**Tomcat Apache Serveur**

****

**Projet : SMB214**

**Préparé par: Chantale Al Hajj**

**Quelle est la différence entre un serveur web et un serveur d’application ?**

* Couche 1 : Serveur web Http
* Couche 2 : Conteneur web
* Couche 3 : Serveur d’application

**Couche 1 : Serveur web http**

Pour commencer, il nous faut tout d’abord : Un **serveur http**, c’est un serveur qui gère exclusivement des requêtes HTTP. Il a pour rôle d’intercepter les requêtes Http, sur un port qui est par défaut 80, pour les traiter  et générer ensuite des réponses Http.

Exemple de serveurs web : Apache, Nginx, Lighttpd, IIS…Les fonctionnalités d’un serveur WEB :

A part sa fonction basique qui est d’intercepter et de répondre en Http, un serveur web   peut avoir d’autres fonctionnalités tel que

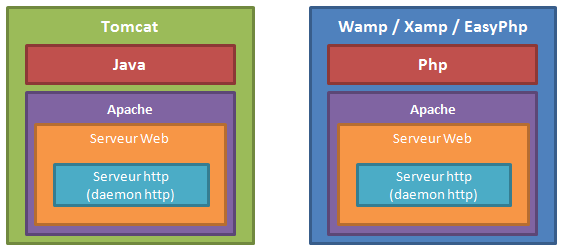
* La gestion de la sécurité, comme les fonctionnalités de restriction des accès par domaine, par utilisateur, par groupe ou par adresse IP,
* La gestion du contenu, comme la redirection des requêtes http, la personnalisation des messages d’erreurs, ou la gestion des timeout...

Dans le serveur apache par exemple ces fonctionnalités sont implémentées sous forme de modules (mod\_alias, mod\_authn\_core, mod\_proxy…).

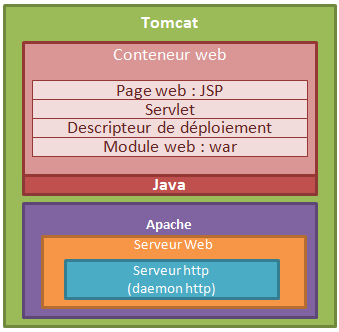
**Couche 2 : Conteneur web**

Ensuite, on va étendre notre serveur web pour devenir un **conteneur Web.** Cette extension va permettre d’avoir la possibilité d’exécuter des programmes écrits avec des langages de programmation (java, php, C# ou autres) dans le serveur web.

Par exemple le serveur Tomcat n’est autre qu’un serveur Apache couplé avec un moteur web java et, les serveurs tel-qu’EasyPhp, wamp ou Xamp ne sont que des serveurs Apaches couplés avec un moteur web php.

[](https://medimagh.files.wordpress.com/2015/05/conteneur_web.png)

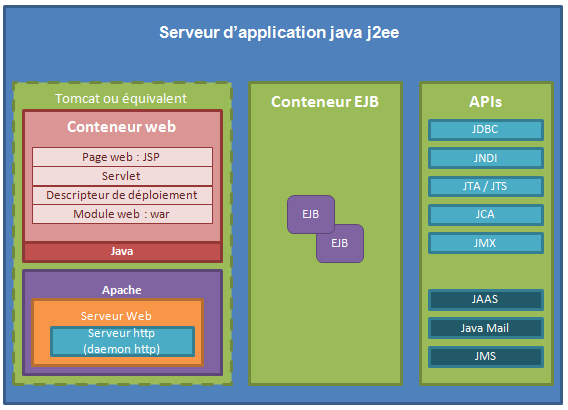
Maintenant, on va se concentrer sur le serveur **Tomcat qui est un conteneur web java** (et pas un serveur d’application jee), pour analyser son architecture.

[](https://medimagh.files.wordpress.com/2015/05/architecture_conteneur_web1.png)

Le conteneur web Tomcat, est composé d’un moteur jsp, un moteur servlet et d’un descripteur de déploiement pour les modules web de type war. Ces moteurs sont en réalité des API qui sont implémentés dans le serveur Tomcat, et qui permettent de faire déployer seulement des applications web Java de type war.  
Les applications java de type EAR, ne peuvent pas être déployées dans Tomcat, parce que tout simplement le serveur manque les API nécessaires et conformes aux spécifications  pour l’implémentation des serveurs d’application java jee.  
Par exemple, si vous utiliser la bibliothèque JPA dans votre application web et vous le déployez sur un serveur Tomcat, vous serez obligé d’embarquer les jars JPA dans le répertoire lib de l’application, alors que si vous déployez sur un serveur d’application comme JBOSS, vous n’aurez besoin d’aucun jar additionnel.

