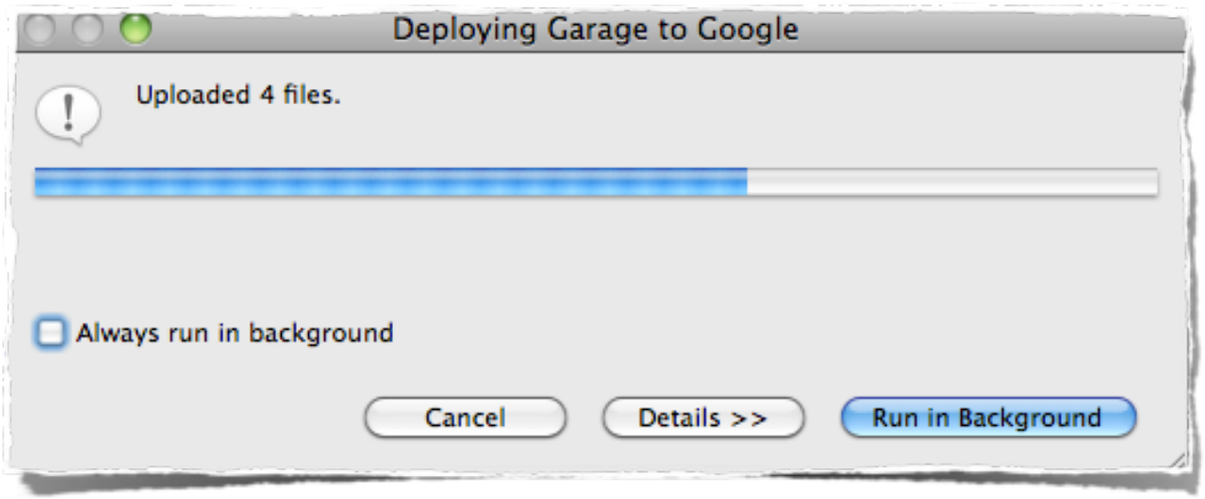
... Google App Engine ...

Development (9)

- Now you're ready to upload your app







... Google App Engine
Development (10)

- Check online the result

- And the dashboard