REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE AMMAR TELIDJI LAGHOUAT

FACULTE DES SCIENCES

DEPARTEMENT DE MATHEMATIQUES ET INFORMATIQUE

MEMOIRE EN VUE DE L'OBTENTION DU DIPLOME DE MASTER

EN INFORMATIQUE

OPTION : RESEAUX, SYSTEM ET APPLICATION REPARTI

THEME :

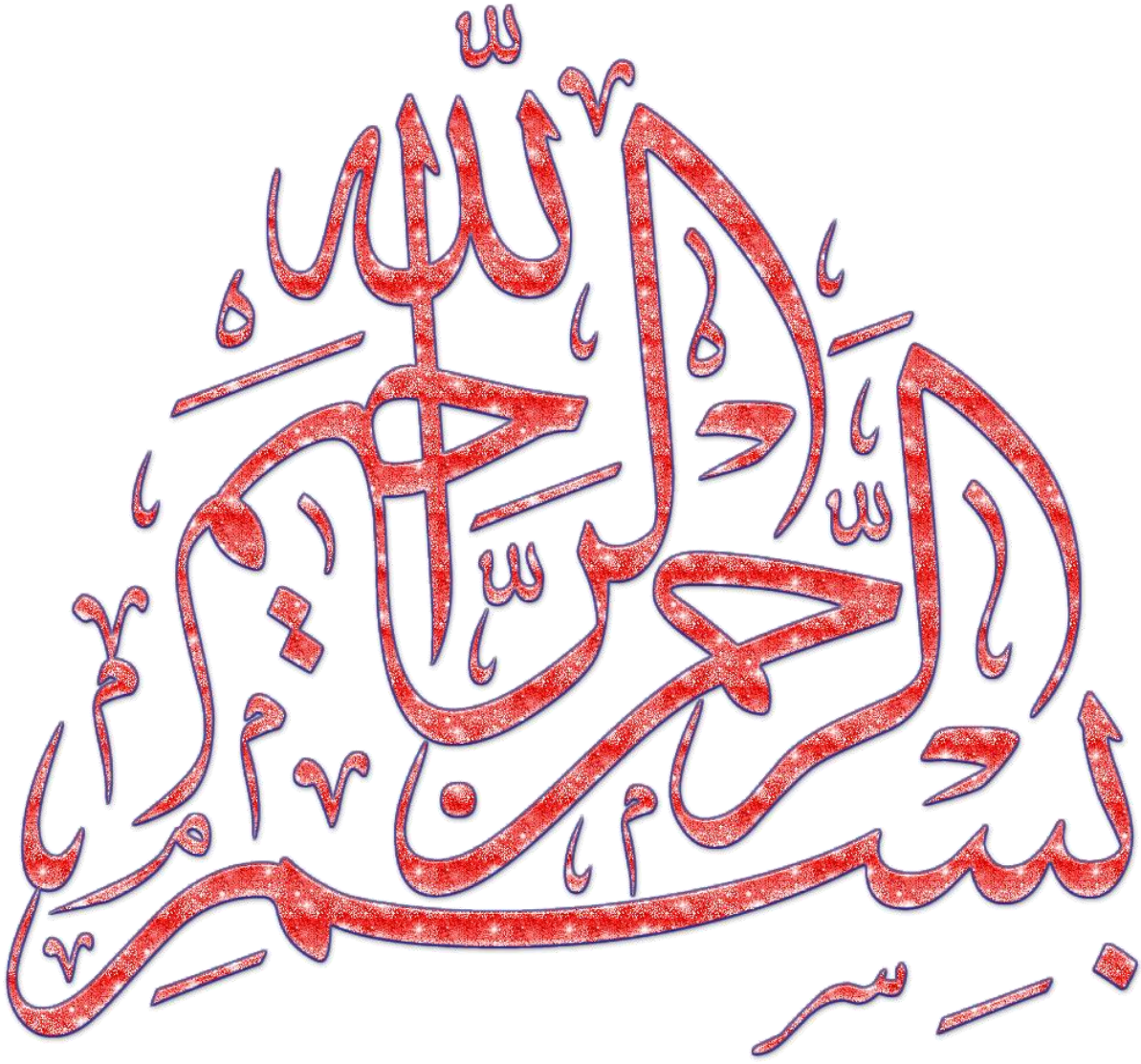
Etude et mise en œuvre de serveur d'application JBoss

Présenté Par : Mme GHOZLANE Fatiha

Soutenu devant le jury composé de :

Mr. ZIANI Ben Ammeur	MAA	président de jury	Université de Laghouat
Mr. GUELOUMA Younes	MAA	Examinateur	Université de Laghouat
Mr. KECHNA Lakhdar	MAA	Examinateur	Université de Laghouat
Melle. BELABBACI Amel	MAA	Encadreur	Université de Laghouat

Année Universitaire : 2014-2015


بسم الله الرحمن الرحيم

Résumé

Le but de notre travail est l'étude de l'architecture et la mise en œuvre d'un serveur d'application. Le serveur d'application étudier est JBoss, vu sa popularité et sa puisance. La mise en œuvre a été réalisée sur Virtuel Box/ubuntu 14.04, et poura être adaptée facilement dans un contexe réel, le centre de calcul du departement d'informatique par exemple.

Les mots clés : JBoss, serveur d'application, middleware.

Dédicace

Je dédie ce travail de fin d'études à :

Mes parents, ceux à qui je dois tant pour leur amour et leur support continu tout le long du cursus de mes études. Que ce travail soit le témoignage sincère et affectueux de ma profonde reconnaissance pour tout ce que vous avez fait pour moi. Grâce à leur collaboration, leur détermination dans le travail m'a soutenu au prix de sacrifices inoubliables.

- A mon cher mari, Abed el Aziz pour son aide précieuse et sa persévérance toute au long de mon projet.
- A mes très chères sœurs : Hayat, Sarah, Nour, et la petite Iman.
- A mon très cher bébé.
- A mes chers frères : Yacine et Sofiane.
- A ma chère belle mère Houria, mon cher beau père lazhari.
- A toute mes grandes familles GHOZLANE et BOUNEIF et TOOBI sans exception
- A tous mes amis.

J'exprime ma reconnaissance à tous ceux qui ont contribué de prés ou de loin à accomplir ce modeste travail.

Fatiha

Remerciement

Je tiens à remercier tout d'abord « Allah » le tout puissance, de m'avoir donné le courage, la volonté, la force et la patience afin de parvenir à terminer ce modeste travail.

Je voudrai adresser l'expression de ma gratitude, ma profonde sympathie ainsi que mes vifs remerciements à :

- A mon promoteur Melle. BELABBACI Amel pour avoir dirigée et guidée ce travail. Je le remercie surtout pour son entière disponibilité et ainsi que pour la marque de confiance qu'il m'a donnée pour accomplir cette étude.
- à qui m'a fait l'honneur de présider le jury, à pour avoir accepté de faire partie de ce jury.
- à tous les professeurs qui m'ont donné le meilleur d'eux même en contribuant à augmenter mon savoir durant mes études.

Enfin, je remercier toutes personnes qui m'a aidé et encouragé de près ou de loin, à la réalisation de ce mémoire et durant mon chemin universitaire.

Merci

Table des matières

Table des figures

La liste des abréviations

AOP Aspect oriented programming**.**

API Application Programming Interface.

ASP Active Server Pages.

CGI Common Gateway Interface (interface de passerelle commune).

COM+ Component Object Model +.

EAI Enterprise Applications Integration.

EOF Enterprise Objects Framework.

ERP Enterprise Resource Planning.

EJB Enterprise JavaBeans.

IDE Integrated Development Environment.

JDNI Java Naming and Directory Interface.

JMS Java Message Service.

JMX Java Management Extensions.

JSP Java Server Pages.

J2EE Java 2 Enterprise Edition.

LDAP Lightweight Directory Access Protocol.

RMI Remote Method Invocation.

WAP Wireless Application Protocol.

WSDL stands for Web Services Description Language.

WML Wireless Markup Language.

Glossaire

API Application Programming Interface : ensemble des règles d'utilisation d'un service applicatif par un programme.

Back-office Le back-office regroupe la partie du système d'information à laquelle l'utilisateur n'est pas confronté. Il s'agit des bases de données, des moniteurs transactionnels, des ERP, etc.

CGI Common Gateway Interface : interface de programmation pour les serveurs web, permettant de faire le lien entre le serveur HTTP et le serveur d'applications.

COM Component Object Model : modèle objet de Microsoft.

CORBA Common Object Request Broker Architecture : proposition de standard émise par l'OMG concernant les échanges entre objets dans les applications tournant sur des architectures distribuées.

DCOM Distributed Component Object Model : est une technique propriétaire de Microsoft qui permet la communication entre des composants logiciels distribués au sein d'un réseau informatique.

EAI Enterprise Application Integration : ensemble des logiciels visant à faciliter l'intégration entre les applications hétérogènes de l'entreprise.

EAR Enterprise Archive : fichier archive JAR contenant une application J2EE.

EJB Enterprise JavaBean : composant Java renfermant la logique métier d'une application.

ERP Enterprise Resource Planning : progiciel permettant de gérer l'ensemble des processus d'une entreprise, en intégrant l'ensemble des fonctions de cette dernière comme la gestion des ressources humaines, la gestion comptable et financière, l'aide à la décision, mais aussi la vente, la distribution, l'approvisionnement, le commerce électronique.

Framework	Ensemble de composants pré-assemblés qui modélisent au mieux les opérations d'un domaine applicatif donné. Le framework est un guide d'architecture permettant de générer rapidement une application pour ce domaine.
Gestion de contexte	Fonctionnalité du serveur d'applications gérant les données et le contexte d'un internaute pendant une session.
GUI	Graphical User Interface : interface homme-machine de type multifenêtres Windows.
IDE	Integrated Development Environment : atelier de développement intégré.
Instance	Programme à l'exécution. Un programme peut être instancié plusieurs fois. Appelé également " process ".
LDAP	Lightweight Directory Access Protocol : protocole d'accès aux annuaires.
Log http	Fichier où sont enregistrées les requêtes HTTP (l'heure, la date, l'adresse IP…).
J2EE	Java 2 Enterprise Edition : extension de la plate-forme standard qui intègre les technologies nécessaires au développement des applications d'entreprise.
JAR	Java Archive : extension des fichiers d'archive Java.
JDBC	Java DataBase Connectivity : API Java standard permettant l'accès aux bases de données relationnelles.
JDK	Java Development Kit : ensemble d'API définissant une version de Java.
JMS	Java Message Service : API de J2EE définissant un modèle d'accès à un service de messages dans un contexte d'échange d'informations.
JNDI	Java Naming and Directory Interface : API de J2EE définissant un modèle d'accès à un service de nommage et d'annuaire.
JSP	Java Server Pages : API de J2EE définissant un modèle de création de pages dynamiques. Cette norme permet d'insérer du code Java au sein des pages HTML.
JVM	Java Virtual Machine : programme permettant d'exécuter du code Java.

Mapping	Le terme signifie " faire correspondre ". Mapping objet-relationnel signifie faire correspondre des données objet dans le modèle relationnel.
PGI	Progiciel de Gestion Intégré.
Moniteur transactionnel	un moniteur transactionnel (ou « TP monitor » pour « Transaction Processing Monitor ») est un système transactionnel de partage des ressources machine (mémoire, base de données, etc.). Les éléments gérés par le moniteur transactionnel sont des transactions dont le champ d'application varie suivant les produits.
Pooling de connexions	C'est la manière d'accéder par des pools à une base de données. Un pool de connexions est un groupe de connexions vers une base de données.
RMI	Remote Method Invocation : technologie permettant à un objet s'exécutant sur une machine de faire appel aux méthodes d'un objet s'exécutant sur une autre machine.
Servlet	Programme Java exécuté côté serveur qui renvoie au serveur web un flux HTTP (HTML par exemple).
Thread	Plus petite unité d'un processus ou tâche élémentaire exécutée par le système d'exploitation.
WAR	Web Archive : fichier archive JAR contenant une application web (module et ressources JSP/Servlet, HTML) ;

Chapitre 01:

Introduction générale 1

1.1 Préambule :

Avec l'évolution des nouvelles technologies, les entreprises, pour la plus part, utilisent de plus en plus l'outil informatique, devenu presque indispensable. Quelques soit le domaine d'activité des entreprises, des applications informatiques appropriées sont développées, pour favoriser et améliorer toutes les étapes du processus métier, allant de la production à la comptabilité.

Ces applications deviennent elles aussi de plus en plus importantes, nécessitant beaucoup de ressources, et par conséquent, des systèmes plus puissants.

Aujourd'hui, ces applications sont centralisées sur de gros systèmes et accessibles à partir des machines clientes. Ces gros systèmes sont appelés Serveurs d'applications, et peuvent contenir plusieurs applications pour l'entreprise. On peut également avoir plusieurs serveurs d'application au niveau de l'entreprise.