**Couche 3 : Serveur d’application**

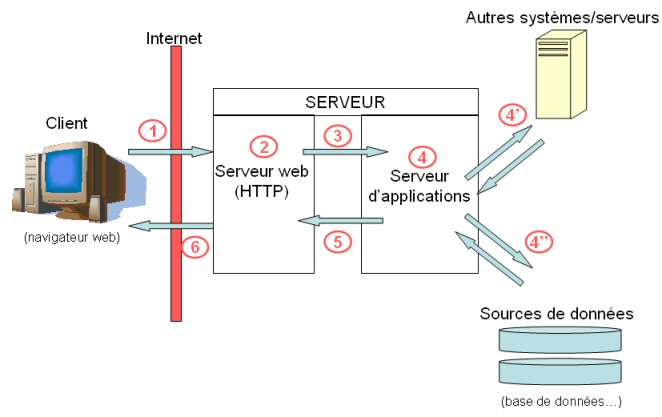
* Donc, vous l’avez bien compris, il faut étendre encore plus le serveur Tomcat pour devenir un vrai serveur d’application java jee.
* L’extension nécessaire est composée de deux parties essentielles :
* **Un conteneur EJB** qui encapsule les traitements des Entreprise JavaBeans.
* Un ensemble de **services** répartie en :
  + Des services d’infrastructures
    - JDBC (Java DataBase Connectivity) API d’accès aux bases de données relationnelles.
    - JNDI (Java Naming and Directory Interface) API d’accès aux services de nommage et aux annuaires d’entreprises.
    - JTA/JTS (Java Transaction API/Java Transaction Services) API pour la gestion de transactions.
    - JCA (JEE Connector Architecture) API de connexion au système d’information de l’entreprise comme les ERP.
    - JMX (Java Management Extension)  API permettant de développer des applications web de supervision d’applications
  + Des services de communication:
    - JAAS (Java Authentication and Authorization Service) API de gestion de l’authentification.
    - JavaMail API pour la gestion de courrier électronique.
    - JMS (Java Message Service) API de communication asynchrone entre application.

[](https://medimagh.files.wordpress.com/2015/05/architecture_serveur_application_java_j2ee.png)

**Tomcat un serveur d'applications:**

Tomcat est un serveur d'applications Java. Nous avons déjà présenté ce qu'est une application web. Elle permet de générer une réponse HTML à une requête après avoir effectué un certain nombre d'opérations (connexion à une base de données, à un annuaire LDAP...). Pour le client (un navigateur web en général), il n'y a pas de différence avec une page web statique : il reçoit toujours du HTML, seul langage qu'il comprend. Seule la manière dont la réponse est formée côté serveur change.

Les requêtes, pour le client, ne diffèrent pas non plus. Qu'il souhaite accéder à une ressource statique ou à une application web, il utilise toujours une URL au même format (standard HTTP). C'est donc côté serveur que la distinction doit s'opérer. Le schéma suivant montre le déroulement classique d'une requête vers un serveur d'applications :



1) Le client émet une requête (i.e. appelle une URL) pour demander une ressource au serveur. Exemple : **http://leserveur.com/welcome**. Il ne sait pas ici si la réponse qui va lui parvenir est statique (page HTML simple) ou dynamique (générée par une application web). Dans notre cas, il s'agit d'une application répondant à l'adresse **welcome** sur le serveur **leserveur.com**.

2) Côté serveur, c'est le serveur web (exemple : Apache) qui traite les requêtes HTTP entrantes. Il traite donc toutes les requêtes, qu'elles demandent une ressource statique ou dynamique. Seulement, un serveur HTTP ne sait répondre qu'aux requêtes visant des ressources statiques. Il ne peut que renvoyer des pages HTML, des images,... existantes.

3) Ainsi, si le serveur HTTP s'aperçoit que la requête reçue est destinée au serveur d'applications, il la lui transmet. Les deux serveurs sont reliés par un canal, nommé connecteur.

