INFRASTRUCTURE DE PRODUCTION

TEST ET DIAGNOSTIQUE

00.00.2017

Photo by Diogène Mouton on Unsplash

TABLE DES MATIÈRES

Photo by Martha Dominguez de Gouveia on Unsplash


1
TEST DE CHARGE
Photo by Diogène Mouton on Unsplash

TEST DE PERFORMANCE

- **Pourquoi tester ?**
  - Le premier objectif des tests de performance est de rassurer les utilisateurs et la DSI
  - Le deuxième est de prévenir les dérives en utilisation réelle
  - Le dernier est de corriger les problèmes liée a la monté en charge

- **Les tests de charges permettent donc de**
  - Ajuster les paramètres de configuration de l'application
  - Ajuter et de calculer l'augmentation du volume des données
  - Anticiper la consommation de bande passante

TEST DE PERFORMANCE

- **Quand tester ?**
  - Le plus tôt possible

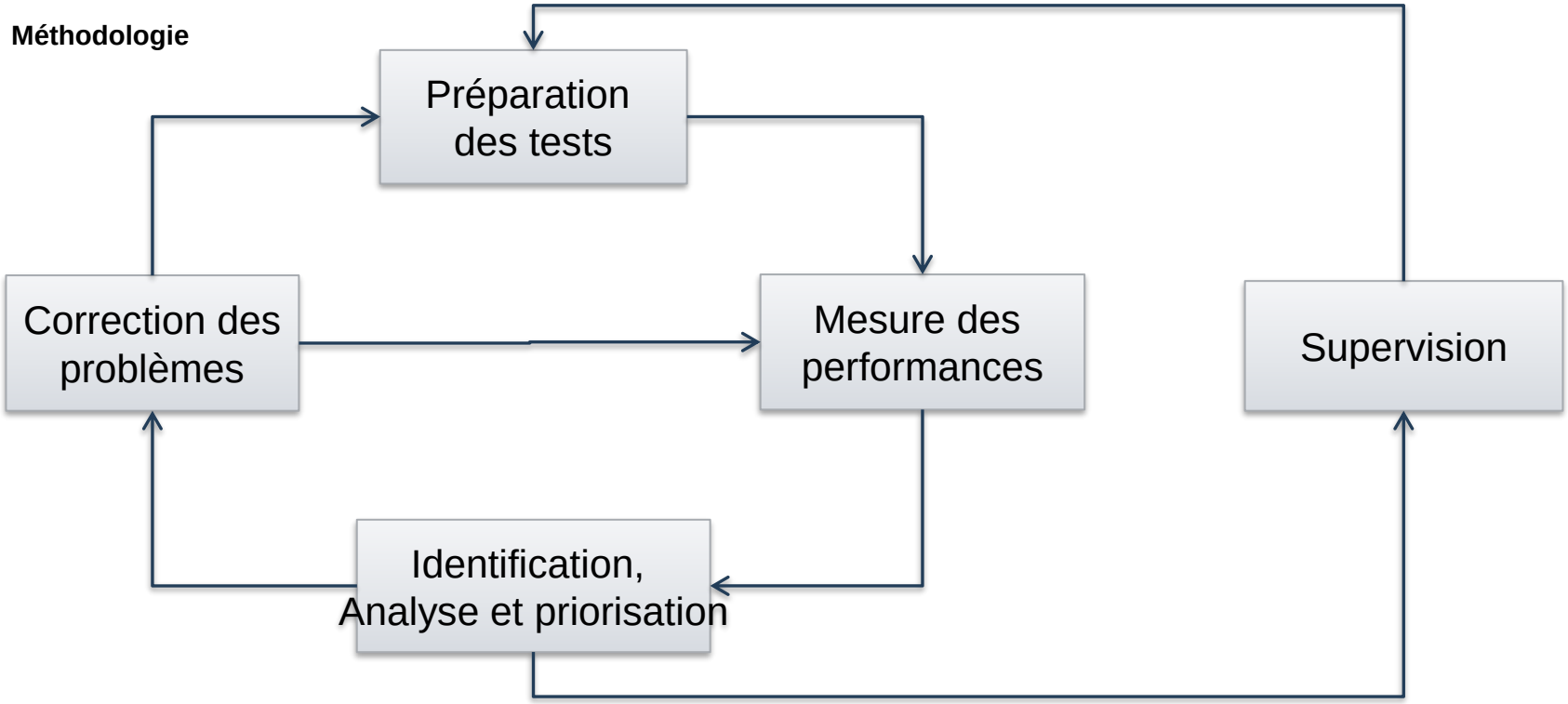
- **Lors de la phase de lancement cela permet :**
  - Valider l'architecture
  - Définir la volumétrie des données ,leur répartition et leur typologie
- **Lors de l'élaboration :**
  - Mise au point des tests et élimination des risques
- **En production :**
  - Supervision / Monitoring

TEST DE PERFORMANCE

- **Les différents type de test de performance**
  - Montée en charge
    assurer le bon fonctionnement en condition normale. En général, un test de 1 à 2 heurs suffi.
  - Test de rupture
    De déterminer la limite approximative à partir de laquelle l'application ne répond plus suffisamment correctement
  - Test d'vieillissement
    Déterminer et valider le comportement de l'application sur la durée. dégradation des temps de réponses, fuite de mémoire ou blocage

TEST DE PERFORMANCE

- **Méthodologie**



TEST DE PERFORMANCE – MAÎTRISE DE LA MESURE

- **La mesure sert à l'identification d'un problème**
- **Et aussi à la validation de sa résolution :**
  - Nécessite une comparaison par rapport à une baseline
- **Cela implique :**
  - Gel (peu envisageable) ou utilisation d'une branche
  - Enregistrement des conditions de mesure (version, état du matériel, environnement système, configuration, paramètres JVM…)
  - Jeux de données identique (dump), statistiques DB identiques
  - Maîtrise du niveau de LOG, aspects actifs (tissage léger)
  - Gestion de l'historique des mesures ; classification des résultats
  - Disponibilité / exclusivité : pas de pollution des tirs
  - Quels KPI : temps de réponse, nombres de requêtes, taux CPU,
- **consommations mémoire, nombre de sessions/connections...**

TEST DE PERFORMANCE – IDENTIFICATION, ANALYSE, PRIORISATION

- **L'identification des problèmes se fait suivant des SLA réalistes**
- **L'analyse ne doit pas être basée sur des hypothèses ou convictions mais sur des faits : les résultats.**
- **Les problèmes / goulets d'étranglement (bottlenecks) doivent être qualifiés en terme de:**
  - Risques d'occurrences
  - Importance (% temps consommé)
  - % de présence sur les différents scénarios
  - Corrélation avec la popularité du scénario
  - Coût de correction et tests
- **Prioriser en agrégeant ces critères, par exemple:**
  - Favoriser une correction à 2j/H pour 30 % de gain par rapport à une à 15j/H pour 50 % de gain
  - Favoriser une correction à 10j/H sur un goulet au niveau du framework représentant 10 % de gain sur 70 % des scénarios plutôt qu'une à 5j/H pour 80 % de gain sur un scénario mineur

TEST DE PERFORMANCE – JEUX DE DONNÉES

- **Travailler le plus tôt possible avec des données de production. Si les données sont sensibles, s'appuyer dessus en masquant / remplaçant les attributs sensibles**
- **Respecter la répartition réelle de la typologie des données (induit des chemins d'exécution différents ainsi que des plans d'exécutions de requêtes différents)**
- **Variabiliser les types d'utilisateurs injectés suivant différents profils**
- **Variabiliser les objets/ressources accédés afin de ne pas travailler exclusivement avec les caches**