L'objectif de ce mémoire est d'étudier les fonctionnalités des serveurs d'applications et d'expérimenter leurs avantages et utilités en développent des exemples d'applications pour la gestion d'un centre de calcul avec le serveur d'application JBoss AS.

1.2 Objectifs :

Dans notre travail, on s'intéresse à étudier les serveurs d'applications.

Les objectifs de ce projet sont :

1. Etudie les types, la fonctionnalité, et le rôle des serveurs d'applications.
2. Etudier les architectures des serveurs d'applications.
3. La mise en œuvre du serveur d'applications JBoss.

1.3 La structure du mémoire :

Ce mémoire est structuré en cinq chapitres qui sont organisés comme suit :

Chapitre 02 : Généralité sur les serveurs d'applications

Présentation des origines des serveurs d'applications qui ont des différents concepts théoriques. Une vue générale sur son rôle, son architecture et ses déférentes typologies. En

mettant l'accès sur les grandes fonctions du serveur d'applications au cœur de l'architecture web.

Chapitre 03 : Les Middlewares

Ce chapitre commence avec une brève historique résumant les grandes étapes de l'évolution des Middlewares, les besoins auxquels elles répondent, et les principales catégories de Middleware.

Chapitre 04 : Serveur d'applications JBOSS AS7

Ce chapitre expose les concepts fondamentaux de serveur d'application JBoss as7. Après un historique très riche, et une terminologie pour définir le vocabulaire employé par notre travail; ce chapitre explique l'architecture de JBoss et comporte une étude comparatif entre JBoss AS7et un autre fameux serveur d'applications Glasfish. La dernière partie explique les modèles de sécurité de JBoss as7.

Chapitre 05 : La mise en œuvre de JBoss AS

Consiste à expliquer et implémenter les fonctions proposées pour la gestion de centre du calcul, passons par l'étape de configuration, puis l'étape de déploiement des applications.

Conclusion générale :

La conclusion générale résume les résultats de notre travail, et présente les perspectives que Nous souhaitons réaliser dans le futur.

Chapitre 02 :

2 Généralité sur les Serveurs d'applications

Nous nous concentrons dans ce chapitre sur les notions de base de serveur d'applications, qui est devenue un moyen très important dans le domaine des applications web. Nous nous aborderons aux différents concepts liés à l'architecture web des serveurs d'applications, le rôle fondamental qu'il joue dans les systèmes pour aider les administrateurs de gérer ces applications web, et les différentes catégories des serveurs d'applications rendre le serveur d'applications capable de répondre aux différents besoins selon les ressources disponibles.

2.1 Historique :

Dans un premier temps, les éditeurs de serveurs web ont développé plusieurs types d'interfaces afin d'insérer des données dynamiques dans les pages. L'interface CGI demeure l'une des plus populaires. Lors de la réception d'une requête HTTP, le serveur web lance l'exécution d'un programme accédant au SGBD et fournit, en retour, les pages contenant les données lues et prêtes pour l'affichage. L'interface CGI présente toutefois quelques défauts. À commencer par de piètres performances, son fonctionnement se révélant assez complexe. à chaque requête HTTP, le serveur web lance un programme qui ouvre la base de données, lit les informations, puis ferme le SGBD. De plus, cette interface n'assure pas le maintien de l'état des sessions utilisateurs entre les requêtes HTTP. Pour pallier ces problèmes, une nouvelle classe d'infrastructures logicielles, appelées serveurs d'application, a été élaborée. Le principal rôle de cette première génération de plates-formes consistait à relier les serveurs web aux SGBD. Les technologies employées sont les servlets et les JSP du monde J2EE, les ASP de Microsoft ou les scripts PHP. Toutes fournissent aux entreprises des moyens simples pour gérer des interactions dynamiques avec le web, au travers de sessions dont le contexte est maintenu entre les requêtes HTTP. Les éditeurs de serveurs d'application proposent des modèles de composants métier (Com+, EJB) en complément des pages ASP/JSP, de façon à mieux structurer en couches la logique applicative. On obtient alors une découpe en 4 couches: client web, logique de présentation web, logique métier et accès aux données .

Cette segmentation facilite la réutilisation du logique métier. Toutefois, les composants métier restent encore peu utilisés, car ils impliquent de nouvelles méthodes de travail et une réflexion architecturale hors de portée de bon nombre d'entreprises.
10 % des applications web recourent à des composants Com+ ou EJB. Toutefois, la situation est en train d'évoluer avec le couplage croissant des applications web avec le système d'information des entreprises, ce qui implique des services *middlewares* d'une complexité plus élevée. Naturellement, la mise au point de ces services se révèle assez longue, étant donnée la réalisation de composants facilitant leur utilisation. La maturité des modèles de composants proposés avec les serveurs d'application fut donc également longue à obtenir. Voilà pourquoi il a fallu attendre la version 2 ou 3 de ces modèles de composants, tant du côté Com+ que du côté des EJB, pour un véritable déploiement sur le marché.
Les modèles de composants Com+/EJB fournis avec les serveurs d'application simplifient grandement l'usage des services *middlewares* de ces derniers, tels que la gestion des transactions et de la sécurité, l'activation et la désactivation des composants. Toutes ces invocations de services *middlewares* sont gérées par la structure d'accueil des composants, dénommée *"conteneur de composant"*. Le comportement de chaque composant vis-à-vis de ces aspects est déclaré par un descripteur de déploiement, fixé le plus souvent lors de l'intégration de l'application.
Les serveurs d'application proposent de nombreuses interfaces d'accès, chargées d'assurer l'interopérabilité des composants métier avec le monde extérieur : communication synchrone prise en charge par un courtier d'objet (DCom, Corba...), communication asynchrone via un middleware orienté message, accès aux bases de données et aux applications d'entreprise via des connecteurs adéquats, etc. Cette interopérabilité place de fait les serveurs d'application au centre des nouvelles architectures informatiques.
Ces plates-formes assurent également la disponibilité et la montée en charge des composants, au travers de services d'équilibrage de charge et de basculement sur panne à l'intérieur de serveurs multiprocesseurs ou de fermes de serveurs. L'ensemble de ces services est géré grâce aux consoles d'administration, fournies avec les serveurs d'application.

2.2 Définition d'un serveur d'application :

Un serveur d'application est un environnement informatique qui fournit les briques nécessaires à l'exécution d'applications transactionnelles sur le web. Le terme serveur d'applications signifiera à tour de rôle un serveur d'objets, un moteur d'exécution pour le Web, un outil de développement, d'administration, de déploiement et d'exécution. [4]

Un serveur d'applications doit :

1. **s'interfacer avec un serveur http :** Pour le développeur, ceci se traduit par l'absence de développement pour envoyer une chaîne vers le serveur web, afin de se concentrer sur le contenu et non pas sur la façon d'envoyer la page web. Celle-ci sera de type HTML ou l'un de ses dérivés, de type XML, ou encore de type WML pour les clients WAP.
2. **fournir un moteur d'exécution des traitements :** Le moteur d'exécution des traitements représente la condition indispensable pour réaliser des applications dynamiques. Il est intéressant à ce niveau de noter qu'il est possible de créer un exécutable indépendant, mais que ce mode de fonctionnement ne facilite pas la couverture des autres points décrivant un serveur d'applications. Avec l'arrivée massive de Java et de J2EE côté serveur, ce moteur d'exécution est aujourd'hui majoritairement la machine virtuelle Java (JVM).
3. **s´ouvrir sur le système d´information de l´entreprise (XML, web services, connecteurs SGBDR,...) :** Le troisième point est l'ouverture vers l'existant. Cet aspect est important dans la mesure où les données et les traitements à réutiliser sont généralement présents dans d'autres éléments de l'architecture. Le serveur d'applications doit donc permettre l'accès aux SGBDR, aux ERP, aux moniteurs transactionnels, aux systèmes centraux, etc...
4. **l'ajout de fonctionnalité :** l'ajout de fonctionnalité est initialement prévu pour permettre aux équipes de développement de s'affranchir de l'ensemble des problématiques techniques des architectures web. Cependant, des contextes particuliers de l'entreprise impliquent parfois une amélioration du moteur, ou la nécessité de développer une fonctionnalité non prévue dans le serveur d'applications. Pour cela, il doit être possible d'enrichir ou de passer outre les fonctionnalités du serveur d'applications, afin de modeler son environnement à un contexte bien spécifique.
5. **répondre aux contraintes induites par les architectures centralisées :** Un serveur d'applications doit être adapté pour répondre aux problématiques techniques inhérentes à l'architecture centralisée (données augmente et nécessite une sécurité accrue, de plus une haute sollicitation d'entré et sortie nécessite une technologie avancé à tous les niveaux (CPU, bus, accès disques, transferts bandes), les applications

deviennent critiques et sensibles du fait de la centralisation et la sauvegarde est rendue difficile car les Batch augmentent et la fenêtre de sauvegarde diminue, sans oublier bien sûr que la production est de 24h/2). A ce niveau, on retrouve entre autres la gestion de contexte nécessaire pour différencier les utilisateurs, la répartition de charges et le pooling de connexions pour offrir des solutions palliatives aux montées en charge, ou encore la fonctionnalité de reprise sur incident pour rendre l'application disponible à tout instant.

2.3 Serveur web versus serveur d'applications

Le serveur web et le serveur d'applications sont séparés l'un de l'autre. Ces deux composants sont nécessaires, puisqu'ils se complètent : le serveur web ne sait pas exécuter d'applications et le serveur d'applications ne sait pas traiter une requête http, Si ces deux composantes sont indispensables, elles ne sont pas nécessairement séparées dans le cas où un serveur d'application qui inclut un serveur web, et est donc capable de traiter à la fois les requêtes HTTP simples et les applications web.

Le principe est de changer de connecteur (figure 2.1), pour en utiliser un comprenant les requêtes HTTP et non plus les requêtes triées venant du serveur web.

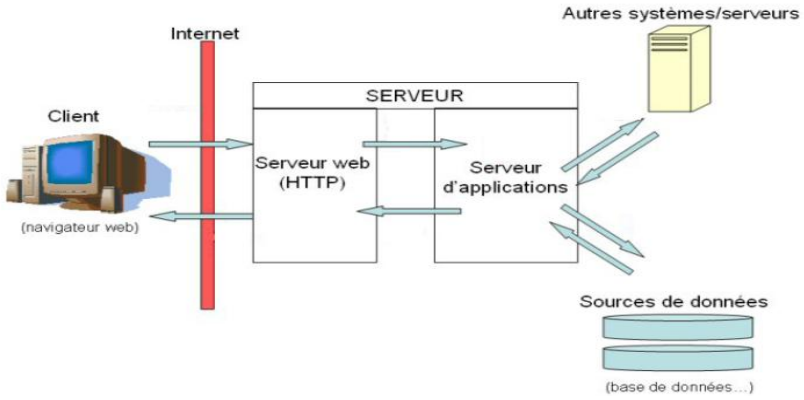


Figure 2.1 - Serveur web versus serveur d'applications [3].

2.4 Rôle des serveurs d'applications

Le serveur d'applications trouve sa place au sein des architectures web dynamiques. En dehors des architectures à pages statiques ne nécessitant l'utilisation que d'un serveur web, et des scripts CGI. Son rôle est de fournir la dynamicité nécessaire aux applications. Sa place exacte se trouve entre le serveur HTTP qui reste le garant du dialogue standardisé avec le client, et entre le reste de l'architecture (partie appelée parfois le back-office) qui peut être plus ou moins riche et complexe.

Le back-office désigne l'ensemble des parties du système d'information aux quelles l'utilisateur final n'a pas accès. Il s'agit donc de tous les processus internes à l'entreprise

(exemple : production, logistique, stocks, comptabilité, gestion des ressources humaines, etc.). Dans la grande majorité des cas, le back-office des architectures se compose de SGBD, d'ERP (Progiciel de Gestion Intégré), de middleware d'accès divers, d'EAI (Enterprise Application Integration) ou de tout système existant tels que les mainframes et les serveurs centraux. [1].

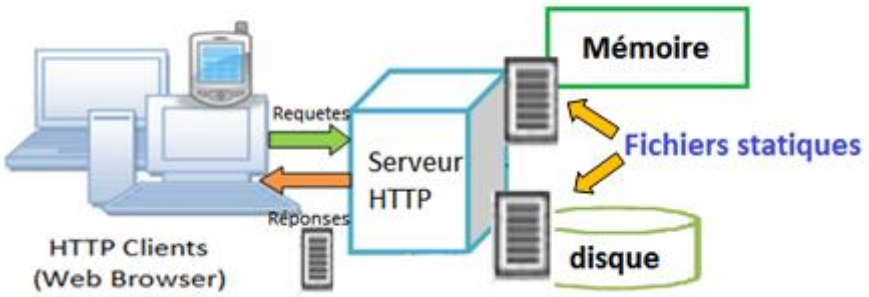


Figure 2.2 - Architecture web statique.

