TP 4 : Stockage de données dans le cloud (datastore)

Objectif du TP :

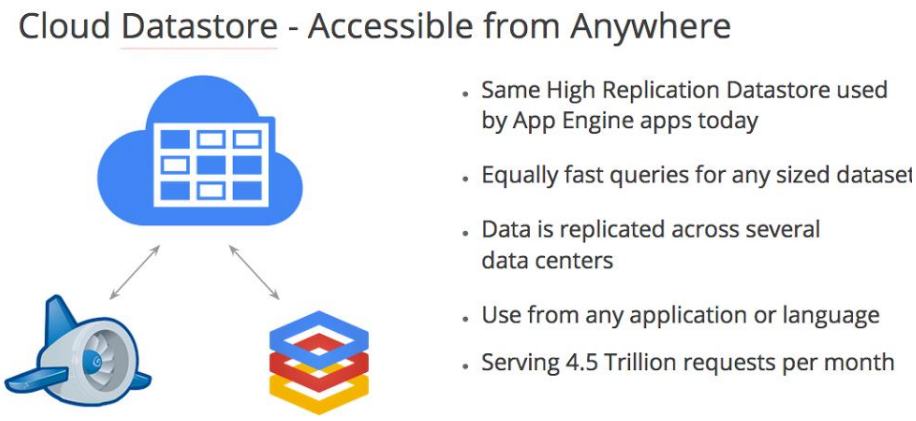
Utiliser l'API Cloud datastore pour gérer des données sur un site web

Travail demandé :

- Suivre les instructions de la partie Enoncé ci-dessous pour réaliser le travail demandé
- Rédiger un **compte rendu** qui permet d'expliciter les étapes nécessaires pour achever le TP.

Mots clés :

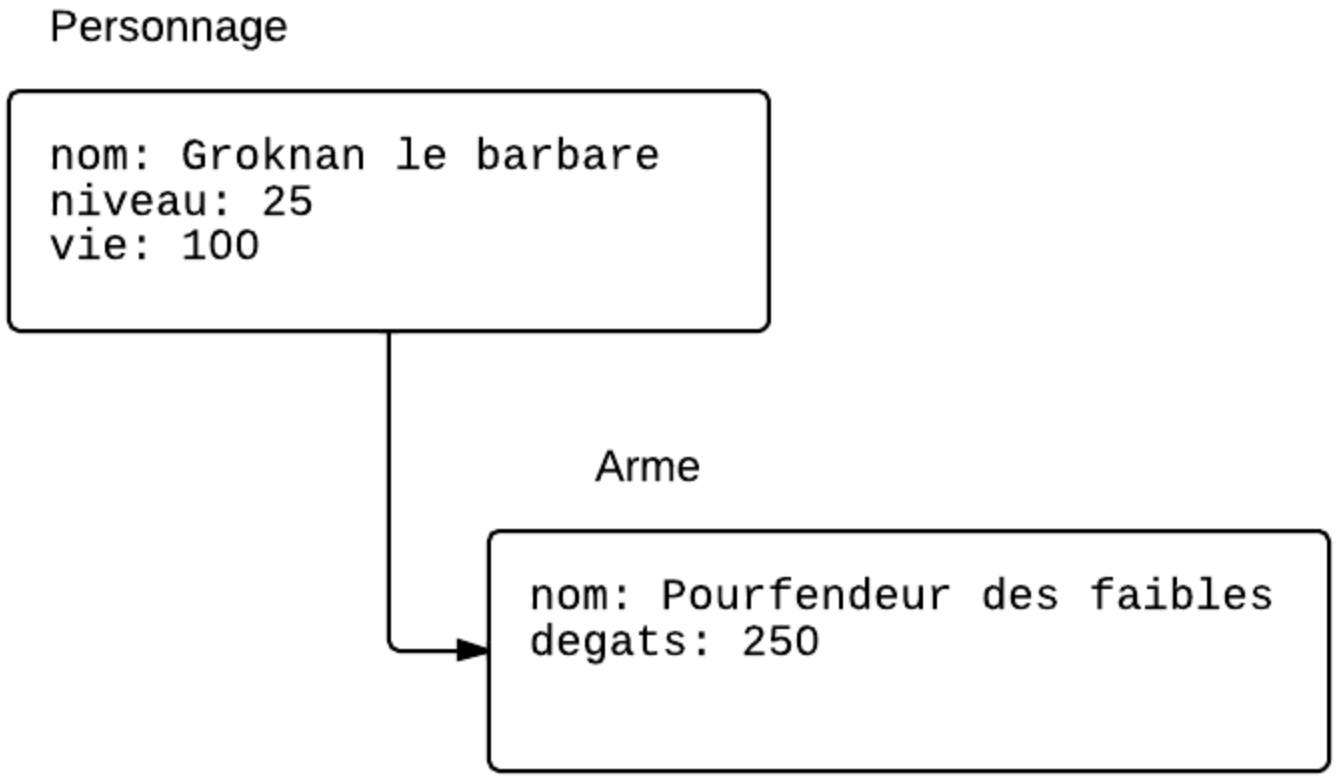
- ✓ **Cloud**
- ✓ **Datastore**
- ✓ **NoSQL**
- ✓ **Entity**
- ✓ **Get**
- ✓ **Post**