TEST DE PERFORMANCE – CHOIX DES SCÉNARIOS

- **Identifier les cas d'utilisations critiques :**
  - Les plus fréquemment utilisés (s'aider de statistiques si déjà en production)
  - Les plus vitaux (la facturation, la paie...)
  - Les plus risqués (qui peuvent pénaliser le système dans son intégralité, ex. : les recherches)
- **Choix de scénarios réalistes mais ne pas être perfectionniste (coûts inutiles)**
- **Attention au coût de maintien d'un nombre trop élevé de scénarios à faire évoluer en phase de développement**
- **Attention aux scénarios faisant appels à des services extérieurs (plus complexes) : risques de ban ou faux positifs**

TEST DE PERFORMANCE – LES OUTILS

- **HP Load Runner (Commercial)**
- **JMeter (licence Apache)**
  - Fondation Apache
  - Maturité
  - GUI facile à utiliser
- **Gatling (licence Apache)**
  - Scripts Scala, avec un DSL
  - Recorder réaliste, injection évoluée d'utilisateurs
  - Rapports détaillés

TEST DE PERFORMANCE – PROFILING/MONITORING APPLICATIF

- **Permettent de suivre l'activité des threads, consommation de la mémoire de la JVM, activité du GC etc...**
- **Outils gratuits :**
  - JConsole : inclus JDK5+, monitoring JMX uniquement
  - JVisualVM : inclus JDK6+ : monitoring JMX + profiling basique mémoire/thread (présente des limites avec trop de threads)
  - Java Mission Control : inclus JDK7+ monitoring JMX avancé, profiler (« flight recorder ») payant
  - Netbeans profiler : nécessite l'IDE, profiler complet
- **Outils payants :**
  - JProfiler : le plus complet (facilité de détection des deadlocks etc.)
  - Yourkit : plus simple à utiliser et moins cher que JProfiler

TEST DE PERFORMANCE – MONTORING PHYSIQUE

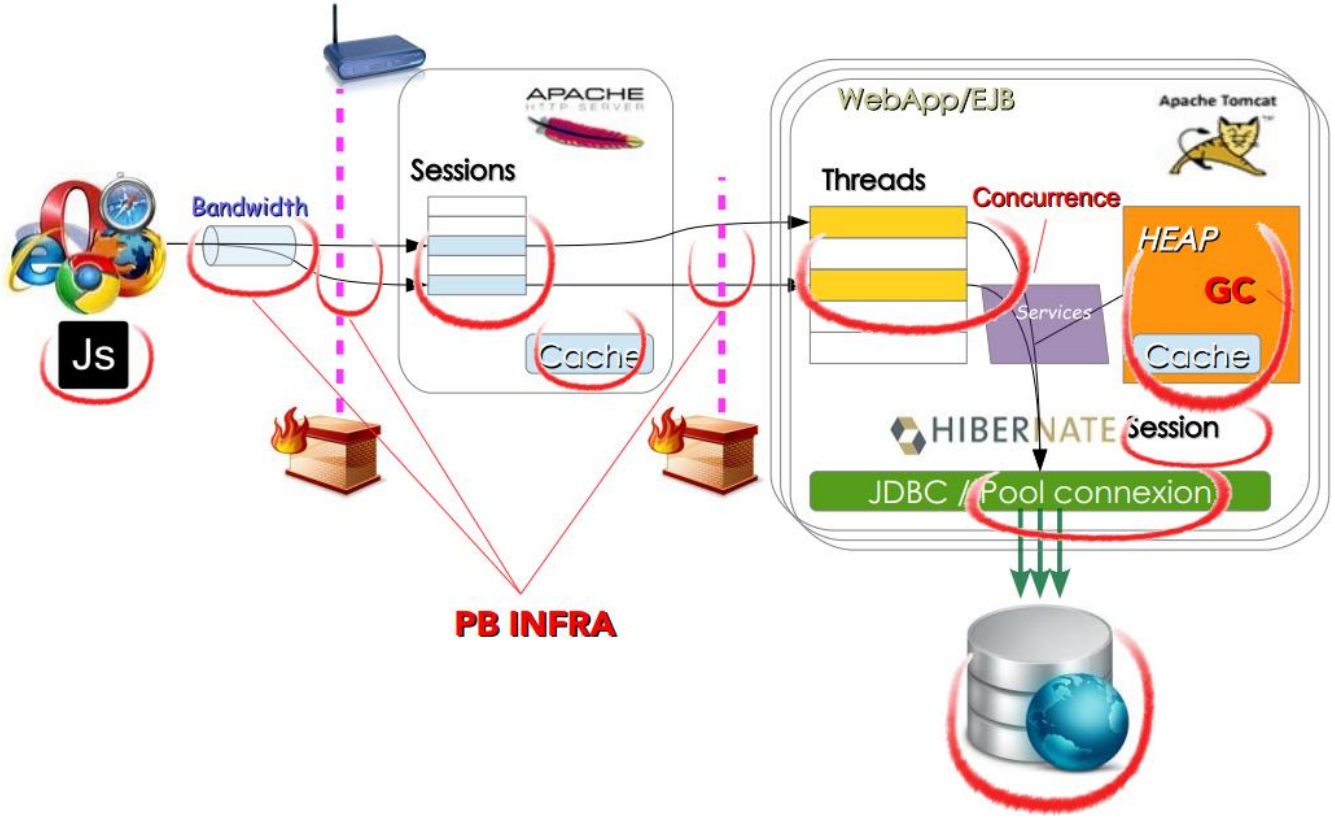
- **Suivre l'utilisation des ressources physiques :**
  - Charge ou % d'utilisation des CPU
  - Mémoire physique
  - Usage disque I/O
  - Nombres de connections
  - Taux d'erreurs
- **Outils :**
  - Perfmon, sysinternals, vmstats, iostat, sysstat, topas, top, nmon
  - TraceRoute, netstat, tcpdump, wireshark...
  - Sous windows il y a le moniteur de performance
  - Et toujours la jconsole ou visual VM
- **Outils de supervision**
  - Metrics + Graphite
  - Hyperic

TEST DE PERFORMANCE – JMX

- **En standard java propose une API permettant le monitoring : JMX (Java Management Extensions)**
  - Activation explicite dans la ligne de commandes en lançant votre application Java
  - Tout les serveurs d'application et/ou container de servlet exposent des beans spécialisés avec des metrics propres

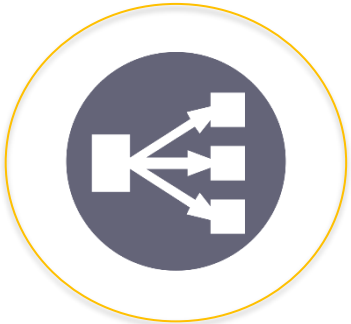
- **Il existe un grand nombre de client JMX, en standard java propose :**
  - JConsole
  - VisualVM

