**Google app engine**

**Logging**

Your application can write information to the application logs using [java.util.logging.Logger](http://java.sun.com/javase/6/docs/api/java/util/logging/Logger.html).

Everything the servlet writes to the standard output stream (System.out) and standard error stream (System.err) is captured by App Engine and recorded in the application logs.

import java.util.logging.Logger;  
// ...  
  
public class MyServlet extends HttpServlet {  
    private static final Logger log = Logger.getLogger(MyServlet.class.getName());  
  
    public void doGet(HttpServletRequest req, HttpServletResponse resp)  
            throws IOException {  
  
        log.info("An informational message.");  
  
        log.warning("A warning message.");  
  
        log.severe("An error message.");  
    }  
}

The App Engine Java SDK includes a template logging.properties file, in the appengine-java-sdk/config/user/ directory. To use it, copy the file to your WEB-INF/classes directory (or elsewhere in the WAR), then the system property java.util.logging.config.file to "WEB-INF/logging.properties" (or whichever path you choose, relative to the application root). You can set system properties in the [appengine-web.xml file](https://developers.google.com/appengine/docs/java/config/appconfig), as follows:

<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">  
    ...  
  
    **<system-properties>  
     <property name="java.util.logging.config.file" value="WEB-INF/logging.properties" />  
    </system-properties>**  
  
</appengine-web-app>

The Google Plugin for Eclipse new project wizard creates these logging configuration files for you, and copies them to WEB-INF/classes/ automatically. For java.util.logging, you must set the system property to use this file.

# Storing Data

Various options

* **App Engine Datastore** provides a NoSQL schemaless object datastore, with a query engine and atomic transactions.
* [**Google Cloud SQL**](http://developers.google.com/cloud-sql/) provides a relational SQL database for your App Engine application, based on the familiar MySQL database.
* [**Google Cloud Storage**](https://developers.google.com/appengine/docs/java/googlestorage) provides a storage service for objects and files up to terabytes in size.

## The Datastore Java API

The Java Datastore SDK includes implementations of the [Java Data Objects](http://java.sun.com/jdo/index.jsp) (JDO) and [Java Persistence API](http://java.sun.com/developer/technicalArticles/J2EE/jpa/) (JPA) interfaces, as well as a low-level Datastore API.

## Introducing the Datastore

holds data objects known as entities. An entity has one or more properties, named values of one of several supported data types: for instance, a property can be a string, an integer, or a reference to another entity. Each entity is identified by its kind, which categorizes the entity for the purposes of queries, and a key that uniquely identifies it within its kind.

The Datastore can execute multiple operations in a single transaction(cannot succeed unless every one of its operations succeeds; if any of the operations fails, the transaction is automatically rolled back.).

### Data Repository

App Engine's primary data repository is the **High Replication Datastore (HRD),** in which data is replicated across multiple data centers using a system based on the [Paxos algorithm](http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/archive/paxos_made_live.pdf).

## Introducing the Java Datastore API

SDK provides a low-level Datastore API with simple operations on entities, including get, put, delete, and query.

The SDK also includes implementations of the [Java Data Objects](http://java.sun.com/jdo/index.jsp) (JDO) and [Java Persistence API](http://java.sun.com/developer/technicalArticles/J2EE/jpa/) (JPA) interfaces for modeling and persisting data.

In addition to the standard frameworks and low-level Datastore API, the Java SDK supports other frameworks designed to simplify Datastore usage for Java developers.

* [**Objectify**](http://code.google.com/p/objectify-appengine/) is a very simple and convenient interface to the App Engine Datastore that helps you avoid some of the complexities presented by JDO/JPA and the low-level Datastore.
* [**Twig**](http://code.google.com/p/twig-persist/) is a configurable object persistence interface that improves support for inheritance, polymorphism, and generic types. Like Objectify, Twig also helps you avoid complexities posed by JDO and the low-level Datastore.
* [**Slim3**](http://sites.google.com/site/slim3appengine/) is a full-stack model-view-controller framework that you can use for a wide variety of App Engine functions, including (but not limited to) the Datastore.

# Backends Java API Overview

App Engine Backends are instances of your application that are exempt from request deadlines and have access to more memory (up to 1GB) and CPU (up to 4.8GHz) than normal instances.

backend may be configured as either resident or dynamic.

Resident backends run continuously, allowing you to rely on the state of their memory over time and perform complex initialization

Dynamic backends come into existence when they receive a request, and are turned down when idle; they are ideal for work that is intermittent or driven by user activity.

Backends are [configured](https://developers.google.com/appengine/docs/java/config/backends) using either backends.xml or backends.yaml.

## Entities

Objects in the App Engine Datastore are known as entities. An entity has one or more named properties, each of which can have one or more values. Property values can belong to [a variety of data types](https://developers.google.com/appengine/docs/java/datastore/entities#Properties_and_Value_Types),

### Kinds, Keys, and Identifiers

Each Datastore entity is of a particular kind.

each entity has its own key, which uniquely identifies it.

The key consists of the following components:

* The entity's kind
* An *identifier,* which can be either
  + a *key name* string
  + an integer *numeric ID*
* An optional [*ancestor path*](https://developers.google.com/appengine/docs/java/datastore/overview#Ancestor_Paths) locating the entity within the Datastore hierarchy.

The identifier is assigned when the entity is created. Because it is part of the entity's key, it is associated permanently with the entity and cannot be changed. It can be assigned in either of two ways:

* Your application can specify its own key name string for the entity.
* You can have the Datastore automatically assign the entity an integer numeric ID.

(this low-level Java API;)package [com.google.appengine.api.datastore](https://developers.google.com/appengine/docs/java/javadoc/com/google/appengine/api/datastore/package-summary), supports the features of the Datastore directly.

In addition to the low-level Java API, the SDK also supports two standard Java interfaces for data storage, [Java Data Objects (JDO)](https://developers.google.com/appengine/docs/java/datastore/jdo/overview) and the [Java Persistence API (JPA)](https://developers.google.com/appengine/docs/java/datastore/jpa/overview),

### Ancestor Paths

**Queries and Indexes**

Every query uses an index, a table containing the query's potential results in the desired order. The Datastore updates the indexes incrementally to reflect any changes the application makes to its entities. Thus the correct results of all queries are immediately available directly from the indexes, with no further computation needed.

Indexes for some types of query are provided automatically; an application can define additional indexes for itself in an index configuration file named datastore-indexes.xml.

**Transactions**

Every attempt to create, update, or delete an entity takes place in the context of a [*transaction*](https://developers.google.com/appengine/docs/java/datastore/transactions)