Introduction au Datastore

Le Datastore est la nouvelle technique pour stocker des données sur Google App Engine. C'est clairement l'outil que Google nous conseille d'utiliser. Vous pouvez vous en servir gratuitement (jusqu'à une certaine limite de trafic) et on ne vous en dit que du bien : il serait rapide, capable de gérer de fortes charges, réparti sur de multiples datacenters. On dit même qu'il serait basé sur Big Table, la technologie qu'utilise Google pour stocker toutes ses données, notamment les résultats d'indexation des milliards de pages web par le GoogleBot. Le Datastore fonctionne en mode High Replication Datastore (HRD). Ce système est très résilient aux erreurs : il fait en sorte que vos données soient copiées sur de multiples datacenters ! L'objectif est que votre application puisse continuer à fonctionner même en cas d'évènement catastrophique. Le Datastore peut être vu comme un espace géant prêt à accepter des données. N'importe quelles données. Vous lui envoyez du contenu et il vous le stocke. Quand vous en avez à nouveau besoin, il vous suffit de le lui demander. Le Datastore n'utilise pas de tables. Il n'y a pas de schéma SQL : on dit que le Datastore est *schemaless* (sans schéma). Le Datastore peut être utilisé de façon massive, avec beaucoup de données et de requêtes simultanées. Pourtant, à son plus bas niveau, il fonctionne de façon très simple : il accepte des clés et des valeurs. Exactement comme

une HashMap (table de hachage). Le Datastore permet de regrouper les données dans des *entités*. Les entités sont des regroupements d'une ou plusieurs paires clé-valeur. Pour reprendre un exemple, imaginez un Personnage d'un jeu vidéo. Il possède des attributs : un nom, un niveau de vie,...etc, Il possède aussi une Arme qui a ses propres caractéristiques. Les entités dans le Datastore pourraient être représentées comme ceci :



Les entités sont comme des objets

NoSQL (*Not only SQL* en anglais) désigne une catégorie de systèmes de gestion de base de données (SGBD) qui n'est plus fondée sur l'architecture classique des bases relationnelles. NoSQL proposé par des acteurs majeurs du Web comme Google, Facebook et Twitter, a rapidement acquis une légitimité réelle dans le domaine de la manipulation de volumétries de données très importantes et ont rapidement gagné tant en maturité qu'en notoriété.

Les besoins majeurs identifiés par ces acteurs (google, facebook et twitter) sont les suivants :

1. Capacité à distribuer les traitements sur un nombre de machines important afin d'être en mesure d'absorber des charges très importantes. On parlera de scaling des traitements
2. Capacité à répartir les données entre un nombre important de machines afin d'être en mesure de stocker de très grands volumes de données. On parlera de scaling des données

3. La distribution de données sur plusieurs datacenters afin d'assurer une continuité de service en cas d'indisponibilité de service sur un datacenter.

Les opérations sur le datasore

Il y a 4 opérations de base disponibles dans une base de données NoSQL (Not only SQL) comme le Datastore :

- get() : récupère une entité en fonction de son identifiant
- put() : écrit une entité (ou en écrase une)
- delete() : supprime une entité via son identifiant
- query() : retrouve des entités en fonction de certains critères de recherche

Voilà, vous en savez assez sur les concepts du Datastore. On va passer au code maintenant.