TEST DE PERFORMANCE – LES POINTS D'ATTENTION



APACHE JMeter™

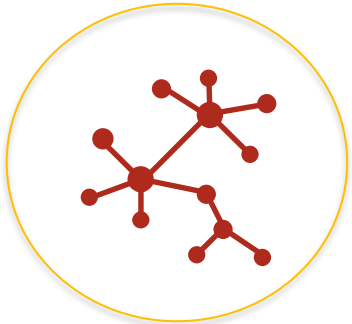
Proxy Server



Genericity

Assertions

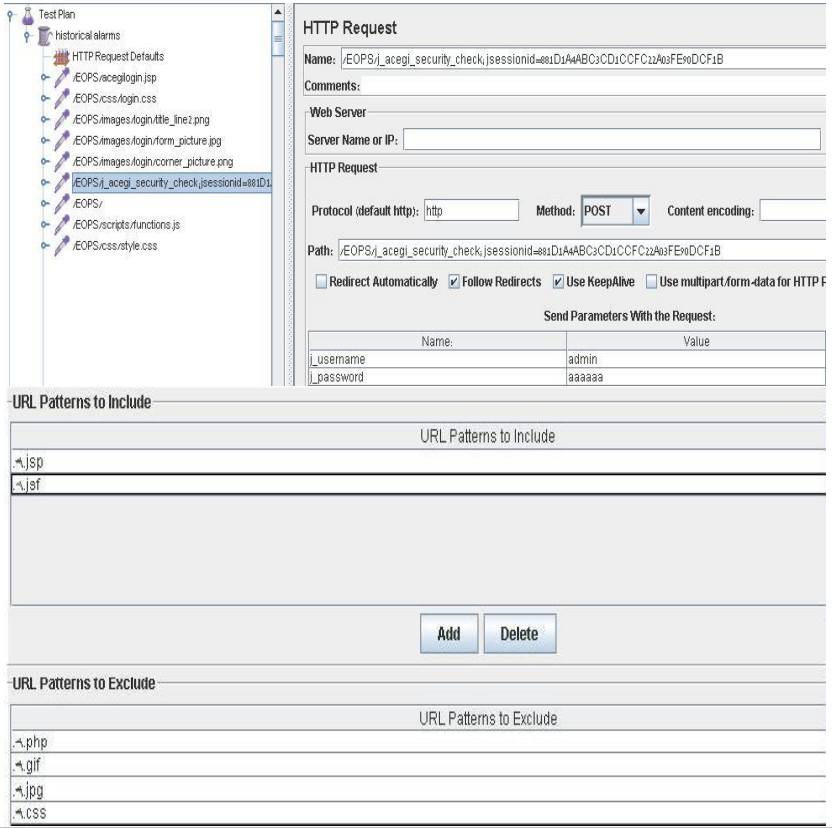
Distributed testing



Analyzing Test

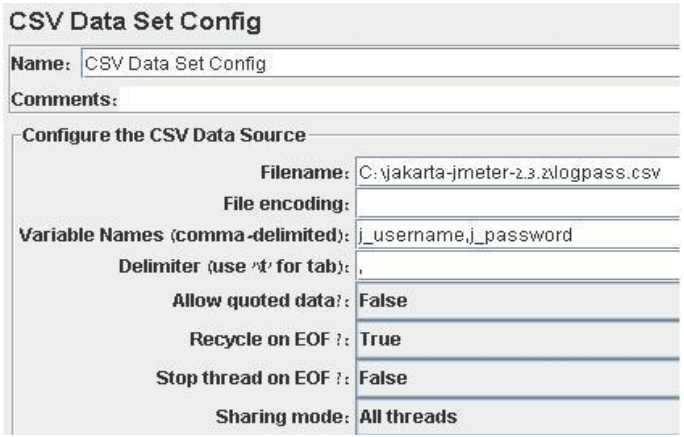
PROXY SERVER

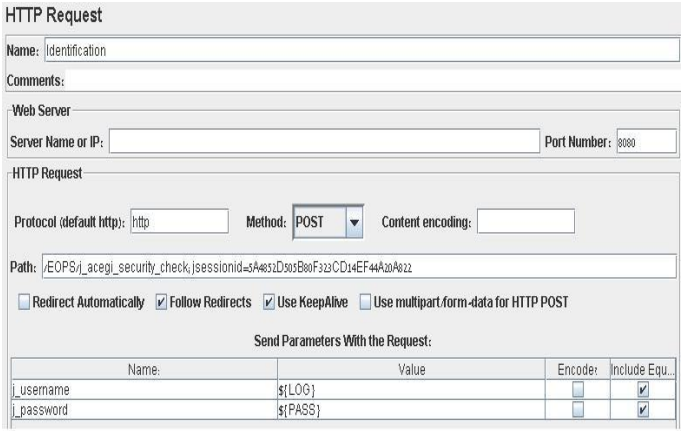
- **Rôles**
  - Enregistre les requêtes lancé par les utilisateurs
  - N'effectue aucune transformation des requêtes
  - Enregistre que les requêtes signifiantes
  - Organise les appels
- **Attention**
  - N'enregistre pas le HTTPS



GENERICITY

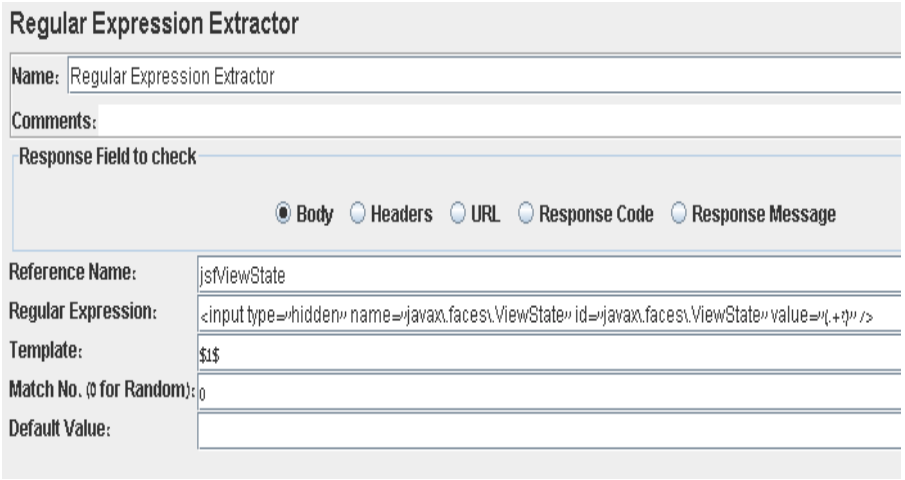
- **Variabilisation :**
  - Afin de ne pas modifier un test pour l'exécuter sur différentes machines
  - Exemple : les utilisateurs et les mots de passe peuvent changer (pour un test complet il faut en utilizer plusieur)
- **Http default Request**
  - Permet de mettre une IP/port et un path par défaut pour toutes les requêtes Http contenues dans le scénario
  - Vous donne un moyen simple d'exécuter votre test d'un appareil à l'autre simplement en changeant l'adresse par défaut.

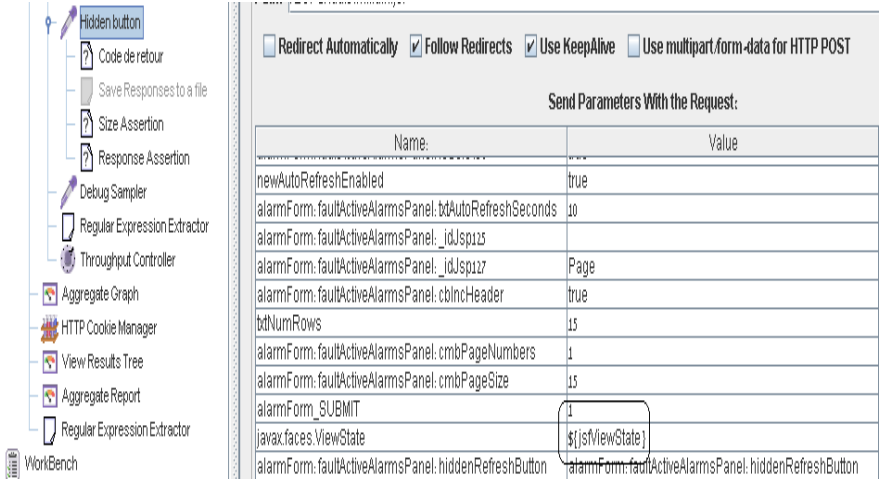




GENERICITY

- **Regular Expression extractor**
  - Si les données doivent être utilisées plusieurs fois le long du test
    - Comme un sessionId par exemple.