4) Le serveur d'applications (exemple : Tomcat !) reçoit la requête à son tour. Il est, lui, en mesure de la traiter. Il exécute donc le morceau d'application (la servlet) auquel est destinée la requête, en fonction de l'URL. Cette opération est effectuée à partir de la configuration du serveur. La servlet est donc invoquée, et le serveur lui fournit notamment deux objets Java (Tomcat est un serveur d'applications Java) exploitables : un représentant la requête, l'autre représentant la réponse. La servlet peut maintenant travailler, et générer la réponse à la demande. Cela peut passer par la consultation de sources de données, comme des bases de données (4'' sur le schéma). Ou bien par l'interrogation d'autres serveurs ou systèmes (4' sur le schéma), l'environnement Java web permettant de se connecter à de nombreux systèmes.

5) Une fois sa réponse générée, le serveur d'applications la renvoie, par le connecteur, au serveur web. Celui-ci la récupère comme s'il était lui-même allé chercher une ressource statique. Il a simplement délégué la récupération de la réponse, et celle-ci a été générée, mais ce n'est plus le problème.

6) La réponse est dorénavant du simple code HTML, compréhensible par un navigateur. Le serveur HTTP peut donc retourner la réponse au client.

**Serveur web / serveur d'applications :**

Dans le schéma en haut de la page, nous avons séparé le serveur web et le serveur d'applications. Ces deux composants sont en effet nécessaires côté serveur, puisqu'ils se complètent : le serveur d'applications ne sait pas traiter une requête HTTP, le serveur web ne sait pas exécuter d'applications !

Si ces deux composantes sont indispensables, elles ne sont pas nécessairement séparées. Tomcat inclut ainsi un serveur web, et est donc capable de fonctionner en autonomie (*StandAlone*), pour traiter à la fois les requêtes HTTP simples (ressources statiques) et les applications web. Le principe est de changer de connecteur (par rapport à notre schéma en haut de la page), pour en utiliser un comprenant les requêtes HTTP et non plus les requêtes triées venant du serveur web.

Dans certains cas, cette possibilité est extrêmement intéressante. Elle permet de proposer un serveur web complet en installant le minimum de logiciels. Pour des besoins de production plus importants, il est intéressant de scinder les deux activités, ne serait-ce que pour alléger la tâche de chacun des serveurs. Tomcat peut ainsi se concentrer uniquement sur l'exécution des applications. Cela économise également sa mémoire et augmente ses performances, en lui permettant notamment de créer moins d'objets... En outre, dans certains cas, comme la mise en place d'un serveur sécurisé (SSL), on préférera gérer la sécurisation sur le serveur web, et pas sous Tomcat, même si celui-ci est capable de mettre en place un environnement sécurisé.

**Tomcat : un serveur d'applications Java :**

Comme nous l'avons déjà évoqué, Tomcat est un serveur d'applications Java. Cela signifie deux choses : d'abord, il est intégralement écrit en Java. Ensuite, les applications qu'il est capable d'exécuter (nommées applications web) doivent être développées en Java. Le rôle du serveur d'applications, nous l'avons compris, est double. Il doit savoir exécuter des applications web pour répondre aux requêtes entrantes. Cela passe par des procédures de chargement de classes (nous sommes en Java), d'invocation dynamique... Il doit également être capable de convertir une requête en objet Java, pour qu'elle soit exploitable par l'application. Et, en retour, savoir convertir l'objet Java contenant la réponse générée, en réponse compréhensible par le serveur web. Ces objets Java sont régis par une API, qui répond aux spécifications Servlet officielles. On comprend dès lors qu'un serveur d'applications est subordonné à un langage. Hormis le concept, le serveur d'applis PHP n'a rien à voir avec un serveur d'applis Java !

Le cœur d'un serveur d'applications Java est le conteneur de servlets, puisque les servlets sont les éléments essentiels d'une application web écrite en Java (elles reçoivent les requêtes et renvoient les réponses). Le conteneur de servlets gère des servlets (sait où se trouvent physiquement les classes Java, pour quelles URL les appeler...), et les exécute lorsqu'elles sont demandées.