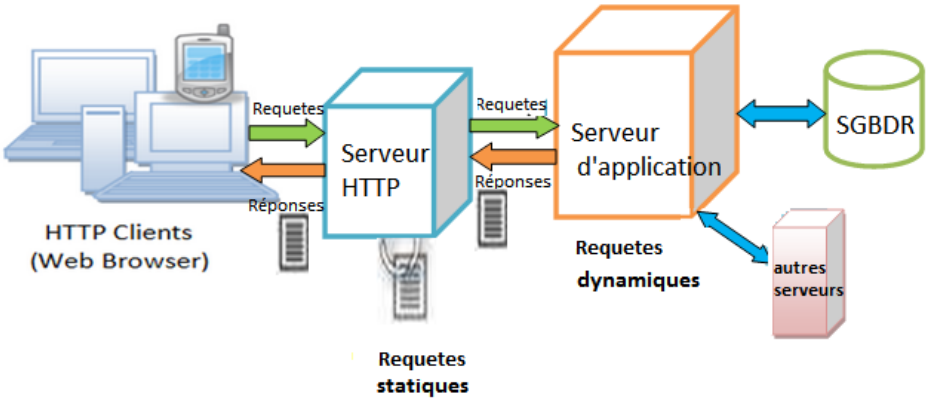


Figure 2.3 - Architecture web dynamique.

Quand le serveur web (serveur HTTP) délivre au client les réponses aux requêtes émises par ce client, en lui fournissant des éléments statiques (pages HTML, images…). Dès que le contenu à délivrer est dynamique (c'est-à-dire élaboré à la demande), c'est le serveur d'applications qui prend le relais en laissant le serveur HTTP devenir un simple intermédiaire entre lui et le navigateur. Le dialogue entre le navigateur et le serveur d'applications est un dialogue régi par le protocole http, où son principe basé sur le paradigme de requête-réponse.

Le client émet dans tous les cas une demande de page web, que son contenu soit statique ou dynamique. S'il perçoit bien des pages web, c'est en réalité tout simplement parce qu'il les reçoit. Que la page soit statique ou qu'elle soit construite dynamiquement par le serveur, celle-ci sort du serveur et arrive sur le poste client de la même manière qu'une page statique, pouvant ainsi donner l'impression que toute page demandée est nativement statique. Ce

mécanisme est important car il impose que toute architecture web, aussi complexe soit elle, doit avant tout, construire ces pages avant de les envoyer aux utilisateurs. C'est là un des rôles fondamentaux du serveur d'applications. En d'autres termes, même si du côté back-office la notion de page disparaît parfois au profit d'une gestion de contenu, c'est bien le serveur d'applications qui doit gérer l'élaboration de ces pages à renvoyer au navigateur.

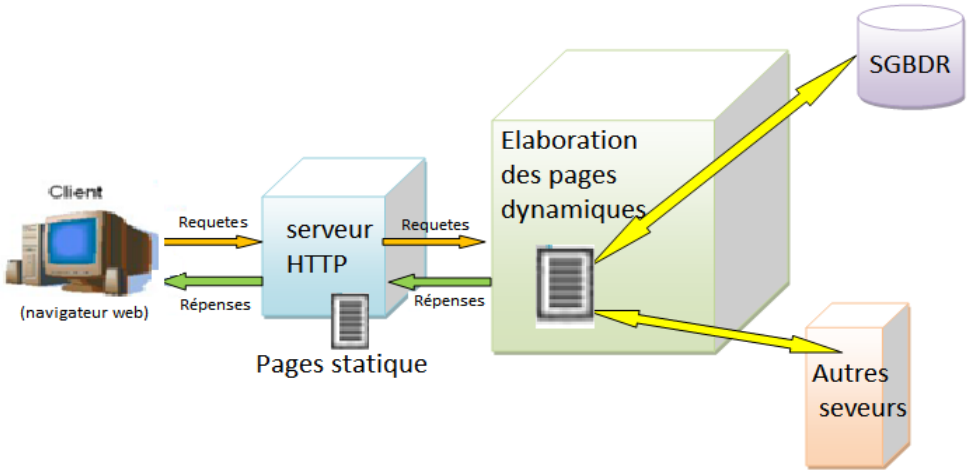


Figure 2.4 – Elaboration des pages dynamique.

2.5 Types des serveurs d'applications

2.5.1 Serveurs d'applications de type Scripting

Dans le milieu des serveurs d'applications, Les serveurs d'applications de type Scripting représentent une catégorie bien particulière. L'élément fédérateur de ce type de serveur d'applications est la page HTML statique. Et pour les pages HTML dynamiques, chaque page HTML dynamique d'une application web issue d'un serveur d'applications de type Scripting correspond à un et un seul fichier source.

Ce fichier contient un ensemble de données liées à l'interface HTML (balises HTML + langage de Scripting client) ainsi que l'ensemble du code applicatif destiné à donner de la dynamicité à l'application. Il y a en effet une relation d'identité entre la page HTML visualisée par l'utilisateur, et le fichier source élaboré par le développeur, de manière similaire à un contexte d'enchaînement de pages HTML statiques. Il est possible d'externaliser des traitements dans des fichiers externes. La différence de traitement se fait simplement lors de la demande d'une page par l'utilisateur. Soit il s'agit d'une demande de page HTML statique et le serveur HTTP renvoie cette page, soit il s'agit d'une page dynamique, et le serveur HTTP délègue la gestion de la page au serveur d'applications. Celui-ci va analyser et traiter le fichier source demandé pour retourner via le serveur HTTP la page générée vers le navigateur de l'utilisateur. [1]

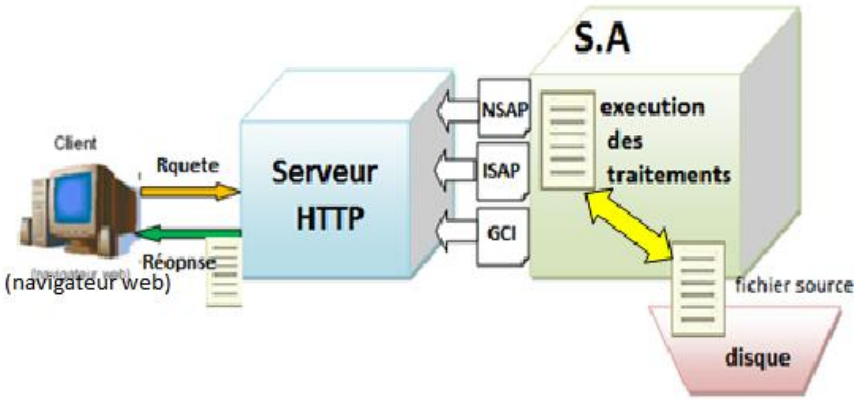


Figure 2.5 - Architecture dynamique de type Scripting.

Dans une telle situation, le rôle du serveur d'applications sera avant tout de parser le contenu du fichier source. Appliquée à un serveur de type Scripting.

Une interprétation plausible du terme " parser " peut être représentée par l'ensemble des séquences suivantes :

- ouverture du fichier.
- parcours de son contenu.
- interprétation de la partie " script serveur ".
- assemblage des parties statiques avec le résultat des traitements générés par l'interprétation du code.
- Une fois la concaténation des différentes parties effectuée, le résultat est renvoyé au serveur HTTP.

2.5.2 Serveurs d'applications de type objet

Le serveur d'applications de type objet, est un serveur d'application dont le principe technique repose sur des mécanismes internes propres.

Dans ce cas, les traitements fonctionnels sont généralement séparés des traitements d'interface, eux-mêmes pouvant être séparés des ressources d'interface (comme les éléments d'interface statique purs).

Ce type de fonctionnement complexifie l'architecture applicative et avec elle sa propre modélisation. En revanche, l'atout principal de ces serveurs d'applications, hormis leur modélisation objet, est d'offrir une souplesse dans la répartition et la gestion des traitements. Il devient possible de répartir les traitements choisis à l'endroit où on le souhaite, en environnement distribué, par exemple.

2.5.3 Serveurs d'applications de type J2EE

Parmi les serveurs d'applications de type objet se démarque le fameux serveur d'application J2EE comme une solution majeure, dont architecture repose sur l'ensemble des spécifications Java 2 Enterprise Edition éditées par Sun.

L'idée principale de J2EE est de fournir le maximum de spécifications qui répondent aux besoins d'un développement applicatif et cela dans un environnement Java. A ce titre, toute implémentation des spécifications respectant J2EE constitue un serveur d'applications à part entière. [1].

J2EE est une norme qui va spécifier à la fois l'infrastructure de gestion de nos applications et les API des services utilisées pour concevoir ces applications. La plateforme J2EE est essentiellement un environnement fournissant :

- Une infrastructure d'exécution pour faire tourner vos applications.
- Un ensemble de services accessibles via l'API J2EE pour vous aider à concevoir vos applications.

Il s'agit des API Java Servlet, JSP et EJB. Ces normes apparaissent complémentaires et répondent globalement à des choix de développement bien spécifiques.

Java Servlet : les implémentations des API Java Servlet fournissent une méthode d'extension des serveurs HTTP à des traitements dynamiques via un moteur de servlet.

Les servlets sont des programmes écrits en Java. Dans ce cas, l'exécution des servlets permet de réaliser des traitements Java côté serveur, dont l'un des objectifs est de renvoyer au serveur HTTP le flux (HTML par exemple) constituant la réponse que le serveur HTTP retransmet au client.

La page (flux HTML) que reçoit le navigateur est donc présente, statiquement ou dynamiquement, dans le code Java de la servlet.

JSP (Java Server Pages) : l'implémentation des API JSP permet, par opposition aux API Java Servlet, d'embarquer du code Java au sein même des pages HTML. Ce moyen permet de retrouver la notion de page d'interface au sein de laquelle les traitements sont définis.

Ainsi, les concepteurs peuvent retoucher graphiquement, à l'aide d'un outil WYSIWYG par exemple, le design de leurs pages HTML. Il est important de noter qu'à l'exécution, le moteur de JSP génère une servlet.

2.6 Caractéristiques d'un serveur d'applications :

Au cœur de l'architecture web, le serveur d'applications fournit un ensemble de services aux

applications. Puisque le serveur d'applications prend une position centrale, leur position le conduit à offrir ses services à plusieurs niveaux, en peut imaginer qu'il est placé entre le système d'exploitation situé en bas, l'application en haut, le serveur web à gauche et le back-office à droite. pour compléter les fonctionnalités du serveur d'applications, l'outil de développement et celui d'administration ont chacun leur rôle à jouer en phase de développement et de production.

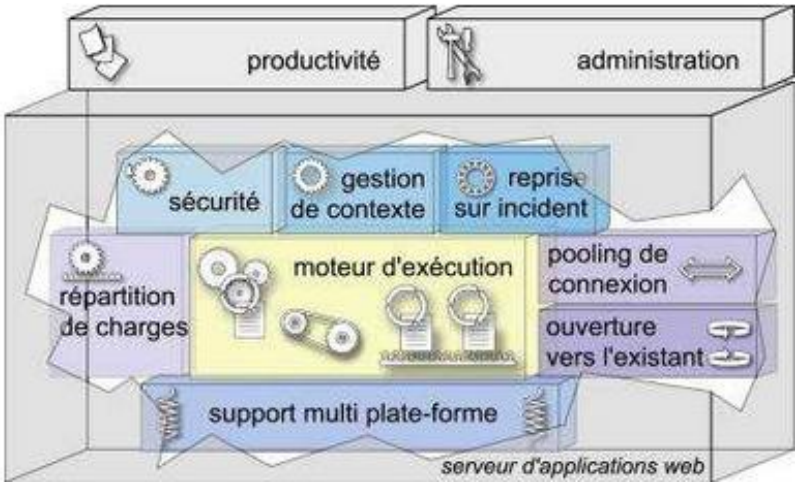


Figure 2.6 - Schéma du serveur d'applications et de ses services.

Voici une liste des grandes fonctionnalités attendues par un serveur d'applications :

a) Le support des plates-formes

Le support des principales plates-formes du marché est une qualité première attendue chez ce type d'outil. Tout l'enjeu de cette fonctionnalité est de désolidariser la technologie applicative de la machine hébergeant l'application afin d'offrir davantage de liberté et d'évolutivité. Pour ce faire, le serveur d'applications doit implémenter un ensemble d'API propre à chacun des nombreux systèmes d'exploitation.

A titre d'exemple, lorsqu'une application fait appel à des fonctions dites système telles que l'ouverture, l'écriture, la lecture ou la suppression de fichiers, le serveur d'applications doit être en mesure de les exécuter indépendamment de la plate-forme utilisée.

b) La répartition de charges

Parmi les fonctionnalités d'un serveur d'application, la répartition de charges joue un rôle essentiel. D'un point de vue technique, est un ensemble de techniques permettant de distribuer une charge d'exécution de plusieurs instances (ou process) sur différentes machines serveur. Cette architecture permet également de diviser l'application en modules différents, sans avoir l'obligation de reproduire sur chacun des serveurs la totalité d'une application. Ainsi, un module peut être isolé sur un seul serveur afin d'absorber plus facilement la charge.

