

TP4 - Tests et diagnostique

Infrastructure de production

MONITORER UNE APPLICATION

L'objectif de ce TP est d'effectuer un test de performance (robustesse) et de diagnostiquer une erreur sur l'application petclinic. Ce TP peut être réalisé sous windows ou sous Docker si ce dernier fonctionne. Tous les outils doivent être lancé en tant qu'administrateur pour fonctionner

- 1 Infrastructure de production
 - 1.1 Les éléments de notation
 - 1.2 Préparation du test de performance
 - 1.2.1 Installation
 - 1.2.2 Préparation du diagnostic
 - 1.2.3 Déployer l'application Exemple
 - 1.2.4 Visualiser la mémoire et le garbage collector
 - 1.2.5 Créer le scénario de test

Les éléments de notation

Répondez au [questionnaire](#) qui demande un certain nombre de fichier et de ligne de commande

Préparation du test de performance

Installation






















Pour réaliser ce test il faut :

1. Un machine virtuel java
2. Un tomcat installé
3. Jmeter, téléchargeable à l'adresse [ici](#)

Préparation du diagnostic

Pour le diagnostic les outils nécessaires sont :

1. une machine virtuel java, presque tous les outils sont fourni avec
2. Des plugins pour VisualVM (Tools > plugins)

Install	Name	Category	Source
<input type="checkbox"/>	Startup Profiler	Profiling	
<input type="checkbox"/>	BTrace Workbench	Profiling	
<input type="checkbox"/>	VisualVM-Security	Security	
<input checked="" type="checkbox"/>	Visual GC	Tools	
<input type="checkbox"/>	SAPPlugin	Tools	
<input type="checkbox"/>	VisualVM-BufferMonitor	Tools	
<input type="checkbox"/>	SysTray	Tools	
<input checked="" type="checkbox"/>	Threads Inspector	Tools	
<input type="checkbox"/>	VisualVM-JConsole	Tools	
<input type="checkbox"/>	VisualVM-MBeans	Tools	
<input type="checkbox"/>	VisualVM-JvmCapabilities	Tools	
<input type="checkbox"/>	VisualVM OSGi Plugin	Tools	
<input type="checkbox"/>	KillApplication	Tools	
<input checked="" type="checkbox"/>	Tracer-Jvmstat Probes	Tracer	
<input checked="" type="checkbox"/>	Tracer-Monitor Probes	Tracer	
<input type="checkbox"/>	Tracer-Swing Probes	Tracer	
<input type="checkbox"/>	Tracer-IO Probes	Tracer	
<input type="checkbox"/>	Tracer-Collections Probes	Tracer	
<input type="checkbox"/>	Tracer-JavaFX Probes	Tracer	
<input checked="" type="checkbox"/>	Tracer-JVM Probes	Tracer	
<input type="checkbox"/>	OQL Syntax Support	UI	

3. Installer les plugins visualGC, ThreadInspector, Tracer monitor probes

4. MAT (Memory Analyser Tools) téléchargeable [ici](#)

Déployer l'application Exemple

Il y a deux possibilités pour réaliser le test de performance. La première possibilité est de tous faire dans windows avec un tomcat installé en local et jmeter aussi. La seconde est d'utiliser linux.

1. Installer tomcat et télécharger le war ([petclinic.war](#)) et installé le dans le répertoire webapps
2. Vérifier la quantité de mémoire (-Xms, -Xmx) alloué pour le processus java dans le fichier catalina (variable JAVA_OPTS ou CATALINA_OPTS et mettre -Xmx64m) et mettre le dump memoire lors du out of memory (-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/tmp/dump)
3. Démarrer l'application

sous linux,

1. Installer l'image tomcat car elle est fourni avec la jdk et non la jre comme les images officielles :

```
$ docker pull tomcat:8.5-jdk8-slim
```

2. lancer la machine docker, les éléments de lancement sont:

- a. JAVA_OPTS="-Xmx64M -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/usr/local/tomcat/temp"
- b. CATALINA_OPTS="-Dcom.sun.management.jmxremote.port=8090 -Dcom.sun.management.jmxremote.rmi.port=8091 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.local.only=false -Djava.rmi.server.hostname=[mettre l'ip du host docker]"
- c. utiliser le war fourni
- d. Le tomcat est installé dans /usr/local/tomcat