ASSERTIONS

- **Response assertion**
  - Pour faire correspondre un pattern dans le code de la réponse
    - Le response code par exemple
- **Xpath assertion**
  - Utilisation du DOM de la réponse pour vérifier si un élément apparaît
    - Un résultat de recherche, par exemple
- **Size assertion**
  - Pour savoir si la taille de la réponse reçue correspond à la taille attendue
    - Pour vérifier si le fichier reçu est le bon

DISTRIBUTED TESTING

- **Pourquoi ?**
  - Pour simuler l'environnement stressé avec beaucoup de clients
- **Comment ?**
  - Editer le “remote_hosts=127.0.0.1” dans jmeter.properties
  - démarrer jmeter_server.bat sur la machine host
  - executer jmeter.bat

ANALYZING TEST

Aggregated graph

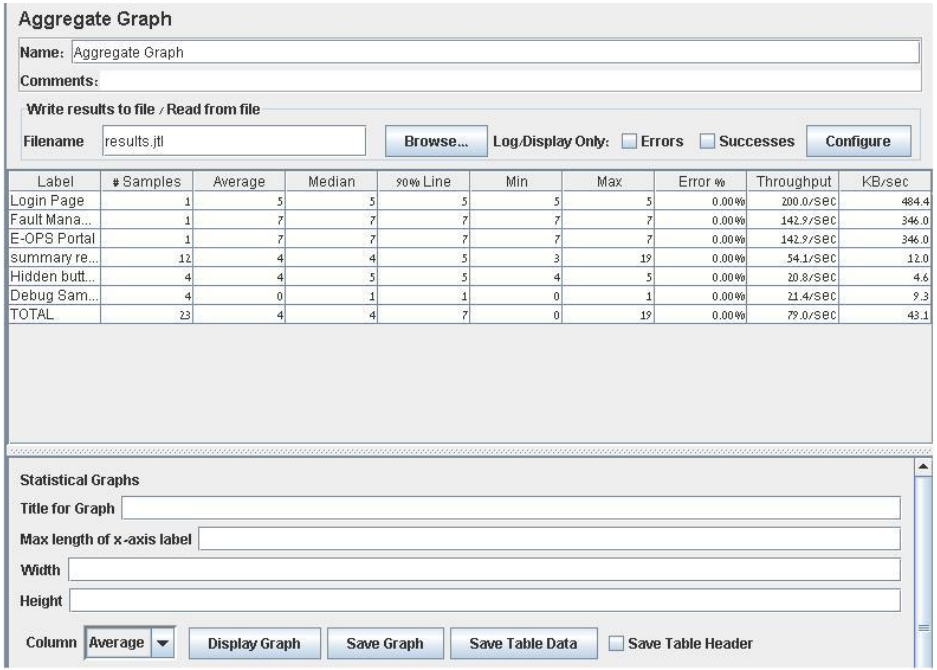
- Gives all the statistics concerning the tests
- May be recorded in a specified file for further treatment (data mining)

Result tree

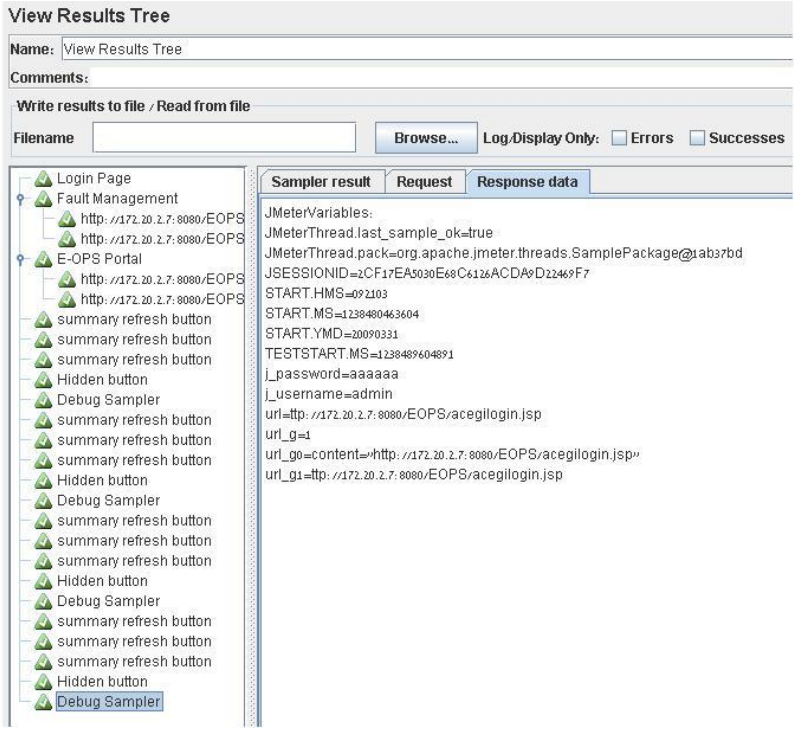
- Gives in a tree form, all the samplers results, the requests, and the sampler data.
- May also be recorded in a specified file for further treatment