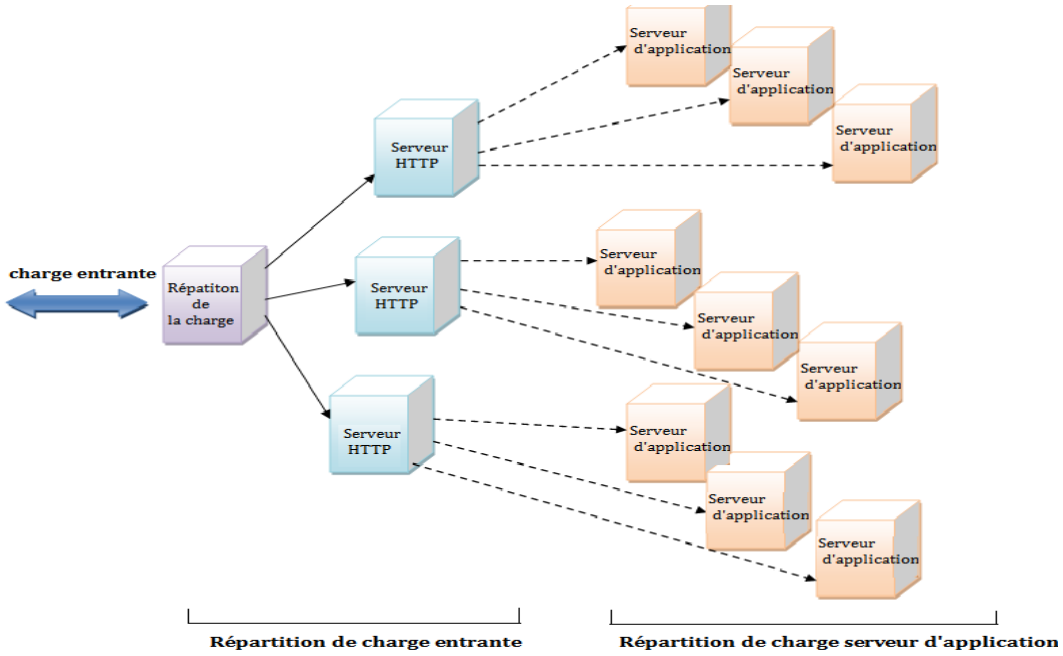


Figure 2.7 - Schéma pour illustrer les fonctionnalités de répartition de charges.

c) La disponibilité

Dans certaines configurations, une application est répliquée sur plusieurs serveurs physiques, donc il est plus facile de mettre en place la fonctionnalité de reprise sur incident. Dans le cas ou une panne se produit au niveau du serveur ou au niveau applicatif, la requête de client est réorientée vers un autre serveur disponible de manière transparente. À ce niveau, la tolérance aux pannes permet d'offrir la disponibilité au niveau application puisqu'il existe toujours un module actif pouvant être invoqué par les utilisateurs. Pour cela, le serveur d'applications doit être également capable de sauvegarder l'ensemble des opérations effectuées par l'utilisateur (du contexte utilisateur). Cela implique la réplication des sessions utilisateur sur une autre machine. Cette réplication se fait soit en base de données, soit sur disque, soit en mémoire. Les serveurs d'applications les plus avancés automatisent la gestion de reprise sur incident au niveau session. [1].

d) Le pooling de connexions

Dans l'architecture web, le serveur d'application permet au client d'accédé à la base de données, mais les temps de réponse deviennent catastrophiques. La solution la plus utile consiste à passer par un pool de connexions. Le principe de pooling de connexions consiste à démarrer un nombre prédéfini de connexions vers un SGBDR. Le serveur d'applications dirige ensuite les demandes utilisateur en répartissant les différentes requêtes sur les connexions disponibles. Ceci permet d'avoir la maîtrise du nombre de connexions maximales ouvertes et d'éviter le goulet d'étranglement à ce niveau. Dans un contexte où l'application est

répartie sur plusieurs serveurs, une autre fonctionnalité attendue est la gestion des connexions dans un mode multi-instances.

Le paramétrage d'un pool de connexions s'effectuant la plupart du temps au niveau application, chaque instance applique naturellement la formule retenue en termes d'ouverture de connexions. Le serveur d'applications doit à ce niveau répartir les différentes connexions entre les différentes instances.

e) L'ouverture vers l'existant, le respect des standards

En architecture web transactionnelle, la reconnaissance des serveurs HTTP et l'accès aux principales bases de données relationnelles du marché par le serveur d'applications est une condition " sine qua non ". Ces ponts ne se révèlent cependant pas suffisants dans la plupart des contextes, tout d'abord parce que peu d'applications fonctionnent de manière isolée. En effet, les entreprises possèdent généralement un existant auquel il faut s'interfacer. Ainsi, l'ouverture vers les protocoles de communication, les principaux ERP et les mainframes est indispensable pour ne pas limiter fonctionnellement l'application. Ensuite, dans la même optique, l'évolutivité et la pérennité de l'application dépendent en partie de la capacité du serveur d'applications à respecter les standards tels que Java, XML et à intégrer les nouvelles technologies.

f) La gestion de contexte

Aujourd'hui, il n'est pas imaginable de réaliser des sites transactionnels sans mettre à profit les capacités de gestion de contexte. Le principe de gestion de contexte consiste à conserver le temps d'une session les données propres à l'utilisateur contrairement aux variables d'application qui ne sont rien d'autre que des variables globales. Toutefois, les variables de session spécifiques à un utilisateur ne sont envisageables que si l'utilisateur peut être identifié. Une des fonctionnalités de base du serveur d'applications est de gérer automatiquement cette identification, suivant une des trois méthodes : le cookie, l'URL long et la variable cachée. Ceci permet au serveur d'applications de créer un espace mémoire dédié à chaque utilisateur. C'est à ce niveau, généralement dans un objet session, que vont être stockées les informations spécifiques à chacun des utilisateurs.

Un serveur d'applications particulièrement étoffé sera celui qui proposera nativement tout un ensemble de fonctions permettant de gérer le plus finement possible ces sessions utilisateur : définition d'un time out, lancement d'un événement en fin de session, etc.

g) La sécurité

Si la sécurité au niveau HTTP n'est pas une fonctionnalité qui concerne directement le serveur d'applications, ce dernier se doit de faciliter sa mise en place. En effet, il est important de noter que la phase de cryptage/décryptage des pages est particulièrement coûteuse en temps, et peut considérablement pénaliser les performances. Cette problématique de dégradation des performances est donc importante. Pour cela, et étant donné que l'ensemble des pages d'une application n'a pas à être sécurisé (ou exceptionnellement), il est important de pouvoir déployer des applications utilisant à la fois les services sécurisés et également les services non sécurisés. C'est à ce niveau qu'interviendra le serveur d'applications, qui doit permettre de déployer une même application s'adressant à deux ports HTTP différents. Enfin, à ce niveau, l'outil de développement peut également offrir des facilités pour mettre en place une même application à deux niveaux. Dans le cas idéal, les outils de développement possèdent dans les propriétés de chacune des pages HTML dynamiques une case à cocher précisant si la page doit être sécurisée ou non. L'atelier de développement propose également de saisir les ports de chacun des deux modes afin de réduire au maximum cette tâche lors de la phase de déploiement. Au-delà de la gestion de la sécurité par cryptage des informations, les outils doivent également proposer un système d'authentification. Cela passe généralement par une validation côté serveur, avec contrôles effectués dans des annuaires LDAP, des SGBDR, ou toute autre source de données permettant d'identifier les utilisateurs.

h) L'administration

Tout serveur d'applications de bonne facture est livré avec un outil d'administration qui se présente sous la forme d'une interface web ou d'une console. En laissant de côté les fonctionnalités élémentaires de cet outil, il doit favoriser le réglage du serveur d'applications. Ce réglage est nécessaire afin d'adapter et d'ajuster les points sensibles de l'application en cas de montée en charge importante. Dans le cadre d'une architecture multiserveurs, l'outil d'administration doit proposer un paramétrage avancé au niveau du répartiteur de charges comme le choix de l'algorithme de répartition par exemple. Concernant la fonctionnalité de disponibilité applicative au niveau session, là encore l'outil doit fournir une interface proposant différentes solutions de sauvegarde. Au niveau du pooling de connexions, un outil d'administration doit permettre de dimensionner l'accès à la base de données en spécifiant par exemple un nombre de connexions au démarrage, le nombre maximum de connexions ouvertes en tout, etc.

Si l'application repose sur une technologie objet, le déploiement des composants dans le serveur d'applications s'opère également depuis cette interface. Enfin, la présence d'un outil de

statistique est toujours une fonctionnalité intéressante. En effet, la génération de graphes d'utilisation à partir des logs HTTP, SQL ou d'échange entre instances d'objet offre un premier retour sur la sollicitation des différents modules de l'application.

i) La productivité

En phase de développement, la productivité est étroitement liée à la maturité de l'outil de développement, mais aussi à la qualité de l'interface entre ce dernier et le serveur d'applications. Ainsi, l'atelier de développement doit offrir aux développeurs le moyen de réaliser des applications web fiables dans un minimum de temps et d'effort.

Pour ce faire, un atelier de développement incomplet doit au moins pouvoir s'interfacer de la manière la plus transparente possible avec les autres outils de production. Le développement d'une application étant rarement le travail d'une seule personne, une interface avec les outils de gestion des développements en équipe tels que PVCS ou VSS s'imposent. A partir de là, les équipes de développement possèdent une bonne base d'outils pour démarrer leur projet. Pour revenir à l'atelier de développement à proprement parler, une des fonctionnalités couramment utilisées lors de la réalisation d'un site web concerne l'éditeur HTML WYSIWYG qui permet la génération des interfaces graphiques. Sa présence évite l'écriture simple mais fastidieuse du code HTML.

Tout au long de la phase de développement, l'atelier doit fournir des fonctionnalités simplifiant la tâche de l'utilisateur et augmentant la productivité du projet. Ainsi, comme toute application web transactionnelle se connecte forcément à une base de données, une manière de répondre au besoin précédent est d'offrir des assistants capables de se greffer au SGBDR et de permettre la saisie de requêtes dynamiques, la saisie de jointures ou encore la sélection de procédures stockées. De plus, l'intégration d'un éditeur SQL dans l'atelier de développement offre le moyen de tester les requêtes en temps réel.

Enfin, l'outil doit envisager des fonctionnalités avancées en terme de débogage. Effectivement, la possibilité d'effectuer du pas à pas sur le code applicatif, de poser des points d'arrêt, d'entrer ou non à l'intérieur du code des fonctions ou des méthodes exécutées, d'interroger ou de modifier les valeurs des variables en temps réel procure un gain de temps considérable en phase de développement.

2.7 Conclusion :

Dans ce chapitre, on a essayé de fournir une vue générale sur le serveur d'application, de donner un aperçu sur leur évolution, et donner une vue globale sur leur rôle, et leur place au sein des architecture dynamique. Nous avons pris en compte la typologie des serveurs

d'application et leur fonctionnalité au cœur de système. Dans le chapitre suivant, on présentera Les systèmes middlewares, et on donne les notions de base en détaillant les fonctions du middleware, et les besoins aux quels elles répondent.

Chapitre 03 :

Les Middleware 3

Middleware, est une couche logicielle qui est invisible pour l'utilisateur. Il prend deux ou plusieurs applications différentes et les faits travailler ensemble de manière transparente. Ceci est accompli en plaçant middleware entre des couches de logiciel et sur les côtés travaillent les uns avec les autres. Sur cette large définition, middleware pourrait être presque n'importe quel logiciel dans un empilement de couches logiciel. En outre, le middleware est un terme en constante évolution. Comme une grande partie de l'industrie du logiciel est entraînée à travers les perceptions des technologies actuelles les plus importantes, de nombreuses entreprises donnent leur logiciel le nom de "middleware", car il est très populaire.

3.1 Présentation des architectures web :

L'architecture de serveur d'application est basée sur l'architecture 3 tiers (Figure. 3.2), au contraire des architectures client-serveur (Fig. 3.1) qui sont base sur l'architecture de 2 tiers, où le poste client demande une ressource au serveur qui la fournit à partir de ses propres ressources.

a. Architecture à 2 Niveaux

L'architecture à deux niveaux caractérise les systèmes clients/serveurs pour lesquels le client demande une ressource et le serveur la lui fournit directement, en utilisant ses propres ressources.