Visualiser la mémoire et le garbage collector

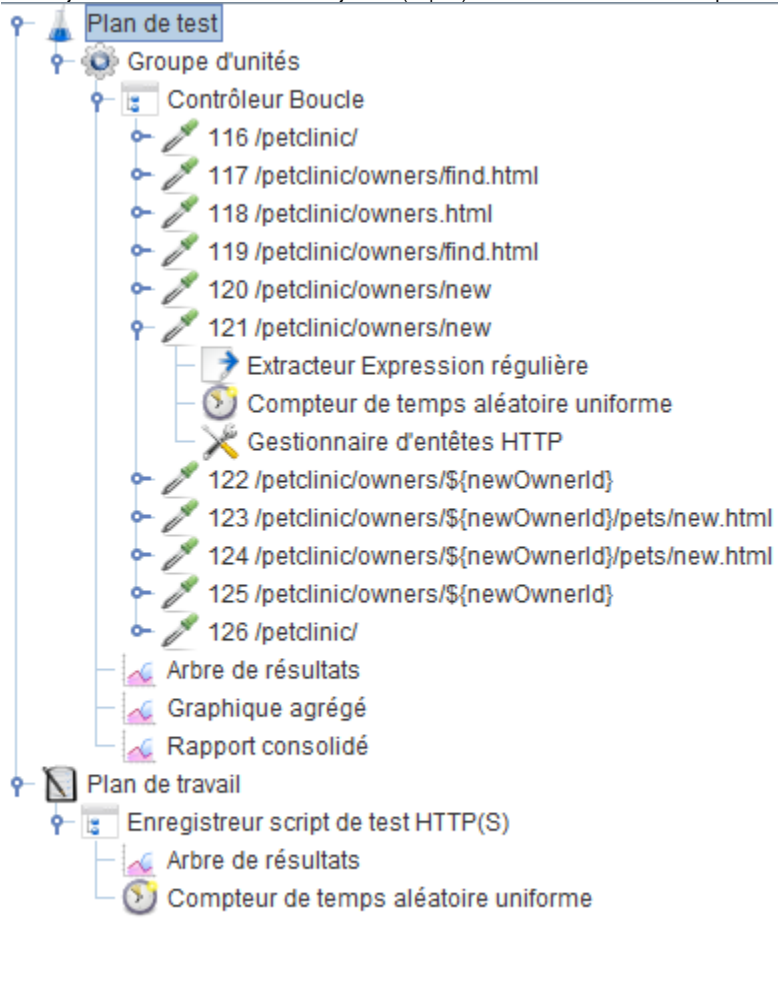
Avec VisualVM consulter les informations concernant la JVM. L'url de connection JMX est de la forme "service:jmx:rmi:///jndi/rmi://10.0.75.1:8090/jmxrmi"

Créer le scénario de test

1. Pour avoir jmeter en francais :

```
export JMeter_LANGUAGE="-Duser.language=fr -Duser.region=FR"
```

2. Lancer jmeter avec la commande `/bin/jmeter.(sh|bat)` selon votre environnement pour avoir :



Le scenarii a effectuer :

- a. Rechercher tous les propriétaires d'animaux
 - b. Après le résultat retournez sur la page de recherche
 - c. Ajoutez un nouveau propriétaire
 - d. Ajoutez un nouvel animal a ce propriétaire
 - e. Retournez sur la page de recherche et recherchez tous les propriétaires
3. La première étape est de mettre en place l'enregistrement de votre scénario pour vous aider : https://jmeter.apache.org/usermanual/jmeter_proxy_step_by_step.html

Attention, penser a supprimer les informations de localhost et de 127.0.0.1 dans la configuration du proxy