ANALYZING TEST

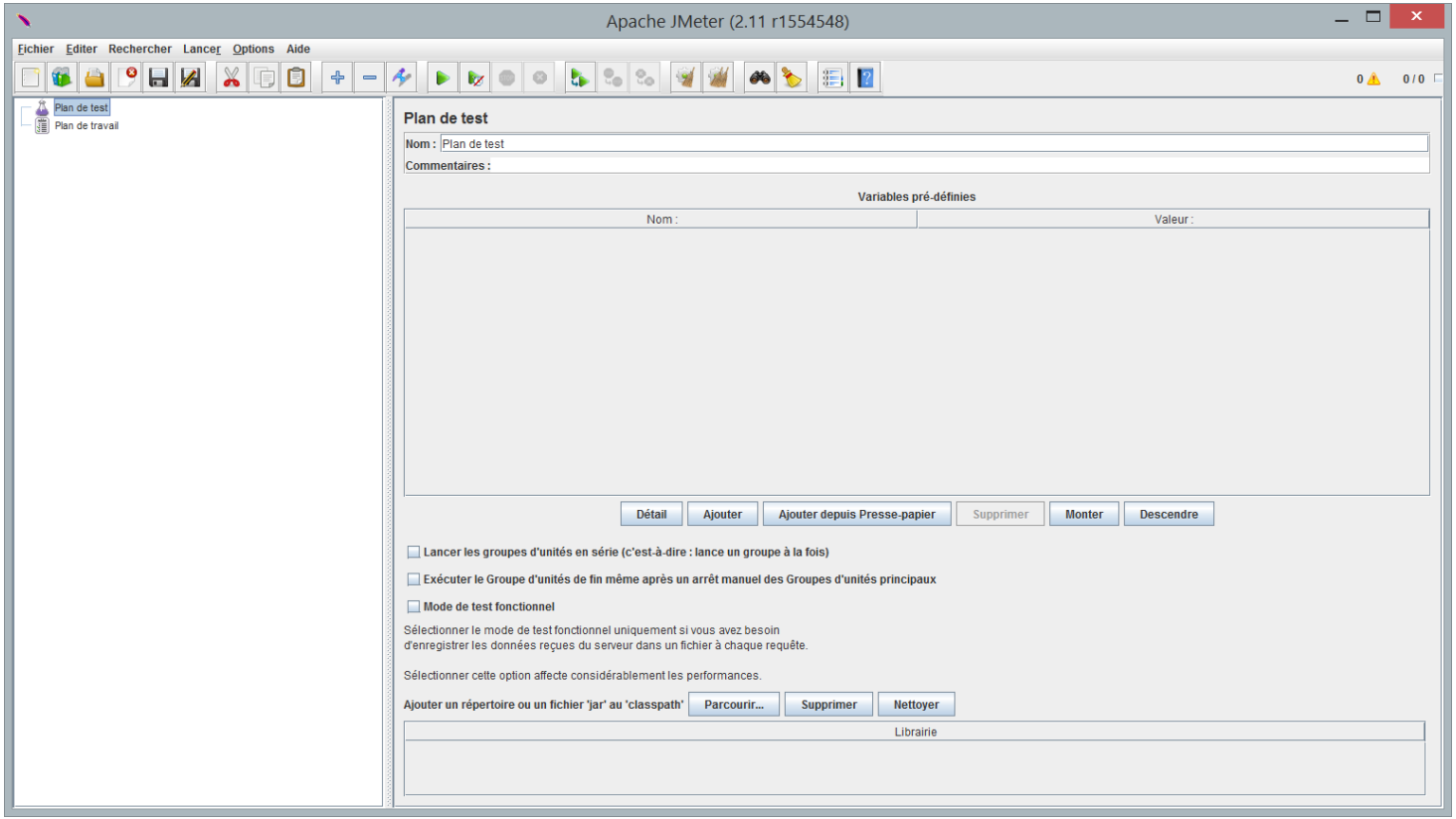
Aggregated graph



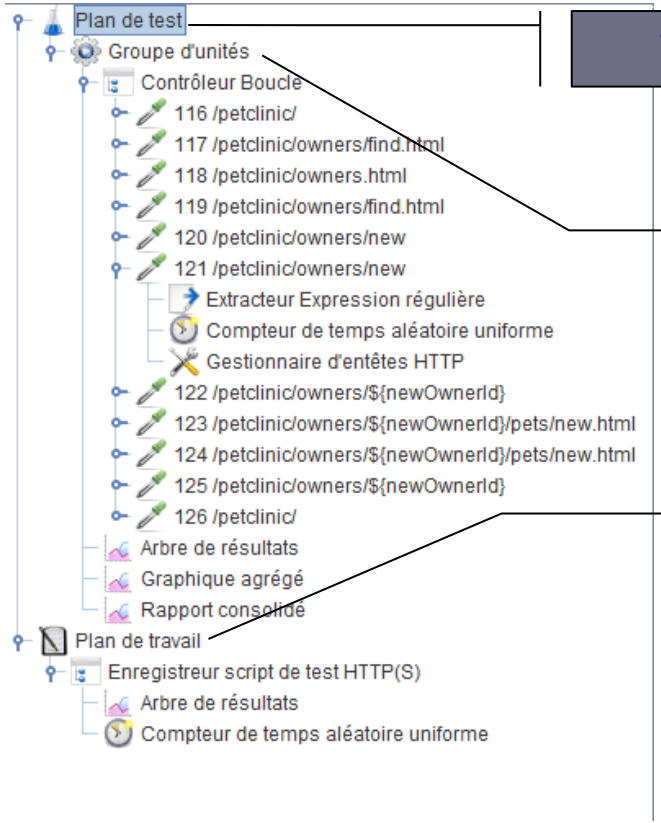
Result tree



TEST PERFORMANCE – JMETER



L'ORGANISATION DU PLAN DE TRAVAIL



Test Plan

- **Eléments exécutés lorsque l'on lance le plan de test Attention c'est la seule partie qui est sauvée**

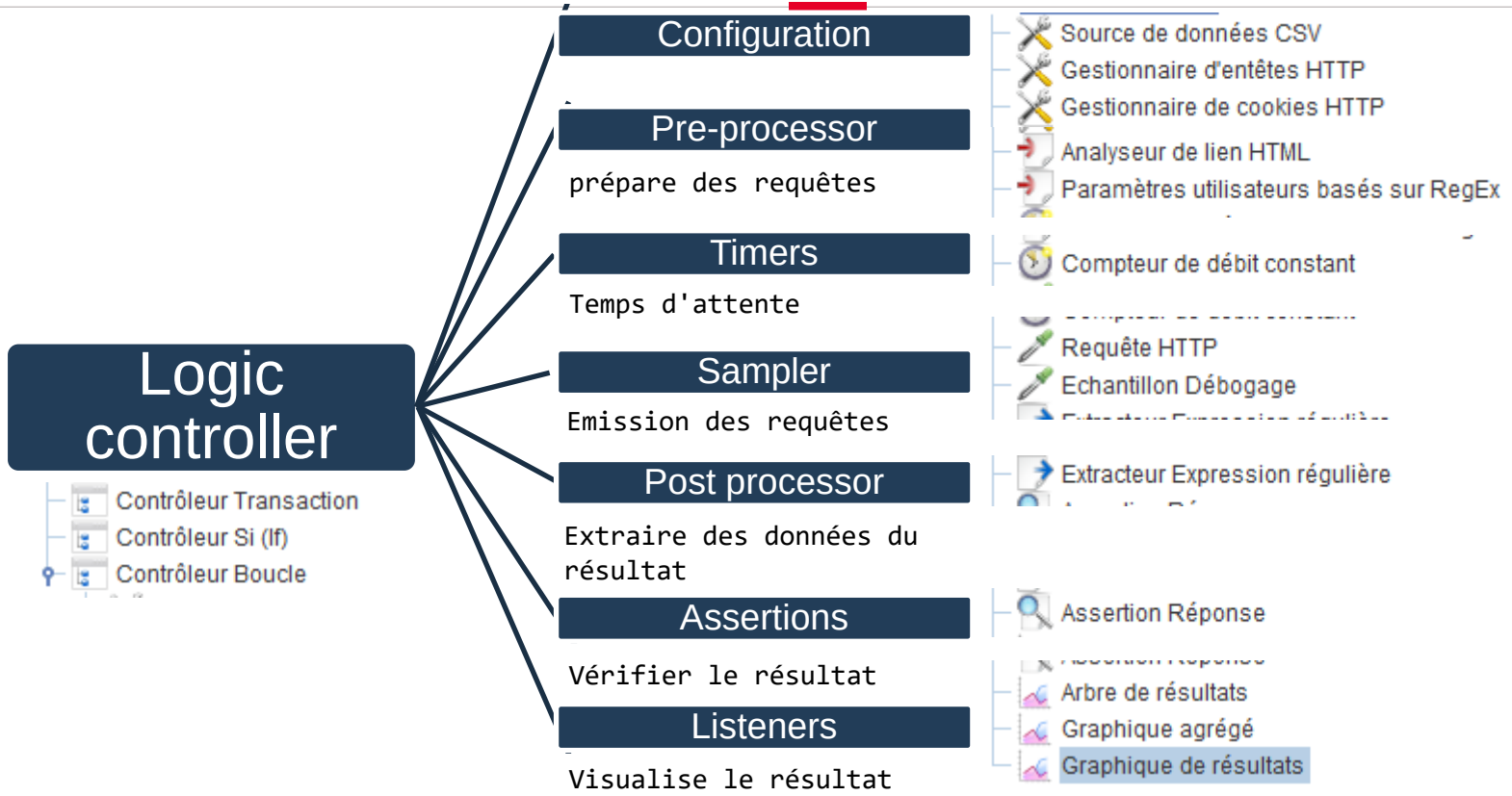
Groupe d'unités

- **Nombre d'utilisateurs et d'itérations, ramp up**

Workbench

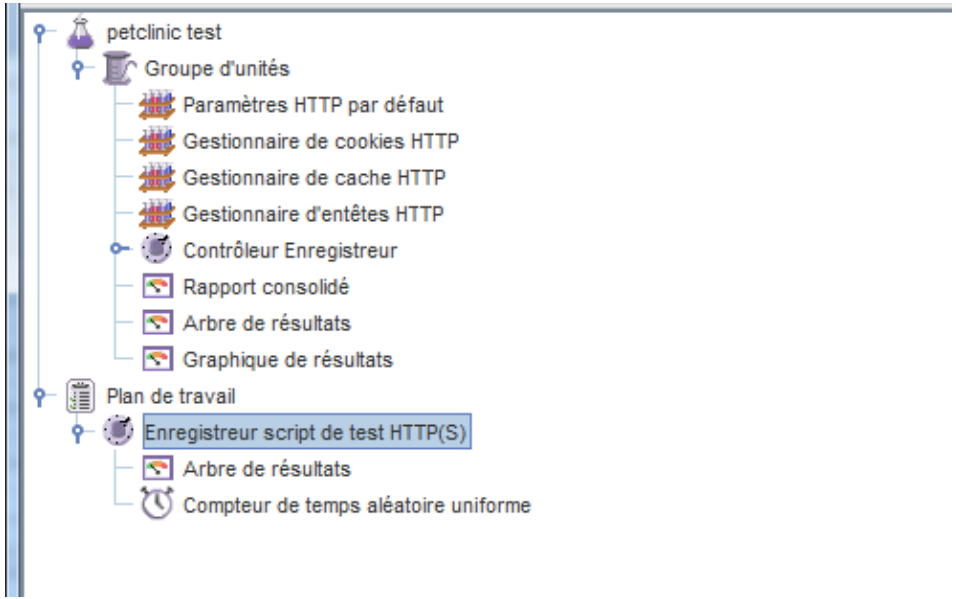
- **Zone de dépôt des éléments utilisés temporairement (le proxy HTTP d'enregistrement par exemple)**