**Tomcat, un projet Jakarta :**

* Tomcat est un projet issu de Jakarta, le groupe de projets open-source du groupe Apache dédié à Java. Apache produit un grand nombre d'autres outils *open-source*, et notamment le fameux serveur HTTP Apache, le serveur le plus utilisé au monde. Tomcat est un serveur également d'une grande qualité, même s'il n'est pas le serveur d'applis Java le plus utilisé ;-)
* Comme tous les projets Jakarta, Tomcat est développé par une équipe de bénévoles, et est distribué sous la licence Apache, qui permet son utilisation dans n'importe quel cadre (personnel ou professionnel), y compris commercial. Notons aussi qu'on peut modifier ces projets, sans être contraint de fournir à Apache les sources des modifications effectuées.
* La dernière version stable de Tomcat est la 5.0.18. Si ce site a été écrit en fonction de la version 4, la plupart des informations restent d'actualité dans la version 5. Dans le cas contraire, elles seront signalées. En outre, un chapitre est dédié à la version 5, fraîchement sortie, pour en évoquer les nouvelles fonctionnalités. Il explique également les problèmes de compatibilité existants !

**En résumé :**

Tomcat est un serveur d'applications Java. Les applications web qu'il est capable d'exécuter sont écrites en Java.

* Pour traiter les requêtes entrantes, il existe des connecteurs capables de véhiculer les requêtes du serveur web frontal au serveur d'applis.
* Tomcat est un projet open source, réalisé par la communauté Jakarta, au sein du groupe Apache

**Installation de Tomcat (Windows & Linux)**

* JVM requise
* Tomcat est écrit en Java, et fait tourner des applications écrites en Java. Dans cette circonstance, il est indispensable d'avoir un environnement d'exécution Java installé et en état de fonctionnement sur la machine ! La variable d'environnement JAVA\_HOME doit pointer sur le répertoire d'installation de la JVM (sous Windows, l'installation du SDK ou du JRE gère généralement cela automatiquement, mais sous Linux, il faut le faire).
* Vous pouvez télécharger Java sur le site de son éditeur, Sun
* Installation sous Linux
* Sous Linux, l'installation n'est pas très compliquée non plus - et c'est assez rare pour être souligné ;). Il suffit banalement de décompresser l'archive téléchargée, grâce à la commande tar zxvf jakarta-tomcat-4.1.30.tar.gz --> en remplaçant le nom de l'archive par le bon ;p. N'oubliez pas que la variable d'environnement JAVA\_HOME doit pointer vers le répertoire d'installation de Java...
* Passons maintenant au test du serveur (se référer aux résultats attendus présentés dans la partie *Installation sous Windows* juste au-dessus). Nous appelons TOMCAT\_HOME le répertoire d'installation de Tomcat. Tout d'abord, il faut démarrer le serveur, grâce au script TOMCAT\_HOME/bin/startup.sh. Testons ensuite que Tomcat fonctionne en tapant dans un navigateur l'URL http://localhost:8080/.
* Pour arrêter le serveur, on utilise le script TOMCAT\_HOME/bin/shutdown.sh.
* Une fois Tomcat installé, l'arborescence suivante est disponible dans le répertoire TOMCAT\_HOME (répertoire d'installation) :
* bin/ contient (comme souvent) les fichiers exécutables de Tomcat. En l'occurrence, principalement les scripts Linux ou fichiers batchs Windows permettant le démarrage et l'arrêt du serveur. Mais également l'archive bootstrap.jar, contenant les classes permettant le démarrage et l'initialisation du serveur.
* common/ contient les classes et librairies partagées par les classes internes du serveur et toutes les applications web.
* conf/ contient les fichiers de configuration de Tomcat.
* logs/ contient tous les logs, à moins que, pour certains loggeurs, on ait spécifié d'autres chemins d'enregistrement.
* server/ contient les librairies et les applications web du serveur en lui-même. L'application d'administration ("Admin") par exemple.
* shared/ contient les classes et librairies partagées par toutes les applications web hébergées sur le serveur.
* temp/ est un simple répertoire temporaire.
* webapps/ contiendra les applications web. C'est en tout cas le répertoire par défaut pour ce faire. Car, comme nous l'expliquons dans la configuration, il est tout à fait possible de mettre une application ailleurs, en spécifiant un chemin absolu.
* work/ est le répertoire workDir par défaut, qui contiendra notamment les JSP compilées.