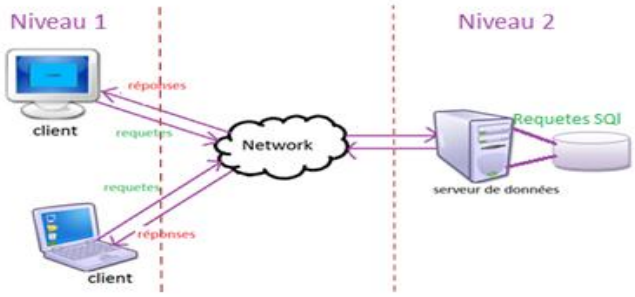


Figure 3.1 - Architecture 2 tiers.

b. Architecture à 3 Niveaux :

Internet a bouleversé les structures et une nouvelle architecture, dite n-tiers (car divisé en parties) a fait son apparition. Ce modèle logique permet la spécialisation de chaque couche : Présentation, application, stockage de l'information pour du 3-tiers.

Cette architecture a l'avantage de bien séparer les domaines au niveau de l'infrastructure. Elle s'appuie sur de nombreux standards et permet une scalabilité (capacité de l'infrastructure à évoluer suivant la montée en charge) importante par des évolutions ciblées des infrastructures supportant tel ou tel domaine.

Cette scalabilité répond à la demande des entreprises ayant des besoins de productivité et de développements toujours plus grands en relation avec la croissance fulgurante des applications Internet au sein des organisations (Transactions, Sécurité, haute disponibilité, CRM, projets Web,…)

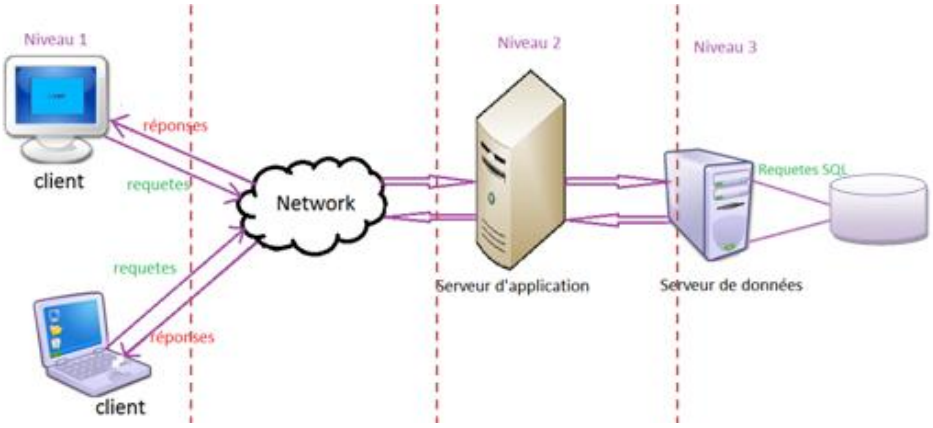


Figure 3.2 - Architecture 3 tiers.

Le serveur d'application devient alors la base d'infrastructure sur laquelle vont être déployées les applications métier. Profitant des technologies Java et du concept des machines virtuelles, le serveur d'application apporte au code applicatif une isolation par rapport à l'infrastructure.

c. Architecture à N Niveaux

Dans un environnement traditionnel de l'architecture multi-niveaux, un serveur d'application fournit des données aux clients, comme une interface entre les clients et les serveurs de base de données. Cette architecture permet d'utiliser un serveur d'applications à :

- Valider les informations d'identification d'un client, comme un navigateur Web
- Se connecter à un serveur de base de données
- Effectuer l'opération demandée

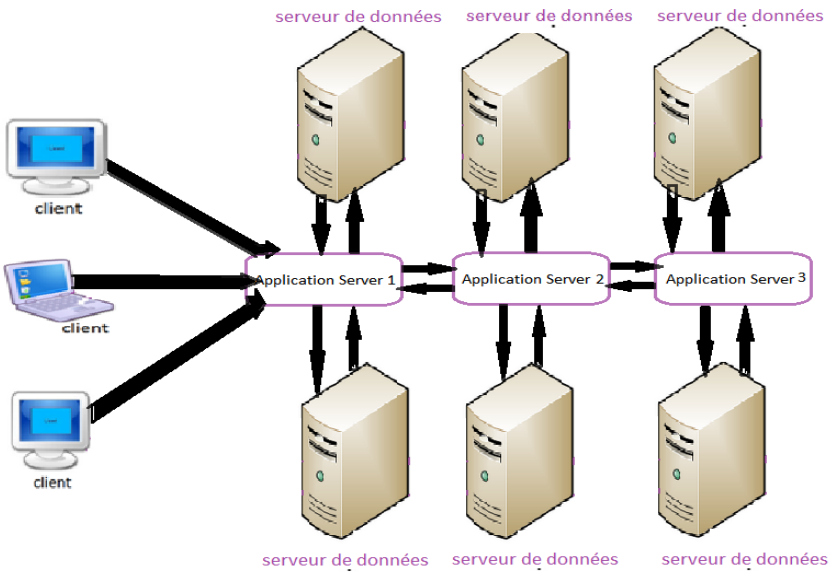


Figure 3.3 - Architecture N tiers.

3.2 Présentation de Middleware :

Un dialogue client/serveur est un dialogue interprocessus appelé (Inter Process Communication).Il permet l'établissement et le maintien de ce dialogue. Il s'est popularisé sous le nom de middleware.

Pour relier une application cliente à un serveur, on s'appuis sur le réseau auquel on a ajouté un ensemble de couches logicielles qui apportent les services de communication nécessaires au dialogue. C'est cet ensemble de couches qui représente le middleware.

Littéralement il désigne « l'élément du milieu ». L'échec de l'architecture à deux (et donc l'absence de middleware) face aux applications a grande échelles, et l'aboutissement à une architecture à trois niveaux dans laquelle le middleware se situe au milieu du client et du serveur, il contient les fonctionnalités intrinsèques de l'application. Il réduit donc les charges du serveur.

Il facilite aussi l'échange entre le client et le serveur :

1. Il prend en compte les requêtes de l'application client.
2. Il les transmet de manière transparente à travers le réseau jusqu'au serveur.
3. Il prend en compte les donnes résultats du serveur vers le client.

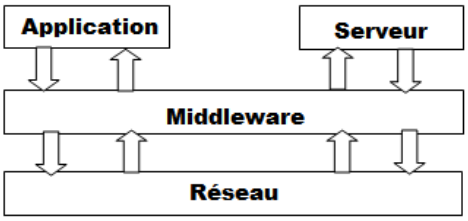


Figure 3.4-Le middleware- la pierre angulaire d'un modèle client/serveur a trois niveaux.

3.3 La Définition de Middleware :

Un middleware est un logiciel servant d'intermédiaire de communication entre plusieurs applications sur un réseau informatique. Ce terme vient de l'anglais middle (milieu) et software (logiciel). Le middleware se situe en dessous de l'applicatif, au dessus du système d'exploitation et entre deux logiciels ayant besoin de communiquer entre eux. Cette classe de logiciels a donc pour but de rendre invisible, à l'utilisation, les spécificités des différentes applications utilisées dans un environnement.

Le middleware permet aux applications de se communiquer les unes avec les autres par le biais de messages, alors qu'elles n'étaient pas conçues jusque-là pour dialoguer ensemble. Il facilite ainsi l'accès à des données stockées dans des systèmes qui ne sont pas toujours compatibles (hétérogènes).

3.4 Le rôle d'un Middleware :

Un middleware gère l'activité d'un ensemble de participants (utilisateurs ou systèmes autonomes). Ces participants peuvent avoir des rôles prédéfinis tels que client (service) et serveur (fournisseur de service), ou diffuseur (émetteur d'information) et abonné (récepteur d'information). Les participants peuvent aussi se trouver au même niveau, chacun pouvant indifféremment assumer un rôle différent, selon les besoins ; une telle organisation est dite peer to peer, ou P2P.

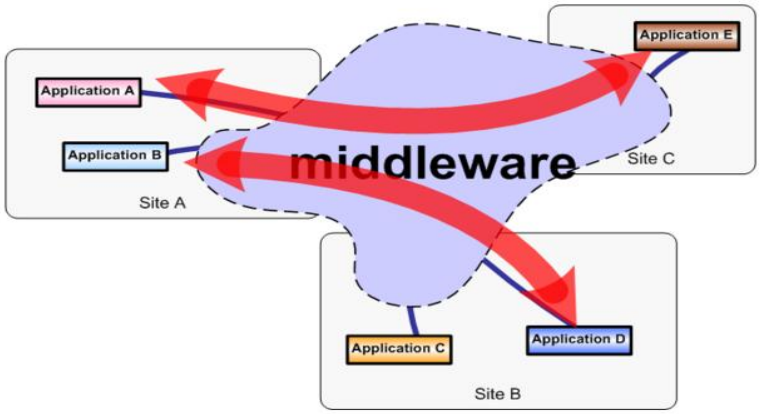


Figure 3.5 – Rôle d'un middleware.

- **Le modèle en H et l'intérêt du Middleware :**

  Le modèle en H montre l'intérêt du middleware. Grace au middleware le réseau est vu comme étant un ensemble de serveurs et de services, toutes les applications peuvent y accéder à travers l'unique point d'accès l'API. A l'intérieur, le middleware peut fournir plusieurs fonctions.

  Le modèle en H est schématisé ci-dessous :

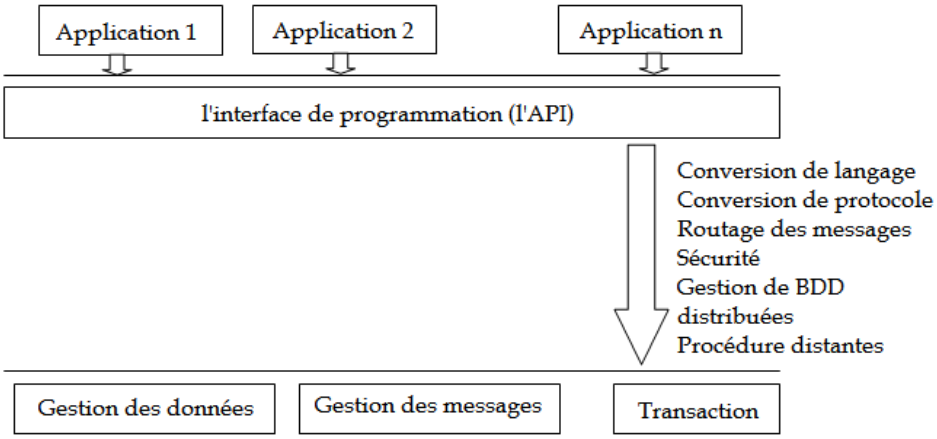


Figure 3.6 Le modèle en H.

D'autres fonctions sont également possibles :

1. L'accès aux données à distance.
2. Gestion des accès concurrents.
3. Procédure de fermeture et d'établissement de connexions.

Pour un type de service, le middleware se charge d'offrir plusieurs fonctions. Par exemple, pour le transactionnel, il se charge de :

1. Traduire la requête.
2. Masquer la localisation des données.
3. Gérer la réalisation de jointure, y compris des sources de données de nature différentes.

3.5 Les différentes catégories de middleware :

Plusieurs types de middlewares existent :

1. **Middleware à message orienté:**

Ce type de middleware est une infrastructure qui prend en charge la réception et l'envoi de messages sur des applications distribuées. Il permet aux applications d'être décaissés sur différentes plateformes et rend le processus de création d'applications logicielles couvrant de nombreux systèmes d'exploitation et protocoles réseau beaucoup moins compliqué.

Il possède de nombreux avantages, et est l'un des types de middleware les plus utilisés.

2. **Middleware Objet**

Également appelé « Object Request Broker », donne aux applications la possibilité d'envoyer des objets et des services de demande par l'intermédiaire d'un système orienté objet. En bref, il gère la communication entre les objets.

3. **Middleware à appel de procédure à distance (Remote Procedure Call (RPC) Middleware)**

Une RPC est exactement ce que cela ressemble. Il appelle les procédures sur des systèmes distants, et est utilisé pour effectuer des interactions synchrones ou asynchrones entre applications ou systèmes. Il est généralement utilisé dans le cadre d'une application logicielle.

4. **Base de données Middleware**

Ce type de middleware permet un accès direct aux bases de données, permettant une interaction directe avec les bases de données. C'est le type le plus général et connue de middleware par exemple (ODBC, JDBC,….).

1.5 Conclusion

Dans ce chapitre, on a vu les middlewares, présentation, définition, rôle, ainsi que les différentes catégories de middleware qui existent.

Les notions étaient importantes pour bien comprendre l'architecture de serveur d'applications choisit JBoss et son architecture. En effet, JBoss est lui même considéré comme un middleware.

Chapitre 04 :

Le Serveur d'applications JBOSS AS 4

Dans ce chapitre, on va étudier le serveur d'application JBoss AS7.
Une étude bibliographe a été réalisée sur sa popularité par rapport aux autres serveurs d'applications. On a étudié son architecture, son arborescence fichier, ainsi que la gestion de la sécurité.