LES CONCEPT


Logic controller
Contrôleur Transaction
Contrôleur Si (If)
Contrôleur Boucle
Configuration
Source de données CSV
Gestionnaire d'entêtes HTTP
Gestionnaire de cookies HTTP
Pre-processor
prépare des requêtes
Analyseur de lien HTML
Paramètres utilisateurs basés sur RegEx
Timers
Temps d'attente
Compteur de débit constant
Sampler
Emission des requêtes
Requête HTTP
Echantillon Débogage
Post processor
Extraire des données du résultat
Extracteur Expression régulière
Assertions
Vérifier le résultat
Assertion Réponse
Listeners
Visualise le résultat
Arbre de résultats
Graphique agrégé
Graphique de résultats

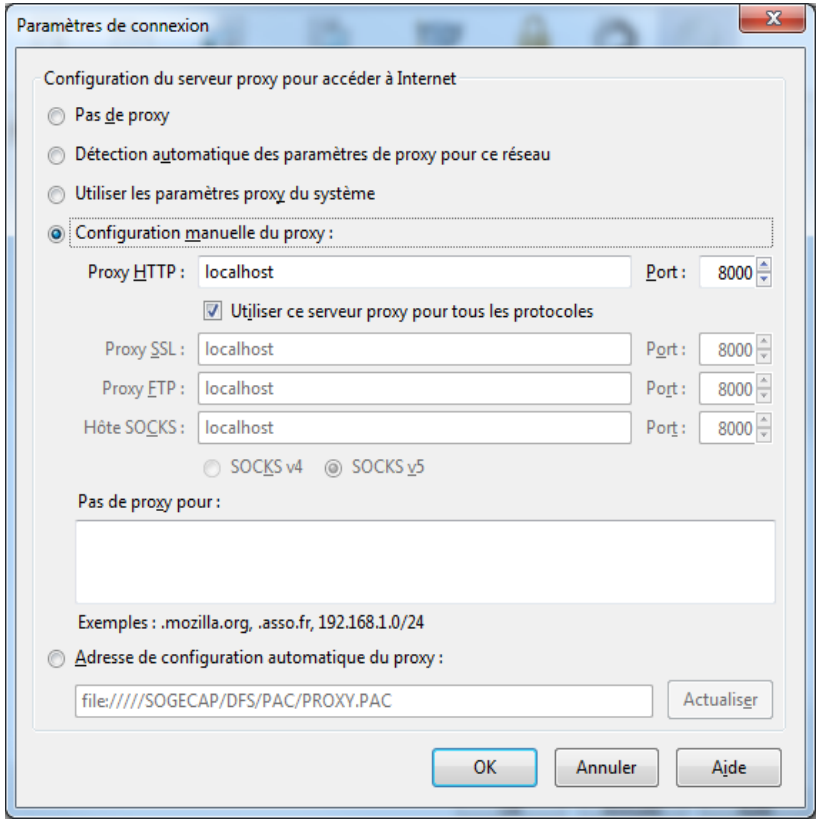
Configuration du plan de travail et de test :

- Gestionnaire de cookie http
  Ceci nous servira à nettoyer les cookies à chaque itération afin de ne pas réutiliser le même panier (penser à cocher la case)
- Enregistreur script de test http(s)
  - Définir le port du proxy à 8000
  - Cocher la capture des entêtes, ajout d'assertion (facultatif), suivre les redirections
  - Ajouter le motif a exclure
    - .*\.gif
    - .*\.js
    - .*\.png
    - .*\.css



TEST DE PERFORMANCE – JMETER

- **Lancez sur l'enregistreur et cliquez sur le bouton « Lancer » ; un proxy va être créé, il faudra donc activer l'utilisation d'un proxy au niveau du navigateur**
- **Naviguez dans l'application**

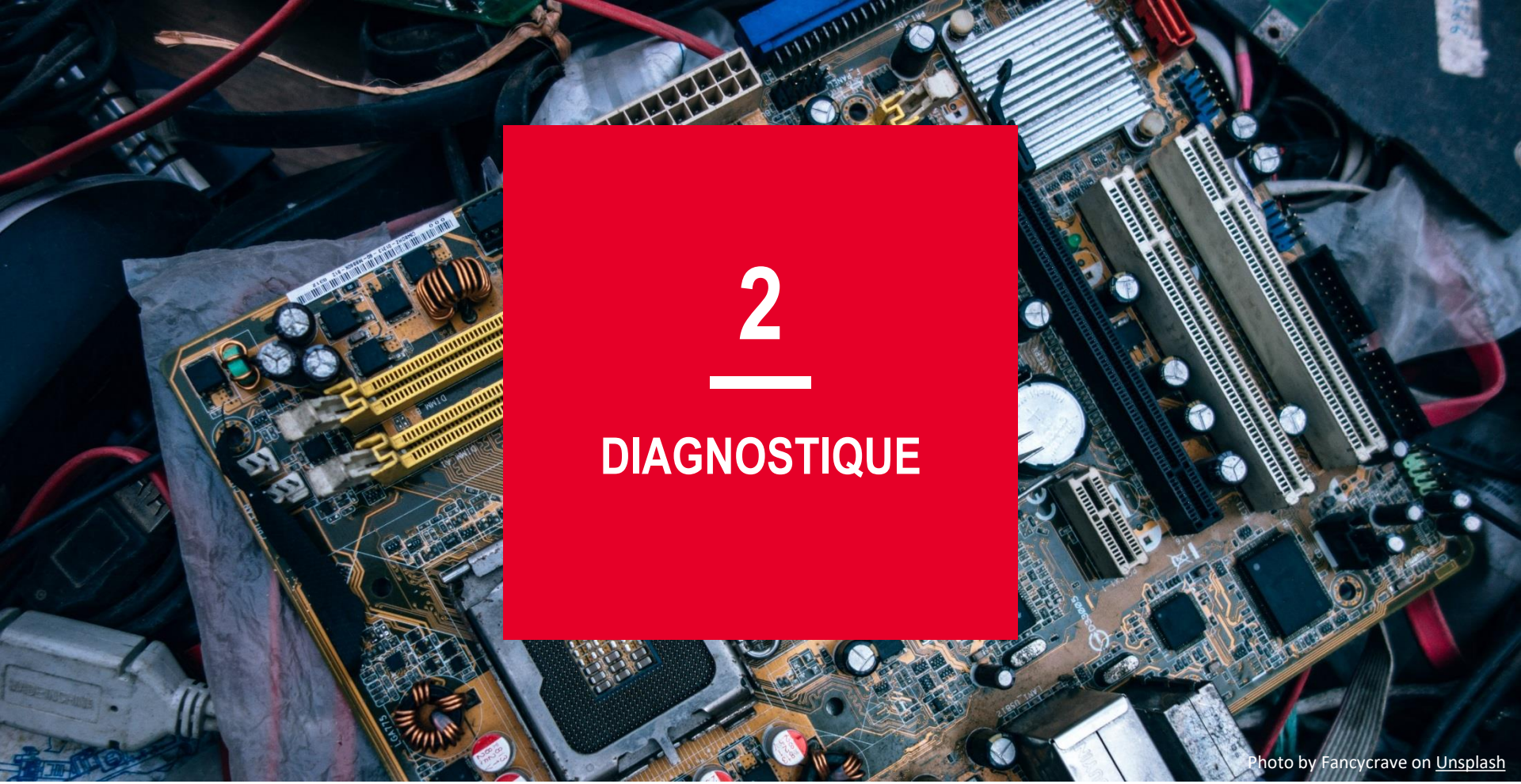


TEST DE PERFORMANCE – JMETER

- **Dans l'arbre de résultat on voit l'ensemble des requêtes effectuées et leurs détails**
- **Stoppez l'enregistreur et observez le contrôleur enregistreur, il contient automatiquement le jeu effectué (attention à bien nettoyer les actions intermédiaire parasites si vous êtes allé sur d'autres sites)**
- **Configurez les assertions générées sous les requêtes HTTP, cela permettra de s'assurer que les tirs de test ont bien le retour attendu et ainsi détecter un retour vide si le serveur est stoppé ou si le code a changé. Exemple d'assertion : motif « contient », motif à tester « grossPrice":80 »**

