Prise en main de Google App Engine

Ce TP propose de prendre en main Google App Engine en créant une application Web. Il est nécessaire pour ce TP d'avoir Java 7+ et maven d'installé. Il faut également un compte chez Google.

Une documentation assez complète relativement à l'utilisation de Java dans Google App Engine est disponible ici: https://cloud.google.com/appengine/docs/java/.

Une application Web basique

Cet exercice est en fait un complément d'informations au tutorial officiel de Google[1]. Attention, il y a plusieurs centaines de mégaoctets de dépendances à télécharger. Il faut donc prévoir du temps ou une bonne connexion.

1. Générez un squelette d'application depuis l'archetype *com.google.appengine.archetypes:appengine-skeleton-archetype*
    - Prenez les dernières versions pour les dépendances. Actuellement, pour gcloud, il s'agit de la version *2.0.9.121.v20160815*
2. Finalisez l'installation avec un *mvn clean install*.
3. *mvn appengine:devserver* pour lancer l'application en local.
    - Visitez http://localhost:8080 pour visiter la page exemple
    - Visitez http://localhost:8080/_ah/admin pour visiter la console de l'application
4. Pour déployer
    - Créez votre projet sur la console de google app engine[2] et notez le nom du projet qui a été généré
    - Raportez le nom du projet dans la variable *app.id* décrite dans le fichier *pom.xml*
    - *mvn appengine:update* va réaliser le déploiement et mettre le site en ligne
    - Une fois en ligne, visitez la console google pour voir l'étendu des possibilités.

Ajout d'une route REST

1. Regardez la classe *HelloAppEngine* et le fichier *web.xml* pour comprendre comment une requête HTTP est récupérée par l'application.
2. Créez une classe métier *Article* avec 3 attributs (*nom, prix, quantité)*, les getters/setters, le constructeur vide et le constructeur complet.

---

[1] https://cloud.google.com/appengine/docs/java/tools/maven
[2] http://console.cloud.google.com

3. Créez une nouvelle *HttpServlet* nommé *Articles* et doit retourner une liste d'articles, créée à la volée en réponse à la requête *GET /articles/.* Testez
4. Pour le moment, la réponse retournée n'est pas réellement utilisable. Pour résoudre ce problème, sérialisez l'objet en JSON. Par défaut, App Engine inclut une version spéciale de GSON[3] pour vous aider.
5. Ajouter une nouvelle route REST permettant d'envoyer un article avec la requête *POST /articles/.* Le corps de la requête contiendra le message JSON décrivant l'article. Pour tester le code, créez un objet *Article* avec GSON depuis la requête et affichez le en réponse.

Elasticité horizontale

Nous allons maintenant aborder les règles d'élasticité pour le service. Une documentation assez complète des possibilités est disponible ici: https://cloud.google.com/appengine/docs/java/config/appref#scaling_elements

1. Définissez une nouvelle route REST qui nous servira d'injecteur de charge. Insérez de la latence avec un *Thread.sleep(50)* dans le code du traitement de la requête pour test.
2. Modifiez le fichier *appengine-web.xml* pour définir une élasticité automatique. Les valeurs exemples de la documentation pourront être suffisante
3. Testez en local la route REST et l'injection de la latence puis déployez
4. Utilisez *ab* par exemple pour injecter de la charge et observer l'élasticité de l'application sur la console de App Engine. Le principe consiste à faire une première injection avec un faible niveau de concurrence, puis d'en lancer d'autres en parallèle toute les minutes par exemple.

Communication asynchrone

Pour le moment, l'application fonctionne de manière synchrone. Pour communiquer de manière asynchrone, App Engine met à disposition des files d'attentes de type *push* et *pull*[4]. Pour ce TP, nous nous contenterons des files de type *push.*

Les files *push* laissent à AppEngine les décisions d'élasticité. AppEngine observera les paramètres de la file et la charge pour inférer le nombre d'instances.

1. Créez une file d'attente ayant un débit de 5 tâches par seconde[5].
2. Créez une servlet mappé sur une route */push* qui enfilera des objets *Article* sérialisé[6]
3. Testez et vérifiez dans la console en local que les tâches sont bien enfilées

[3] https://github.com/google/gson/blob/master/UserGuide.md
[4] https://cloud.google.com/appengine/docs/java/taskqueue/
[5] https://cloud.google.com/appengine/docs/java/taskqueue/push/creating-push-queues
[6] https://cloud.google.com/appengine/docs/java/taskqueue/push/

4. Créez un handler qui affichera l'article injectera une latence de 500ms par un *Thread.sleep*[7]. Testez
5. Déployez le code et utilisez un injecteur de charge pour voir dans l'interface des queues, les messages qui circulent, puis le nombre d'instances qui varie en fonction de la charge

Pull queue

Les files *pull* laissent plus de contrôle au développeur. Elles laissent programmeur notamment le contrôle de l'élastcité et de l'opération de defilement. Il est également possible de *tagguer* chaque tâche et de défiler des tâches par tag.

Et après

En plus des files *pull* et *push,* App Engine met à disposition plusieurs API pour l'envoie de mails, de SMS, pour le stockage structuré ou non, …
Regardez la console de App Engine et la documentation en ligne:
https://cloud.google.com/appengine/docs/java/

[7] https://cloud.google.com/appengine/docs/java/taskqueue/push/creating-handlers