4.1 Historique :

Avec l'apparition du java en 1996, beaucoup de chercheurs ont créé leurs propres infrastructures applicatives à partir du zéro.
En 1999, Marc Fleury a lancé le projet JBoss afin de faire progresser ses intérêts de recherche sur les middlewares. L'organisation du projet était basée sur l'open source dès le début.
La figure 4.1 résume l'historique de l'apparition du JBoss.

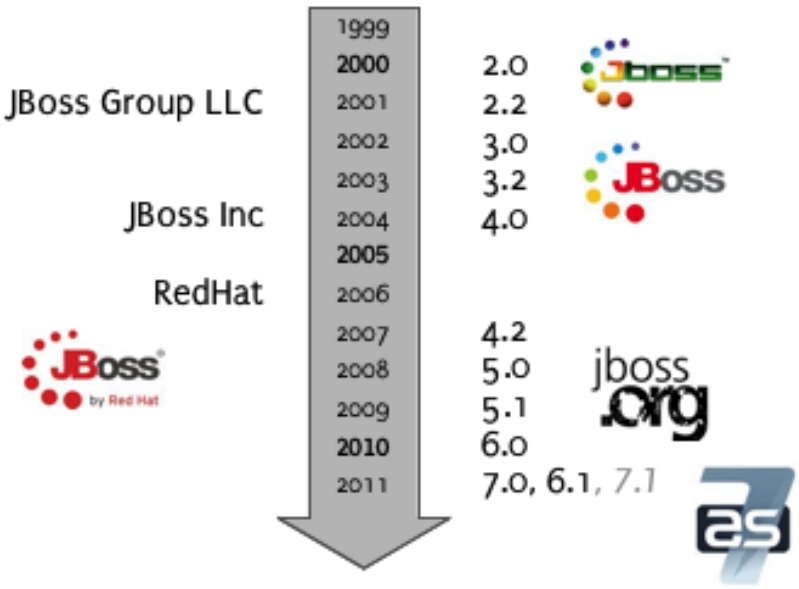


Figure 4.1 – Développement de JBoss AS [11].

Après la publication de la version 2 en 2000, une progression régulière de versions majeures on apparues jusqu'en 2004, cette année-là, la version 4.0 était la première version certifiée de JBoss.

En 2005, Marc Fleury revendique la place n°1 mondial du Middleware, **Hibernate** sort dans sa version 3 le 1er mars et JBoss voit sa première conférence utilisateurs annuelle **Le JBoss World** à Atlanta aux États-Unis.

Hibernate est : est un Framework open source gérant la persistance des objets en base de données relationnelle. Hibernate facilite la persistance et la recherche de données dans une base de données en réalisant lui-même la création des objets et les traitements de remplissage de ceux-ci en accédant à la base de données. La quantité de code ainsi épargnée est très importante d'autant que ce code est généralement fastidieux et redondant.

Hibernate est très populaire notamment à cause de ses bonnes performances et de son ouverture à de nombreuses bases de données.

En avril 2006, RedHat rachète JBoss Inc pour $350 millions de dollars. En mai, JBoss rejoin le W3C (World Wide Web Consortium), l'OASIS (Organization for the Advancement of Structured Information Standards) et le WS-I (Web Services Interoperability).

Red Hat est une société multinationale d'origine américaine éditant des distributions GNU/Linux. Elle est l'une des entreprises dédiées aux logiciels Open Source les plus importantes et les plus reconnues. Elle constitue également le premier distributeur du système d'exploitation GNU/Linux. Red Hat a été fondée en 1993 et son siège social se trouve à Raleigh en Caroline du Nord. Elle possède en plus de ce dernier un nombre important de bureaux dans le monde entier [10].

Puis une version 4.2 en 2007, dont le but était de nous apporter une implémentation partielle de JavaEE 5 et de nous faire patienter avant la version 5.0 qui devait être certifiée JavaEE 5. La version 6 qui a suivi est basée sur la même architecture que la 5, mais implémente JavaEE 6, sans être certifiée. En 2011, la version 7 est sortie, Fondamentalement différente des versions précédentes (JBOSS AS 4,5 et 6), cette nouvelle version présente une nouvelle interface Web d'administration. Aujourd'hui le projet JBoss est à sa version 8 et il a changé de nom pour devenir WildFly. [8]

4.2 La popularité de JBoss AS :

JBoss AS est un produit open source. De plus en plus les entreprises choisissent de JBoss AS en tant que une plate-forme de production de serveur d'application.

La communauté JBoss a lancé la publicité «7 reasons to love JBoss Application Server7» [2].

Figure 4.2 – JBoss Application Server 7 - JBoss Community

Les 7 raisons pour aimer JBoss Application server 7, qui sont :

- **Temps de démarrage rapide.**
- **Léger :** tourne sur des machine de faible puissance de plus un déploiement rapide.
- **Noyau modulaire :** Les modules sont des morceaux de code qui peuvent être chargés et déchargés dans le noyau sur demande. Ils étendent la fonctionnalité du noyau sans avoir besoin de redémarrer JBoss.
- **Le déploiement parallèle :** la possibilité de déployer plusieurs applications au même temps.
- **Administration élégante :** une interface bien organiser et facile à utiliser.
- **Domain management :** assurer la gestion administrative.

4.3 Étude comparatif entre JBoss et Glassfish:

Parmi beaucoup de serveurs d'application, on a choisie de comparer JBoss avec Glassfish, car JBoss et Glassfish les deux plus utilisés en production et les deux serveurs d'applications implémentent la spécification JEE 6.

Glassfish est l'implantation de référence de JEE 6. En ce sens, il implante l'intégralité des fonctionnalités de JEE (Web, EJB...).Glassfish dispose des bonnes fonctionnalités:

- le clustering.
- une interface de configuration Web complète.

- des fonctionnalités de test des Web Services directement intégrées dans la console d'administration Web.

Contrairement à Glassfish, JBoss AS implémenter Java EE 6, sans être certifié. Néanmoins, il implante une grande partie de ses standards. JBoss se décline en deux versions : une version libre (http://www.jboss.org/) et une version entreprise (http://www.redhat.com/jboss/).Depuis sa version 7, JBoss a grandement amélioré son interface Web. Les configurations se faisant dans des fichiers XML. [6].

- **Utilisation**

JBoss est plus mature que Glassfish avec une communauté plus importante. Aussi pour les développeurs, JBoss est clairement loin devant Glassfish. L'intégration aux IDE est similaire à celle de Glassfish mais ce qui le démarque, ce sont tous les projets rattachés à JBoss, Certains de ces projets peuvent être utilisés avec un autre serveur d'application. Concernant l'intégration dans les IDE, Glassfish est nativement intégré à Netbeans et s'intègre facilement à Éclipse, mais JBoss doit s'intégrer dans les deux (JBoss tools). Par défaut, la configuration du plugin permet de redéployer automatiquement une application sur le serveur à chaque modification du code source pour les deux serveurs.

- **Tests de performances :**

Le tableau suivant montre une comparaison entre les deux serveurs d'applications JBoss as et Glassfish [6] :

Serveur d'applications	**Durée de démarrage**		**Durée de déploiement**	**Consommation du mémoire**	
	à vide	**avec une application**	**Avec application web**	**à vide**	**avec une application**
JBoss AS	**2.911 s**	**17.165 s**	**3.324 s**	**120.7 Mo**	**310 Mo**
Glassfish	**3.626 s**	**17.165 s**	**15.625 s**	**264.3 Mo**	**496 Mo**

Figure 4.3- teste de performance ente JBoss AS et Glassfish.

On remarque que Glassfish est indéniablement plus gourmand en termes de mémoire et de temps de déploiement. Glassfish est plus simple à prendre en main de part son interface web. Du point de vue des fonctionnalités de base, on dirait que les deux serveurs se valent. Glassfish étant plus jeune, et n'a pas fini d'évoluer.

JBoss ne se résume pas à un serveur d'application mais aussi à une multitude de projets qui viennent se greffer; ce qui peut expliquer sa large adoption par les entreprises.

Pour conclure on dirait que les deux serveurs sont excellent mais que Glassfish est plus adapté à un débutant. Néanmoins JBoss reste le plus utilisé et le plus populaire à cause de sa rapidité et sa faible demande en ressource. [6].

4.4 Installation :

JBoss Application Server 7 peut être obtenus à partir de site officiel de JBoss.

La version choisi dans notre travaille est JBoss Application Server 7.1.1.final, il y aura deux distributions disponibles pour le téléchargement dans les formats de fichier zip ou tar. Pour installer JBoss Il suffit d'extraire le téléchargement choisi dans un répertoire. On peut installer JBoss Application Server 7 sur n'importe quel système d'exploitation qui prend en charge les formats zip ou tar. Dans notre cas on à installé JBoss as7 sous linux (ubuntu14.04).

Pour installer JBoss il faut :

1. Installation du JDK.
2. Défini les Variables d'environnements JBOSS_HOME et JAVA_HOME.

   Pour défini les deux variables d'environnement JAVA_HOME et JBOSS_HOME on doit accéder au fichier .bashrc qui se trouve dans le chemin /home/USR/.bashrc comme illustré dans la figue 4.4.

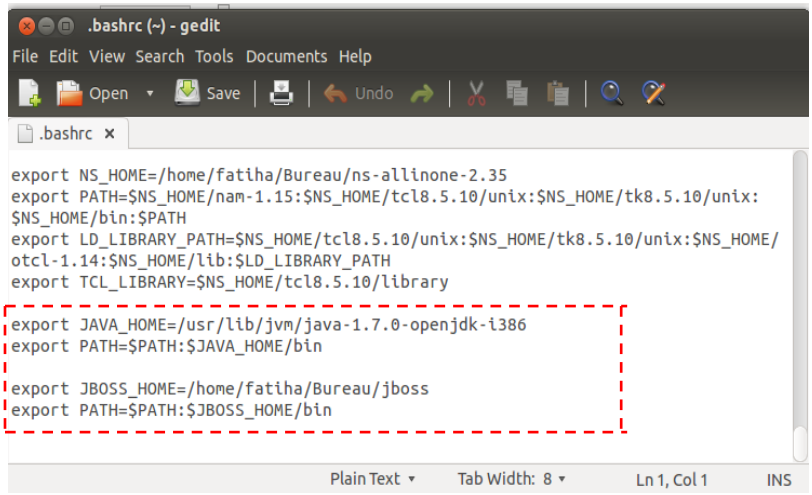


Figure 4.4– Le fichier .bashrc.

3. Dézipper l'archive JBoss avec la commande : « unzip jboss-as-7.1.1.final.zip »

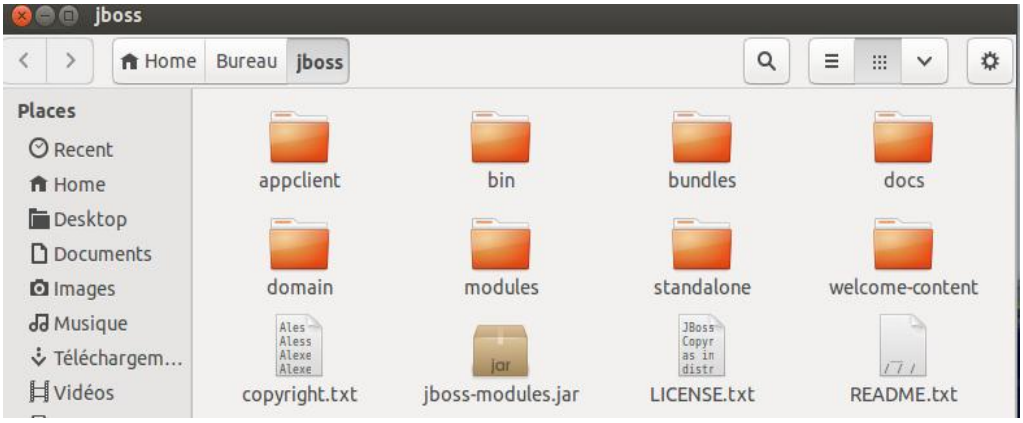


Figure 4.5 – Le répertoire JBoss.