TEST DE PERFORMANCE – JMETER

- **Ajoutez des récepteurs de type « Rapport consolidé » ou « Arbre de résultat » afin de suivre la bonne exécution de nos tests.**
- **Attention lors de la mesure, il est conseillé de désactiver les assertions ainsi que tous les récepteurs ou autres éléments qui ne servent qu'à l'interprétation des résultats ; ainsi la mesure ne sera pas parasitée et donc plus précise.**
- **Afin de lancer les tests il suffira d'appuyer sur le bouton « lecture » et tous les éléments non désactivés s'exécuteront.**
- **Si l'on veut modifier le nombre d'utilisateurs, la durée de montée en charge ou le nombre d'itérations (l'option infini peut être intéressante pour la détection de fuite mémoire), on peut l'effectuer sur le groupe d'unités.**
- **Maintenant on pourra suivre l'exécution du tir sur VisualVM**



2

DIAGNOSTIQUE

Photo by Fancycrave on Unsplash

DIAGNOSTIC – LES OUTILS

- **Permettent de suivre l'activité des threads, consommation de la mémoire de la JVM, activité du GC etc...**
- **Outils gratuits :**
  - JConsole : inclus Oracle JDK5+, monitoring JMX uniquement
  - JVisualVM : inclus Oracle JDK6+ : monitoring JMX + profiling basique mémoire/thread (présente des limites avec trop de threads)
  - Java Mission Control : inclus Oracle JDK7+ monitoring JMX avancé, profiler (« flight recorder ») payant
  - Netbeans profiler : nécessite l'IDE, profiler complet
- **Outils payants :**
  - JProfiler : le plus complet (facilité de détection des deadlocks etc.)
  - Yourkit : plus simple à utiliser et moins cher que JProfiler

DIAGNOSTIC – LA MÉMOIRE

Les exceptions possibles en java

- **La pile (heap) : mémoire ou son stocké les objets java. (La valeur initial est fixé par –Xms et –Xmx pour sa valeur max). Lorsqu'il n'y a pas plus de place :**

```
java.lang.OutOfMemoryError: Java heap space.
```

- **Le permgem space est l'espace alloué pour le chargement des classes. Taille de mémoire fixé au démarrage par -XX:PermSize. Si il n'y plus d'espace pour ajouter des class :**

```
java.lang.OutOfMemoryError: PermGen space
```

- **La dernière type d'erreur, les erreurs natives. Plus rare car elles apparaissent uniquement lorsqu'il n'y a plus de mémoire vive.**

```
java.lang.OutOfMemoryError: request <size> bytes for <reason>.
```

DIAGNOSTIC – LA MÉMOIRE

- young generation : la plupart des objets sont instanciés dans cette génération
- old(tenured) generation : contient les objets qui ont survécu à plusieurs collections de la young generation
- permanent generation : contient des objets nécessaires au fonctionnement de la JVM

DIAGNOSTIC – LA MÉMOIRE

- **Les champs statiques ne peuvent pas être garbage collectés car leur durée de vie est permanente pour un classloader donnée**
- **Chaque instance d'objet à une taille mémoire donnée (**shallow heap**)**
  - Constitué de primitif (int, char)
  - Référence vers d'autres objets. (Le terme employé pour la taille de l'objet et de toutes ces dépendance : retained Heap)

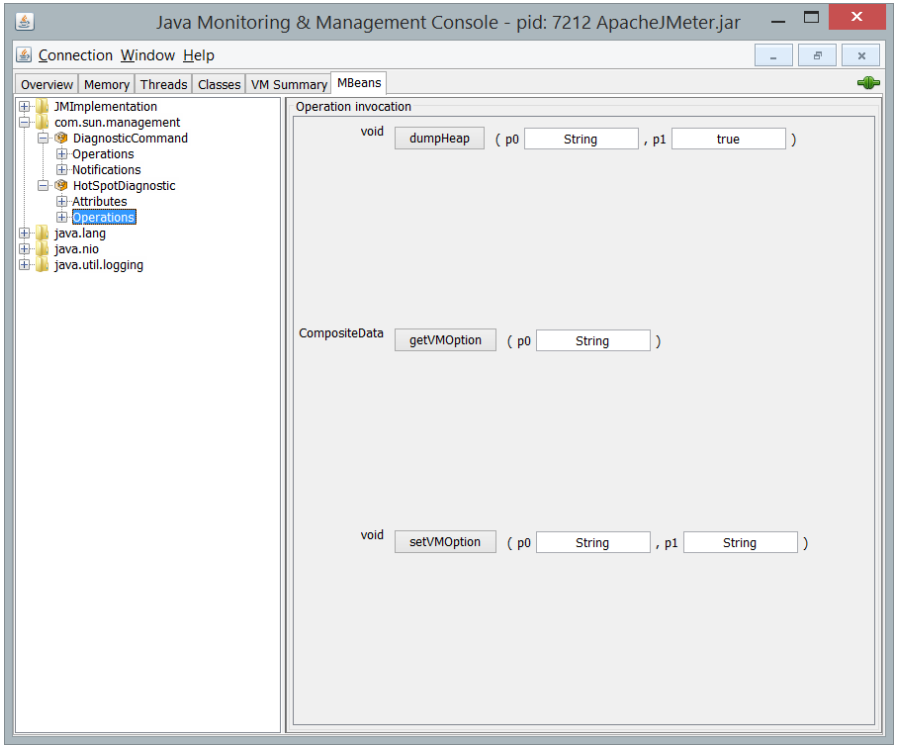
⇒ **Les objets ainsi liés entre eux constituent un graphe.**

- **La racine d'attache (**"GC root"**) est le point de départ des graphes**
  - Thread, classloader

- **Une fuite de mémoire est une dérive de l'utilisation de la mémoire par un programme. Ces fuites peuvent être dû à des références vers des objets non libérés empêchant le GC de faire son travaille.**