Proxy SSL localhost Port 8888

Proxy ETP localhost Port 8888

Hôte SOCKS localhost Port 8888

☐ SOCKS v4 ☒ SOCKS v5

Pas de proxy pour

localhost, 127.0.0.1

Exemples : .mozilla.org, .asso.fr, 192.168.1.0/24

☐ Adresse de configuration automatique du proxy

Aide Annuler OK

4. Ensuite il faut jouer le scénario pour enregistrer toutes les pages

5. Variabiliser le nom du propriétaire (\${__RandomString(10,ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789,)})

Paramètres Corps de la requête Téléchargement de fichiers

Envoyer les paramètres avec la requête :

Nom :	Valeur :
firstName	Diogène
lastName	\${__RandomString(10,ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789,)}
address	1 rue Philippe le Bel
city	45000 ORLEANS
telephone	066666666

6. Extraire l'identifiant du nouveau propriétaire

Portée

☐ Corps ☐ Corps (non échappé) ☐ Corps en tant que Document ☐ Entêtes (Réponse) ☐ Entêtes (Requête) ☒ URL

Nom de référence : newOwnerId

Expression régulière : /petclinic/owners/(id+)

Canevas : \$1\$

Récupérer la Nième corresp. (0 : Aléatoire) : 1

Valeur par défaut : error ☐ Utiliser la chaîne vide comme valeur par défaut

7. Lancer un tir de performance

Propriétés du groupe d'unités

Nombre d'unités (utilisateurs) : 15

Durée de montée en charge (en secondes) : 30

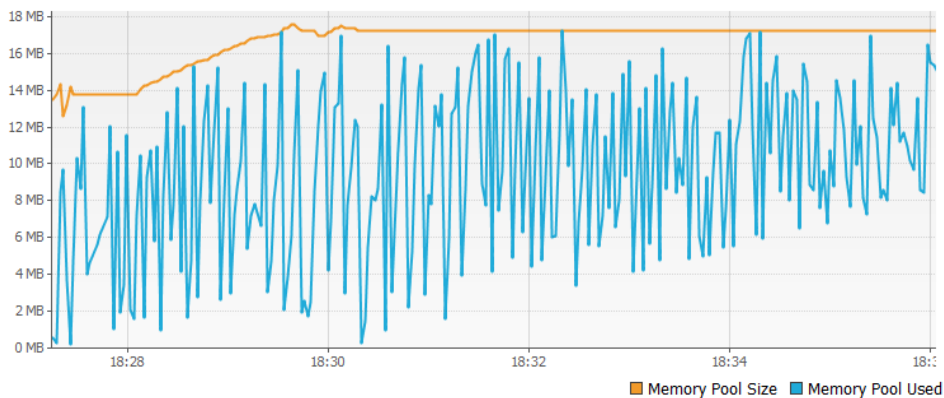
Nombre d'itérations : ☐ Infini 50

☐ Créer les unités seulement quand nécessaire

☐ Programmeur de démarrage

Lancer un premier tir avec 50 itération et un second en infini

8.



L'eden grossit et possède de plus en plus d'objet

9. Les tests ne doivent pas fonctionner. vous avez l'erreur :

```
Exception: java.lang.OutOfMemoryError thrown from the UncaughtExceptionHandler in thread "RMI TCP Connection(idle)"
java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "RMI TCP Connection(idle)" java.lang.OutOfMemoryError: GC overhead limit exceeded
```

10. Effectuer un heapdump de la mémoire pour analyse si le dump n'a pas été fait automatiquement

```
docker exec -it [identifiant de votre docker] su -m root -c '/usr/local/openjdk-8/bin/jmap -dump:
format=b,file=/usr/local/tomcat/temp/heapDump.hprof 1'
```

11. Récupérer le heap dump

```
docker cp [identifiant de votre docker]:/usr/local/tomcat/temp/heapDump.hprof .
```

12. Faites la même chose avec un threaddump. La commande java est :

```
jstack [PID du tomcat] > /usr/local/tomcat/temp/threadDump.tdump
```

13. La commande pour connaître le PID d'un processus sous linux (commande pour ps dans docker tomcat apt-get update && apt-get install procs)

```
ps -ef | grep java | awk '{print $1}'
```

14. Analyser le dump et trouvez la cause de la fuite de mémoire

15. Rechercher dans le [source](#) l'erreur