4.5 Arborescence Jboss7:

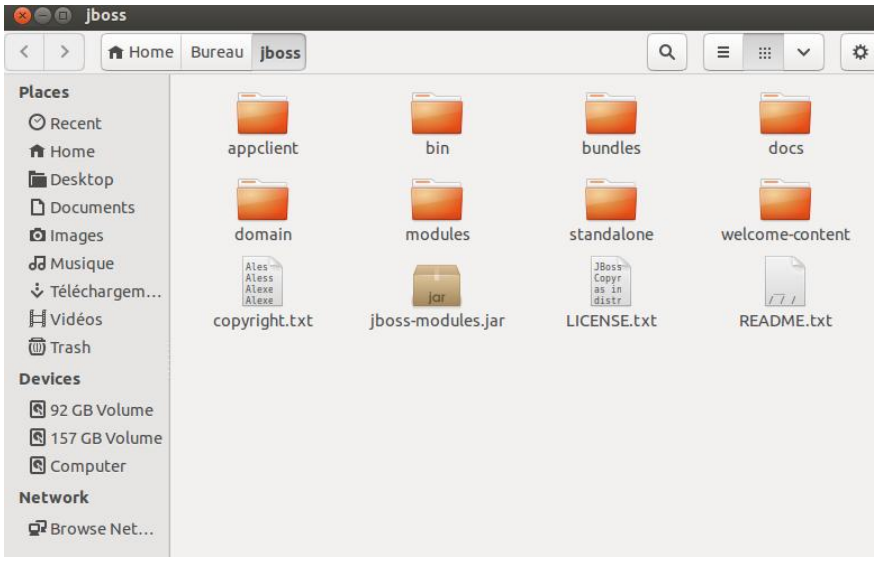


Figure 4.6 – arborescence-jboss-7-1-1final.

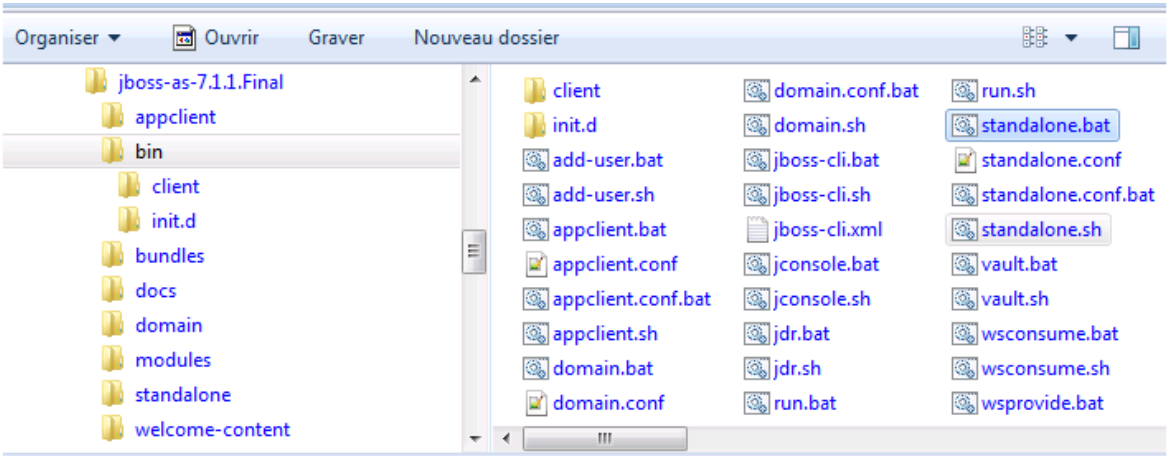
Dans cette arborescence on trouve plusieurs répertoires, dont les plus importants sont: Le répertoire bin, le répertoire standalone et le répertoire domain.

a) Répertoire bin :

Le répertoire qui nous intéressera le plus important est le répertoire 'bin', dans lequel se trouvent :

1. Scripts et fichier de configurations pour le lancement en mode 'standalone' **(**standalone.bat et standalone.conf.bat / standalone.sh et standalone.conf) équivalent du run.bat des précédentes versions.
2. Scripts et fichier de configurations pour le lancement en mode 'domain' (domain.bat et domain.conf.bat / domain.sh et domain.conf).

3. Scripts et fichier de configuration (jboss-cli.xml) lancement du nouveau mode interactif (jboss-cli.bat / jboss-cli.sh).
4. les scripts de génération du WSDL web services (wsconsume.bat / wsconsume.sh et des interfaces java associées (wsprovide.bat/ wsprovide.sh). WSDL est une grammaire XML permettant de décrire un service web.



. Figure 4.7- structure de répertoire bin.

On trouve également les deux scripts liés à la sécurité de JBoss AS 7 :

5. script add-user.bat/add-user.sh : pour ajouter des utilisateurs dans les fichiers de propriétés.
6. script vault.bat/vault.sh : pour sécuriser un mot de passe (cryptage).

   Enfin, un script dédié au diagnostic :
7. Script jdr.bat / jdr.sh (rapport de diagnostique de JBoss).

b) Répertoire standalone :

Dans JBoss 7, le démarrage dit 'standalone' est géré dans le répertoire 'standalone'. Cela représente le lancement d'une instance de JBoss 'isolé.

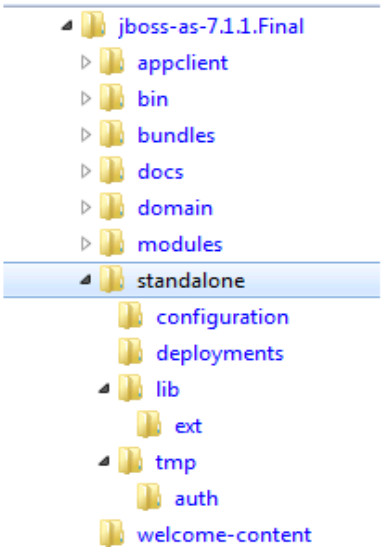


Figure 4.8- structure de répertoire standalone.

1. le sous-répertoire 'déploiements' est dédié à la réception des livrables (war, ear, jar, rar, sar).
2. le sous-répertoire 'configuration' est dédié à la configuration du serveur en mode standalone. c'est le fichier de configuration standalone.xml qui est utilisé par défaut.

c) Répertoire 'domain'

Dans JBoss AS 7, le démarrage dit domain est géré dans le répertoire 'domain'. Un domaine JBoss 7 est utilisé pour gérer et coordonner un ensemble d'instances JBoss 7. Ces instances sont gérées par un contrôleur de domaine.

Un contrôleur de domaine : est une instance de serveur de JBoss qui agit en tant que point central de gestion pour un domaine. Une instance de contrôleur d'hôte est configurée pour agir en tant que contrôleur de domaine. Les responsabilités principales d'un contrôleur de gestion sont :

- maintenir la politique centrale de gestion du domaine.
- s'assurer que tous les contrôleurs soient mis au courant de leurs contenus actuels.
- assister tous les contrôleurs pour que toutes les instances en cours de JBoss EAP 6 soient configurées suivant cette politique.

  La politique de gestion centrale est stockée par défaut dans le fichier domain/configuration/domain.xml.

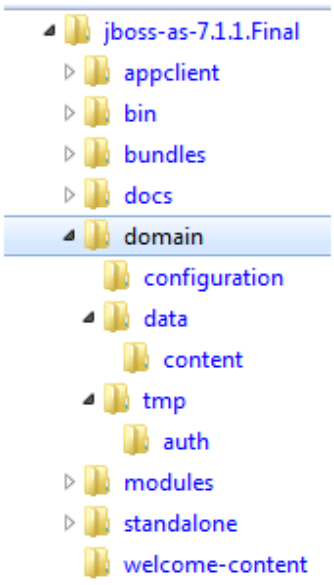


Figure 4.9- structure de répertoire Domain.

Le répertoire data\content est dédié aux livrables (war, ear,...).

4.6 Démarrage :

4.6.1 Démarrage en mode standalone :

Pour démarrer JBoss en mode standalone on se localise dans le répertoire bin :

Par la commande : cd jboss-as-7.1.1.final/bin.

Puis on lancer le script ./standalone.sh.

```
fatiha@fatiha-HP-Pavilion-g6-Notebook-PC: ~/Bureau/jboss/bin

fatiha@fatiha-HP-Pavilion-g6-Notebook-PC:~/Bureau/jboss/bin$ ./standalone.sh
=========================================================================

  JBoss Bootstrap Environment

  JBOSS_HOME: /home/fatiha/Bureau/jboss

  JAVA: /usr/lib/jvm/java-7-oracle/bin/java

  JAVA_OPTS:  -server -XX:+TieredCompilation -Xms64m -Xmx512m -XX:MaxPermSize=256m
 -Djava.net.preferIPv4Stack=true -Dorg.jboss.resolver.warning=true -Dsun.rmi.dgc.c
lient.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000 -Djboss.modules.s
ystem.pkgs=org.jboss.byteman -Djava.awt.headless=true -Djboss.server.default.confi
g=standalone.xml

=========================================================================

01:31:49,033 INFOS [org.jboss.modules] JBoss Modules version 1.1.1.GA
01:31:49,235 INFO  [org.jboss.msc] JBoss MSC version 1.0.2.GA
01:31:49,292 INFO  [org.jboss.as] JBAS015899: JBoss AS 7.1.1.Final "Brontes" start
ing
01:31:50,463 INFO  [org.xnio] XNIO Version 3.0.3.GA
01:31:50,469 INFO  [org.jboss.as.server] JBAS015888: Creating http management serv
ice using socket-binding (management-http)
01:31:50,486 INFO  [org.xnio.nio] XNIO NIO Implementation Version 3.0.3.GA
01:31:50,496 INFO  [org.jboss.remoting] JBoss Remoting version 3.2.3.GA
01:31:50,581 INFO  [org.jboss.as.logging] JBAS011502: Removing bootstrap log handl
ers
01:31:50,585 INFO  [org.jboss.as.configadmin] (ServerService Thread Pool -- 26) JB
AS016200: Activating ConfigAdmin Subsystem
01:31:50,630 INFO  [org.jboss.as.clustering.infinispan] (ServerService Thread Pool
 -- 31) JBAS010280: Activating Infinispan subsystem.
01:31:50,697 INFO  [org.jboss.as.connector] (MSC service thread 1-4) JBAS010408: S
tarting JCA Subsystem (JBoss IronJacamar 1.0.9.Final)
01:31:50,707 INFO  [org.jboss.as.naming] (ServerService Thread Pool -- 38) JBAS011
800: Activating Naming Subsystem
01:31:50,713 INFO  [org.jboss.as.connector.subsystems.datasources] (ServerService
Thread Pool -- 27) JBAS010403: Deploying JDBC-compliant driver class org.h2.Driver
```

Figure 4.10- Démarrage en mode standalone.

4.6.2 Démarrage en mode domain:

Pour démarrer JBoss en mode domain on se localise dans le répertoire bin :

Par la commande : cd jboss-as-7.1.1.final/bin.

Puis on lancer le script ./domain.sh.

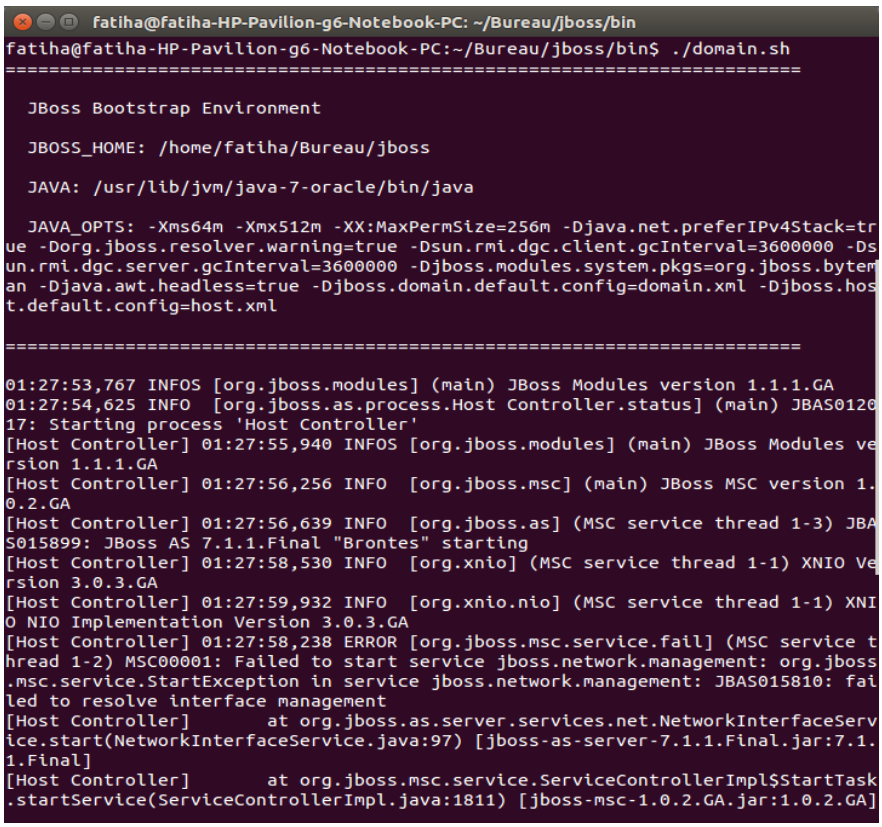


Figure 4.11- Démarrage en mode domaine.