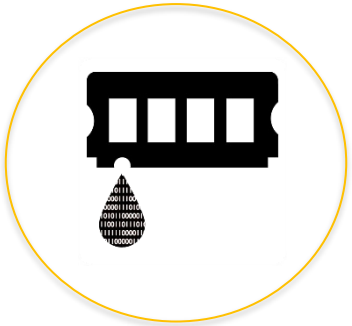
DIAGNOSTIC – HEAPDUMP

- **Sélectionnez le processus Java pour lequel vous voulez faire le dump.**
- **Onglet "VM summary" – Mbeans – Item com.sun.management / hotspot diagnostic / operation / dumpheap**
- **L'opération dumpHeap fige la JVM, donc a ne pas faire en production**



eclipse MAT : Memory Analyser Tool

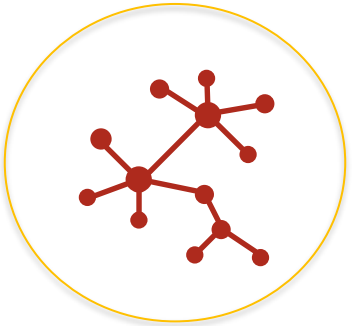
Leak detection



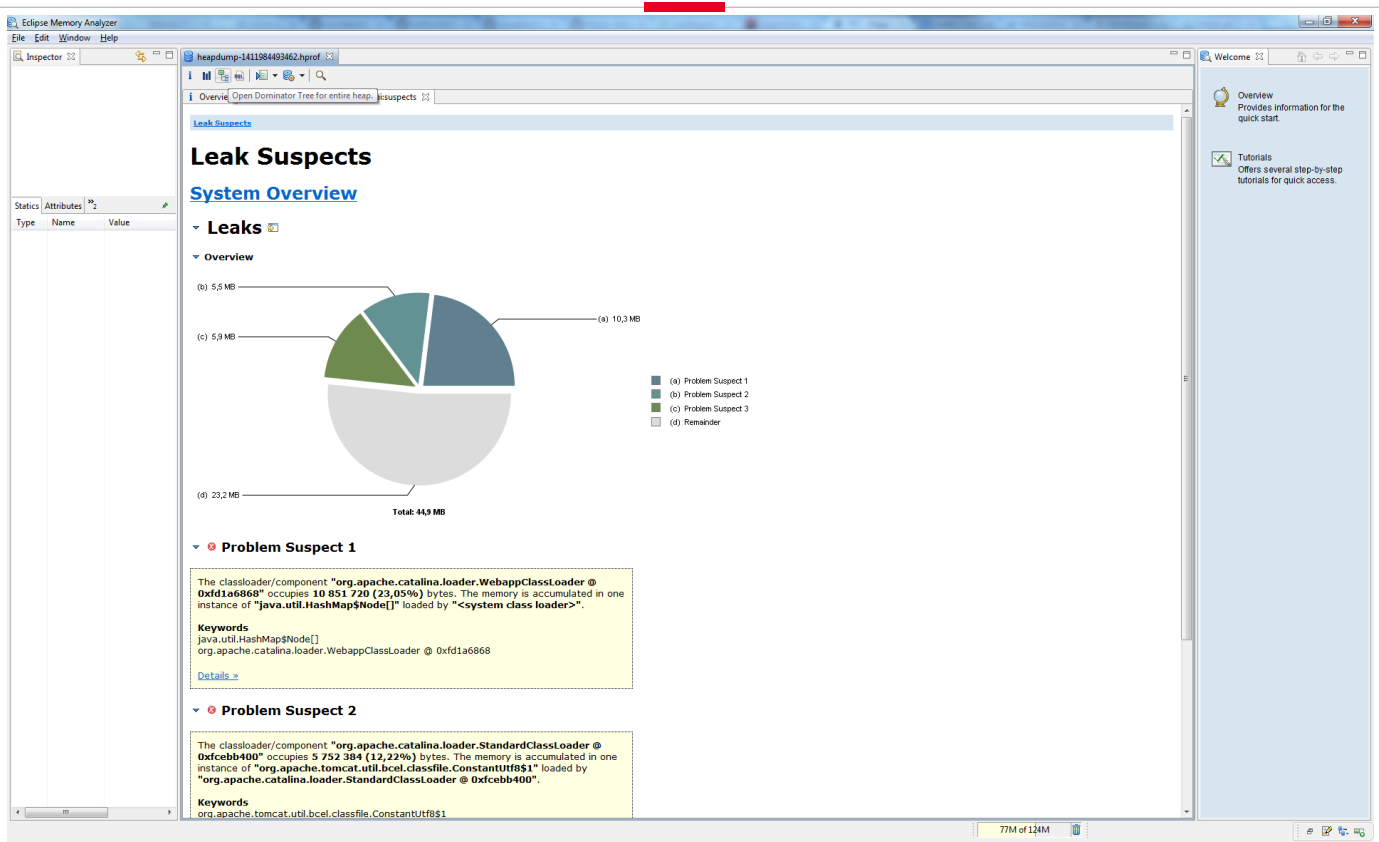
Retained Heap

Dominator Tree

GC Roots



Inspector


Eclipse Memory Analyzer
File Edit Window Help
Inspector
heapdump-1411984493462.hprof
Open Dominator Tree for entire heap
Leak Suspects
Leak Suspects
System Overview
Leaks
Overview
(b) 5,5 MB
(a) 10,3 MB
(c) 5,9 MB
(d) 23,2 MB
Total: 44,9 MB
(a) Problem Suspect 1
(b) Problem Suspect 2
(c) Problem Suspect 3
(d) Remainder
Problem Suspect 1
The classloader/component "org.apache.catalina.loader.WebappClassLoader @ 0xfd1a6868" occupies 10 851 720 (23,05%) bytes. The memory is accumulated in one instance of "java.util.HashMap$Node[]" loaded by "<system class loader>".
Keywords
java.util.HashMap$Node[]
org.apache.catalina.loader.WebappClassLoader @ 0xfd1a6868
Details »
Problem Suspect 2
The classloader/component "org.apache.catalina.loader.StandardClassLoader @ 0xfcebb400" occupies 5 752 384 (12,22%) bytes. The memory is accumulated in one instance of "org.apache.tomcat.util.bcel.classfile.ConstantUtf8$1" loaded by "org.apache.catalina.loader.StandardClassLoader @ 0xfcebb400".
Keywords
org.apache.tomcat.util.bcel.classfile.ConstantUtf8$1
Welcome
Overview
Provides information for the quick start.
Tutorials
Offers several step-by-step tutorials for quick access.
77M of 124M

DIAGNOSTIC – MAT

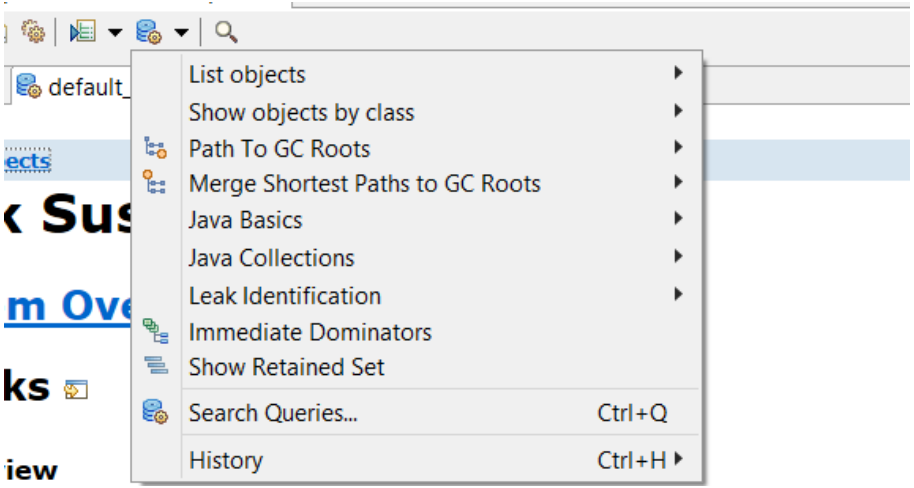
- **MAT : Memory Analyser Tool**

- **Outil de lecture et de diagnostic de la mémoire java**
  - Identification des fuite de mémoire (memory leak)
  - Consommation excessive de mémoire

- **Pour analyser une fuite de mémoire il faut :**
  - Rechercher les plus gros objets (Les outils sont fait pour ça)
  - Analyser si l'objet est pertinent et est susceptible d'être issue d'une fuite de mémoire
  - Pour les objets pertinents, analyser pourquoi ils reste en mémoire ou pourquoi ils sont gros

DIAGNOSTIC - MAT

- **Pour une bonne analyse il faut**
  - Un bon heapDump, les tests de performance sont prévu à cette effet
  - Lorsque c'est l'application en production qui ralenti ou plante par manque de mémoire il faut absolument récupérer un dump
- **L'utilisation du dominator tree permet de trouver les objets les plus consommateur**
- **L'open query browser permet d'avoir des vues différentes des objets**



TP

Photo by Annie Spratt on Unsplash