**Déploiement automatique d'applications :**

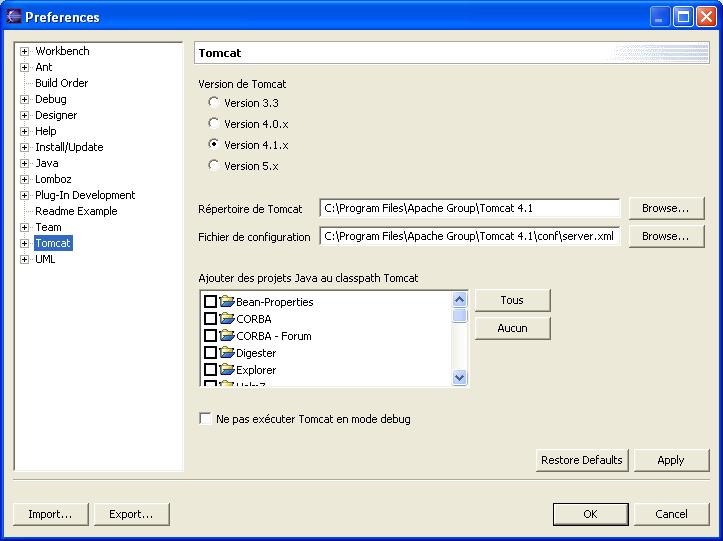
* Si, dans le fichier de configuration de Tomcat, dans une balise Host, vous avez défini l'attribut autoDeploy à *true* (ou que vous ne l'avez pas spécifié, *true* étant sa valeur par défaut), le déploiement automatiquement d'applications aura lieu. Il se comporte de la manière suivante :
* Les archives WAR (Web Application Archives) non déployées sont tous les fichiers \*.war du répertoire appBase (voir configuration de Host) pour lesquels il n'existe pas de répertoire portant le même nom dans ce même répertoire. Exemple : si un fichier "monappli.war" existe, mais qu'il n'y a pas de répertoire "monappli" dans appBase, l'application est considérée comme non déployée. Dans ce cas, le WAR sera automatiquement déployé, dans un répertoire portant le même nom. Attention toutefois, vous pouvez empêcher cette procédure en définissant le paramètre unpackWARs à *false* (voir configuration de Host).   
    
  Donc, si vous souhaitez mettre à jour un WAR, n'oubliez pas de supprimer le répertoire associé pour bénéficier du déploiement automatique au prochain démarrage du serveur.
* En ce qui concerne les applications non compactées (un répertoire est présent dans appBase, sans fichier WAR portant le même nom), ils se verront affecter un Context automatique dans le fichier server.xml. Ce contexte sera configuré en fonction du DefaultContext éventuellement spécifié pour le Host associé. Le chemin pour ce contexte (c'est-à-dire son URL relative) sera le nom du répertoire. C'est-à-dire que, si Tomcat trouve un répertoire nommé monAppli contenant un fichier web.xml (une webapp donc), il lui fournira automatiquement le contexte /monAppli. On pourra ainsi accéder à cette application.
* Dans le répertoire appBase d'un Host, on peut aussi trouver des fichiers de configuration de Context. C'est-à-dire des fichiers XML contenant comme élément de base un Context, et tous les sous-éléments que l'on peut souhaiter. Il s'agit en fait ici de délocaliser les informations de contexte d'une application. Ce qui s'avère intéressant, car ces fichiers peuvent être pris en compte (et les applications correspondantes déployées) sans que le serveur redémarre. Alors que toute modification du fichier server.xml nécessite un redémarrage de Tomcat pour être effective.
* Notons enfin que ce déploiement automatique peut également intervenir pendant l'exécution de Tomcat. En paramétrant liveDeploy à *true* dans un Host, les applications web (WAR, répertoires) ajoutées pendant l'exécution de Tomcat seront elles aussi déployées automatiquement.