Travail demandé:

On veut créer une application rudimentaire pour récupérer des messages de la part des utilisateurs sur un thème bien déterminé. On propose à l'utilisateur un formulaire où il saisit son nom et un message puis il valide ces entrées. Une fois validée les informations peuvent être affichées au-dessous de la page web (voir figure1).

Figure 1. Interface de saisie (modèle MVC)

1. Lancer eclipse et créer un nouveau projet (New Web Application Project) à l'aide de l'icône " g " .
2. Modifier le fichier web.xml comme suit :

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
version="2.5">
<servlet>
  <servlet-name>Guestbook</servlet-name>
  <servlet-class>guestbook.GuestbookServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Guestbook</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
<welcome-file-list>
   <welcome-file>guestbook.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

L'application fonctionne sur une seule URL, la racine "/". Avec la méthode HTTP GET, on affiche le formulaire et les derniers messages. Avec la méthode HTTP POST, on enregistre le message et on redirige vers la même page en GET pour lister les résultats.

3. Changer le contenu du servlet avec le code suivant :

```
package guestbook;
import java.io.IOException;
import java.util.Date;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.*;
import com.google.appengine.api.datastore.*;
import com.google.appengine.api.datastore.Query.SortDirection;

@SuppressWarnings("serial")
public class GuestbookServlet extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse resp) {
      try {
        DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();

         // Demande tous les 5 derniers messages triés par date décroissante
```

```
 Query q = new Query("Message").addSort("date", SortDirection.DESCENDING);
  List<Entity> results = datastore.prepare(q).asList(FetchOptions.Builder.withLimit(5));
  req.setAttribute("messages", results);
  this.getServletContext().getRequestDispatcher("/WEB-
INF/guestbook.jsp").forward(req, resp);
        } catch (ServletException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp) {
        try {
          DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();

            // Stocke le message posté
            Entity message = new Entity("Message");

            message.setProperty("name", req.getParameter("name"));
            message.setProperty("message", req.getParameter("message"));
            message.setProperty("date", new Date());

            datastore.put(message);

            resp.sendRedirect("/");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Il n'y a absolument rien de compliqué dans ce code : on crée un seul type d'entité ("Message") et on le stocke avec 3 propriétés. A l'affichage, la requête récupère les derniers messages et les transmet à une page JSP qu'on va créer. Celle-ci n'a ensuite plus qu'à itérer sur cette liste d'entités.

4. Créer une page guestbook.jsp et saisir le code suivant :

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="com.google.appengine.api.datastore.*" %>
<%@ page import="java.util.List" %>

<!DOCTYPE html>
<html>
  <head>
      <title>Livre d'or</title>
      <meta charset="utf-8" />
</head>
<body>
```

```
  <h1>Vous avez aimé mon site ? Dites-le !</h1>

<form action="/post" method="post">
<p>
<label>Votre nom : <input type="text" name="name" /></label>
</p>
<p>
<label>Votre message :
<textarea name="message" style="width: 200px; height: 100px;"></textarea>
</label>
</p>
<p>
<input type="submit" />
</p>
</form>
<h1>Ils ont aimé :</h1>
<p><em>(et c'est stocké dans le Datastore !)</em></p>
<%
List<Entity> messages = (List<Entity>) request.getAttribute("messages");
for (Entity message : messages) {
%>
<p>
<strong><%= message.getProperty("name") %></strong> a écrit :
<%= message.getProperty("message") %>
</p>
<%
}
%>
</body>
</html>
```

5. Pour exécuter votre application, choisir Run->Debug as->Web Application, Eclipse va compiler votre application. Pour voir le résultat aller sur l'adresse :http ::// localhost :8888. Saisir des données et valider vos saisies.

6. Aller sur l'adresse : http://localhost:8888/_ah/admin, sélectionner les entités messages, de combien d'entités votre datastore est composé ?

7. Aller au site :http://console.developers .google.com, créer un nouveau projet intitulé TP datasore, le serveur google donne à votre projet un identifiant (voir le champ ID). Revenir à Eclipse et changer le fichier appengine-web.xml, dans la balise <application>…</application> mettre votre identifiant google du projet déjà créé, exemple <application>tp-datastore</apllication>. Cliquer sur l'icône de google et choisir Deploy to App Engine. Google va donner une adresse publique à votre application de type : http://1-dot-tp-datastore.appspot.com/.