**Tomcat & Eclipse :**

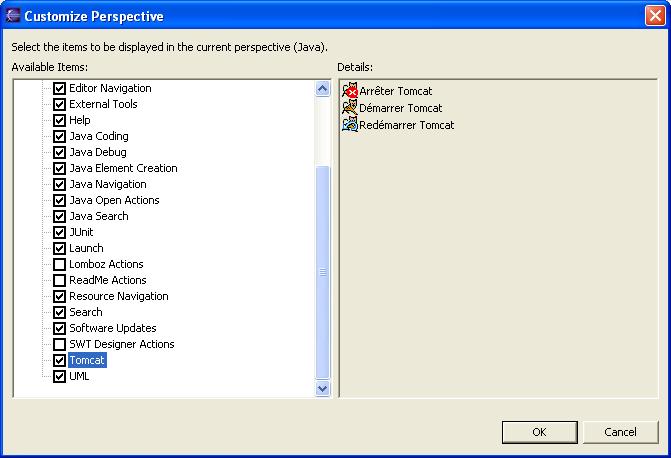
* Eclipse est une plate-forme de développement *open source* créée par IBM, et maintenue par une communauté active. Très orientée Java (elle est destinée à permettre le développement dans d'autres langages, la plate-forme C/C++ est prête), il s'agit de l'environnement de développement Java le plus confortable du moment. Eclipse est basé sur une architecture de plug-ins (tout est plug-in dans ce logiciel). Les développeurs ne s'y sont pas trompés et de nouveaux plug-ins voient régulièrement le jour.
* Eclipse peut être téléchargé sur son site officiel : [http://www.eclipse.org](http://www.eclipse.org/). Si vous êtes intéressés par Eclipse, vous pouvez également vous rendre sur [eclipsetotale.com](http://www.eclipsetotale.com/) qui propose un certain nombre d'informations sur la plate-forme.
* Parmi tous les plug-ins Eclipse, il en est un qui nous intéresse ici au premier chef : le plug-in Tomcat. Développé par Sysdeo, il permet de manipuler un serveur Tomcat depuis Eclipse. Mais également de créer des applications web pour Tomcat directement dans Eclipse. Cela signifie que le plug-in pourra gérer le **Context** dans le fichier de configuration du serveur, qu'on pourra créer une archive WAR à partir de notre projet, debugger les JSP... Fonctionnalités que nous allons détailler plus bas.
* Le plug-in Tomcat pour Eclipse est disponible sur sa page de téléchargement, qui vous renseignera plus avant sur ses fonctionnalités complètes.

**Installation du plug-in :**

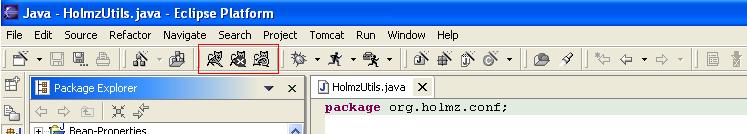
* Le plug-in se présente sous la forme d'une archive ZIP, qu'il convient de décompresser dans le répertoire **plugins** de votre installation d'Eclipse. Au prochain démarrage d'Eclipse, il sera accessible. Dès lors, il faudra suivre ce *modus operandi* pour le faire fonctionner :
* **1)** Affichez la boîte de dialogue "Préférences" d'Eclipse (menu Window -> Preferences). A gauche, vous pourrez accéder à la page de configuration du plug-in, en cliquant sur "Tomcat". A partir de là, cochez le case correspondant à votre version du serveur (notez que le plug-in, à partir de sa version v221, fonctionne avec Tomcat 5). Vous devez ensuite fournir le répertoire d'installation de votre Tomcat (ce que nous avons déjà appelé **TOMCAT\_HOME**). L'emplacement du fichier de configuration **server.xml** est alors supputé automatiquement. Modifiez-le s'il est incorrect. Vous pouvez enfin ajouter des projets au CLASSPATH de Tomcat, ce qui permettra de rendre accessibles les ressources desdits projets à vos applications web.



**2)** Dans le menu "Window", sélectionnez "Customize Perspective...". Cliquez sur "Other", et cochez la case "Tomcat".



**3)** La manipulation 2) a pour effet d'afficher dans votre barre d'outils Eclipse les boutons permettant de démarrer, arrêter ou redémarrer Tomcat. Ils sont cerclés de rouge sur la capture suivante :



**Déploiement d'une application web sous Tomcat :**

* Le déploiement d'une application web sur un serveur d'applications signifie en quelque sorte son installation. Il s'agit en fait de rendre l'application accessible au serveur et de la lui présenter (qu'il sache l'exécuter et lui passer les requêtes la concernant). Nous allons maintenant voir les différents moyens que nous offre Tomcat pour déployer une application.

**Installer un fichier WAR :**

* Tomcat, comme tout moteur de servlet (cela respecte les spécifications) accepte une archive WAR comme application web. Celle-ci devra bien sûr respecter la structure d'une webapp. Ce fichier doit être déposé dans le répertoire **appBase** de l'**Host** sur lequel l'application est installée. En règle générale, ce répertoire est **TOMCAT\_HOME/webapps**. Au prochain démarrage de Tomcat, il détectera le nouveau fichier WAR, et créera automatiquement un **Context** correspondant dans le fichier de configuration. L'application sera dès lors accessible. De plus, en règle générale (sauf configuration contraire), le WAR sera déployé, c'est-à-dire que l'arborescence de fichiers et répertoires qu'il contient sera décompactée dans un répertoire portant le même nom que le WAR.

**Installer l'application décompactée :**

* Nous avons vu qu'un WAR est une archive contenant l'arborescence d'une application web. Tomcat accepte également que l'on dépose directement l'application décompactée dans le répertoire **appBase** de l'**Host** sur lequel l'application est installée. Notons que cela n'est pas un comportement standard, et qu'à ce titre, tous les serveurs d'application Java ne permettent pas forcément cela.

**Créer manuellement le contexte :**

* Il est également possible de créer soi-même un élément **Context** représentant l'application dans le fichier de configuration du serveur, **TOMCAT\_HOME/conf/server.xml**. Cela permet de spécifier un répertoire racine pour l'application différent d'un sous-répertoire du répertoire **appBase** de l'**Host** sur lequel l'application est installée. En effet, Tomcat ne surveille que ce répertoire. Donc, si vous avez copié votre application dans "c:/monappli", il vous faudra créer manuellement le contexte.

**Installer Apache Tomcat [–](http://tutoriels.meddeb.net/installer-apache-tomcat-tomee/) TomEE sur Linux Ubuntu ou compatibles :**

* ? Apache Tomcat est un conteneur de servlet très performant,  Apache TomEE est un serveur d’application JEE dérivé de Tomcat et en hérite beaucoup de ses performances. L’installation est supposée à faire sur une machine de bureau (Ubuntu ou compatibles) pour développer des applications qui les cible ou sur une machine serveur (Debian ou compatibles) pour le test ou la production.

J2EE

* J2EE est l'acronyme de Java 2 Entreprise Edition. Cette édition est dédiée à la réalisation d'applications pour entreprises. J2EE est basé sur J2SE (Java 2 Standard Edition) qui contient les API de base de Java. Depuis sa version 5, J2EE est renommée Java EE (Enterprise Edition).
* Ce chapitre contient plusieurs sections :
* La présentation de J2EE : présente rapidement la plate-forme J2EE
* Les API de J2EE : présente rapidement les différentes API qui composent J2EE
* L'environnement d'exécution des applications J2EE : présente les différents éléments qui composent l'environnement d'exécution des applications J2EE
* L'assemblage et le déploiement d'applications J2EE : décrit le mode d'assemblage et de déploiement des applications J2EE
* J2EE 1.4 SDK : installation et prise en main du J2EE 1.4 SDK
* La présentation de Java EE 5.0 : présente rapidement la version 5 de la plate-forme Java EE
* La présentation de Java EE 6
* La présentation de Java EE 7

**La présentation de J2EE :**

* J2EE est une plate-forme fortement orientée serveur pour le développement et l'exécution d'applications distribuées. Elle est composée de deux parties essentielles :
* un ensemble de spécifications pour une infrastructure dans laquelle s'exécutent les composants écrits en Java : un tel environnement se nomme serveur d'applications.
* un ensemble d'API qui peut être obtenues et utilisées séparément. Pour être utilisées, certaines nécessitent une implémentation de la part d'un fournisseur tiers.
* Sun propose une implémentation minimale des spécifications de J2EE : le J2EE SDK. Cette implémentation permet de développer des applications respectant les spécifications mais n'est pas prévue pour être utilisée dans un environnement de production. Ces spécifications doivent être respectées par les outils développés par des éditeurs tiers.
* L'utilisation de J2EE pour développer et exécuter une application offre plusieurs avantages :
* une architecture d'applications basée sur les composants qui permet un découpage de l'application et donc une séparation des rôles lors du développement
* la possibilité de s'interfacer avec le système d'information existant grâce à de nombreuses API : JDBC, JNDI, JMS, JCA ...
* la possibilité de choisir les outils de développement et le ou les serveurs d'applications utilisés qu'ils soient commerciaux ou libres
* J2EE permet une grande flexibilité dans le choix de l'architecture de l'application en combinant les différents composants. Ce choix dépend des besoins auxquels doit répondre l'application mais aussi des compétences dans les différentes API de J2EE. L'architecture d'une application se découpe idéalement en au moins trois tiers :
* la partie cliente : c'est la partie qui permet le dialogue avec l'utilisateur. Elle peut être composée d'une application standalone, d'une application web ou d'applets
* la partie métier : c'est la partie qui encapsule les traitements (dans des EJB ou des JavaBeans)
* la partie donnée : c'est la partie qui stocke les données

**Les API de J2EE :**

* J2EE regroupe un ensemble d'API pour le développement d'applications d'entreprise.
* Ces API peuvent être regroupées en trois grandes catégories :
* les composants : Servlet, JSP, EJB
* les services : JDBC, JTA/JTS, JNDI, JCA, JAAS
* la communication : RMI-IIOP, JMS, Java Mail

**L'environnement d'exécution des applications J2EE :**

* J2EE propose des spécifications pour une infrastructure dans laquelle s'exécutent les composants. Ces spécifications décrivent les rôles de chaque élément et précisent un ensemble d'interfaces pour permettre à chacun de ces éléments de communiquer.
* Ceci permet de séparer les applications et l'environnement dans lequel elles s'exécutent. Les spécifications précisent à l'aide des API un certain nombre de fonctionnalités que doit implémenter l'environnement d'exécution. Ces fonctionnalités permettent aux développeurs de se concentrer sur la logique métier.
* Pour exécuter ces composants de natures différentes, J2EE définit des conteneurs pour chacun d'eux. Il définit pour chaque composant des interfaces qui leur permettront de dialoguer avec les composants lors de leur exécution. Les conteneurs permettent aux applications d'accéder aux ressources et aux services en utilisant les API.
* Les appels aux composants se font par des clients en passant par les conteneurs. Les clients n'accèdent pas directement aux composants mais sollicitent le conteneur pour les utiliser.

**Les conteneurs :**

* Les conteneurs assurent la gestion du cycle de vie des composants qui s'exécutent en eux. Les conteneurs fournissent des services qui peuvent être utilisés par les applications lors de leur exécution.
* Il existe plusieurs conteneurs définis par J2EE:
* conteneur web : pour exécuter les servlets et les JSP
* conteneur d'EJB : pour exécuter les EJB
* conteneur client : pour exécuter des applications standalone sur les postes qui utilisent des composants J2EE
* Les serveurs d'applications peuvent fournir un conteneur web uniquement (exemple : Tomcat) ou un conteneur d'EJB uniquement (exemple : JBoss, Jonas, ...) ou les deux (exemple : Websphere, Weblogic, ...).
* Pour déployer une application dans un conteneur, il faut lui fournir deux éléments :
* l'application avec tous les composants (classes compilées, ressources ...) regroupés dans une archive ou module. Chaque conteneur possède son propre format d'archive.
* un fichier descripteur de déploiement contenu dans le module qui précise au conteneur des options pour exécuter l'application

**Le conteneur web :**

* Le conteneur web est une implémentation des spécifications servlets et par extension des spécifications des JSP. Ce type de conteneur est composé de deux éléments majeurs : un moteur de servlets (servlet engine) et un moteur de JSP (JSP engine).
* Les conteneurs web peuvent généralement utiliser leur propre serveur web et être utilisés en tant que plug-in d'un serveur web dédié (Apache, IIS, ...).
* L'implémentation de référence pour ce type de conteneur est le projet open source Tomcat du groupe Apache.
* Les API spécifiquement mises en œuvre dans un conteur web sont détaillées dans les chapitres «Les servlets»" et "JSP".

**Les services proposés par la plate-forme J2EE**

* Une plate-forme d'exécution J2EE complète implémentée dans un serveur d'applications propose les services suivants :
* service de nommage (naming service)
* service de déploiement (deployment service)
* service de gestion des transactions (transaction service)
* service de sécurité (security service)
* Ces services sont utilisés directement ou indirectement par les conteneurs mais aussi par les composants qui s'exécutent dans les conteneurs grâce à leurs API respectives.