4.7 Architecture générale de JBoss AS

Depuis sa naissance, l'architecture de JBoss AS demeure très complexe et difficile à appréhender. Un important niveau d'abstraction a été mis en place dés le départ par les concepteurs, et plus particulièrement grâce à l'intégration de nombreux concepts dont l'AOP (Aspect Oriented Programming) par exemple.

Cette conception permet de diminuer de manière significative les dépendances entre les composants, mais complique également fortement la compréhension du fonctionnement interne de JBoss. Donc L'architecture JBoss est spécial par rapport aux autres serveurs d'applications J2EE architecture.

L'AOP : est un paradigme de programmation qui permet de traiter séparément les préoccupations transversales, qui relèvent souvent de la technique, des préoccupations métier, qui constituent le cœur d'une application. Un exemple classique d'utilisation est la

journalisation, mais certains principes architecturaux ou modèles de conception peuvent être implémentés à l'aide de ce paradigme de programmation, comme l'inversion de contrôle [9].

4.3.1 JBoss AS 4 et antérieur : Le JMX Microkernel.

JBoss AS utilise le concept de composants. Un composant (Component) au sens JBoss correspond à une entité représentant une ressource au sens large du terme. Une ressource peut donc correspondre aussi bien à un périphérique qu'à une partie du cœur de JBoss AS, signifiant que leur gestion sera strictement identique quelle que soit la nature de la ressource gérée. Pour connecter ces composants entre eux et permettre les interactions extérieures avec ces mêmes composants, JBoss intègre l'implémentation de la Java Management Extension (JMX) qui constitue en quelque sorte la colonne vertébrale de JBoss AS (Figure 4.12).

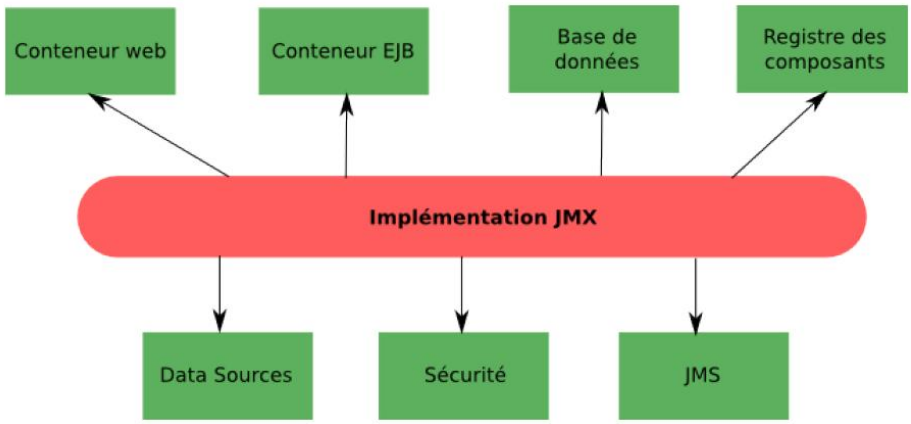


Figure 4.12 – Connexion des composants via la JMX. [5]

Via la JMX, il est possible de gérer l'ensemble des composants de JBoss AS.
Parmi les composants principaux présents par défaut, nous trouvons :

- **un conteneur web** : serveur Tomcat utilise pour l'hébergement d'applications web (interface d'administration, application utilisateur, etc.). Ce conteneur permet d'utiliser JBoss AS comme frontal.
- **un conteneur EJB** : serveur utilisé pour l'exécution d'EJB et principalement utilisé lorsque JBoss AS est utilisé comme middleware au sein de l'architecture applicative.

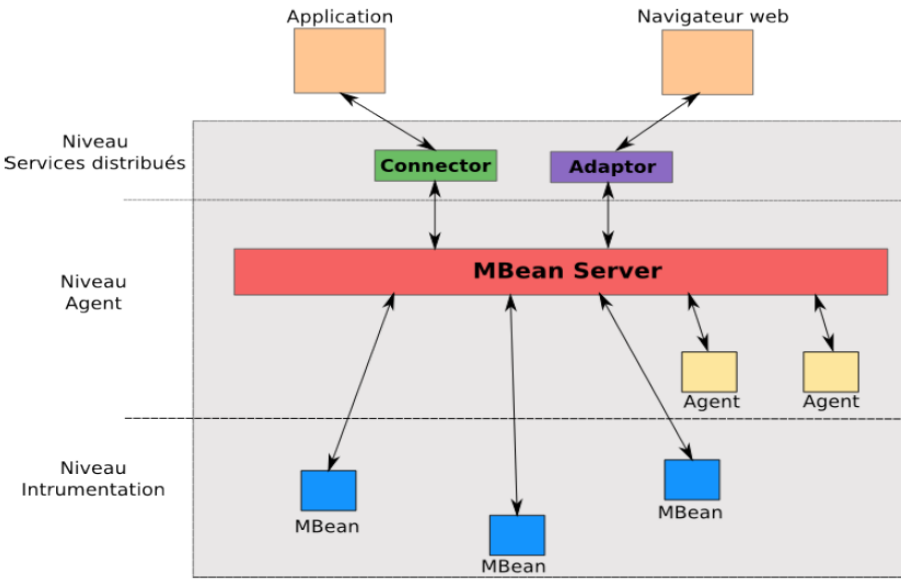


Figure 4.13 –Architecture JMX au sein de JBoss AS. [5]

L'architecture JMX peut être décomposée en trois couches (que la figure 4.13) illustre.
Le niveau Instrumentation est la couche la plus basse mais aussi la plus importante de l'implémentation JMX puisqu'elle correspond à l'ensemble des composants gérés par JBoss (applicatif, périphériques, modules d'authentiquassions . . .). Plus techniquement parlant, il s'agit de classes Java appelées Java Managed Beans (ou MBeans) qui implémentent une interface permettant de gérer la ressource qu'elles représentent. La principale fonction de ce niveau est donc de s'abstraire totalement de la nature de la ressource sous-jacente et de la manipuler via un simple objet Java.

Au dessus de cette couche se trouve le niveau Agent qui, par l'intermédiaire du JMX Microkernel, constitué du MBean Server et d'Agents, va gérer la communication entre le niveau supérieur et l'ensemble des composants du niveau Instrumentation. Pour manipuler un quelconque composant au sein de JBoss, il est donc obligatoire de passer par le JMX Microkernel, et plus particulièrement par le MBean Server qui joue le rôle de contrôleur au sein de l'architecture. Le niveau Services Distribués permet, lui, de faire communiquer les applications tierces (applications utilisateurs, interfaces d'administration ...) avec le MBean Server à l'aide d'objets Java appelés Connectors et Adaptors. Il s'agit d'éléments permettant d'interfacer un protocole arbitraire (HTTP, RMI, SOAP, IIOP ...) avec l'implémentation JMX. Ils garantissent notamment l'invocation distante d'objets Java quel que soit le protocole utilisé. Du point de vue d'un attaquant, le moindre accès au MBean Server suffit pour contrôler totalement l'ensemble des MBeans sous-jacents et par conséquent le serveur JBoss lui-même. Concernant les moyens d'accès, JBoss nous fournit déjà tous les outils nécessaires via les Connectors et les Adaptors avec lesquels il est nécessaire de communiquer (figure 4.14).

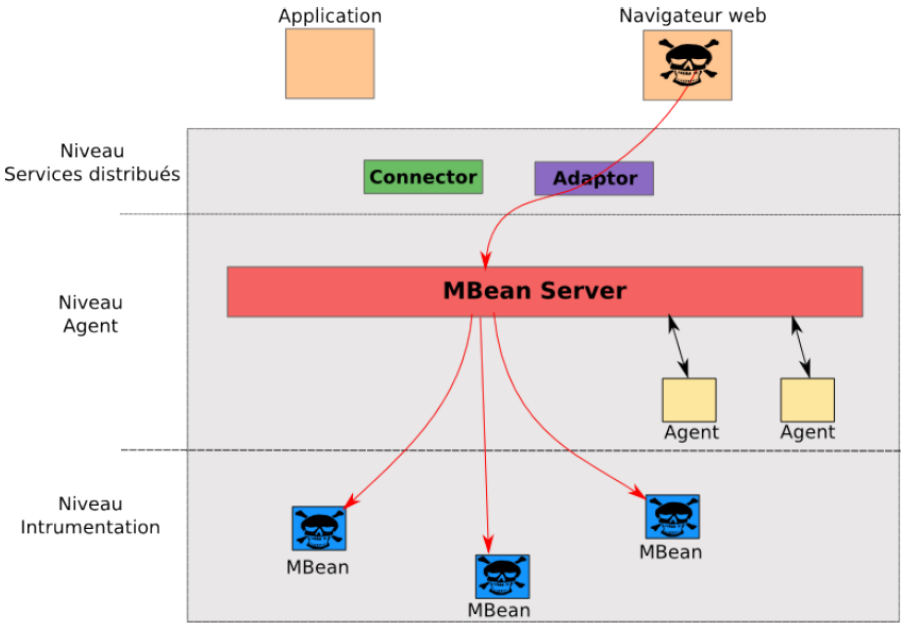


Figure 4.14 – Schéma d'attaque sur un JBoss AS. [5]

4.3.2 JBoss AS 5 et supérieur : un modèle orienté POJO

La version 5 de JBoss AS apporte des différences notables au niveau du modèle et de l'architecture interne du serveur. La principale évolution concerne notamment le JMX Microkernel qui a été progressivement remplacé par un élément déjà prestent dans les versions antérieures mais qui restait secondaire : *le Microcontainer*. Cet élément apporte encore plus de modularité et d'abstraction en intégrant la notion de POJO (Plain Old Java Object) au serveur JBoss AS.

Par définition, un POJO est un objet n'implémentant pas d'interface spécifique à un framework [12].

L'idée est née lorsqu'il a semblé évident que la principale raison pour laquelle les développeurs préféraient faire appel à des objets EJB, plutôt que de travailler avec des objets Java ordinaires mais tout aussi efficaces, était que cette dernière méthode ne permettre simplement pas de se vanter d'avoir utilisé un acronyme.

Le terme est devenu très en vogue, et ne vise plus aujourd'hui directement le framework EJB, mais tous les frameworks compliqués ou spécialisés qui enferment les développeurs dans leurs usages, contrairement à un usage extensif de la bibliothèque standard de Java.

La nouvelle architecture de Microcontainer a plusieurs avantages sur l'ancien JMX kernel architecture. Tout d'abord, il est beaucoup plus léger car il n'a pas à supporter JMX, permettant de construire encore plus petites configurations minimales. Deuxièmement, les services intégrés sur le dessus de la Microcontainer peuvent être déployés autonome

(standalone), dans un autre serveur d'application tel que BEA WebLogic Server, ou même à l'intérieur d'un serveur web comme Tomcat. Par exemple, le conteneur EJB3 peut être déployé sur le serveur web Tomcat parce que le conteneur EJB3 est basé sur la Microcontainer.

BEA WebLogic est une plateforme de serveur d'application basée sur des standards ouverts, extensibles pour l'assemblage, le déploiement, et la gestion d'applications distribuées. Avec WebLogic, vous délivrez plus rapidement des applications de réseau, plus rapides, plus fiables avec tous les bénéfices de Java. Le serveur et conteneur Enterprise JavaBeans puissant de WebLogic fournit une fonctionnalité, flexibilité et performance leader dans l'industrie.

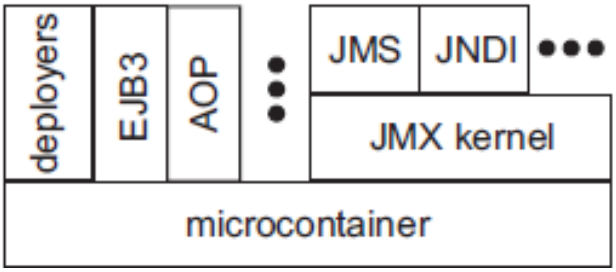


Figure 4.15 – services qui s'appuient sur la Microcontainer.

Parce que tous les services ont été portés à la Microcontainer le noyau JMX joue encore un rôle majeur dans l'architecture, comme illustré à la figure 4.15.

Le noyau de JMX est l'un des POJO primaires définis, et créé par le Microcontainer.

Finalement, dans une version future, JBoss AS sera entièrement basé Microcontainer et ont une architecture similaire à celle de la figure 4.16. JMX passe d'être l'architecture sous-jacente du serveur d'application à l'un des services déployés dans la Microcontainer pour gérer et surveiller les composants.

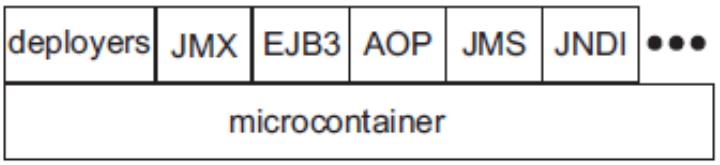


Figure 4.16 – un schéma montrant les services appuyer sur la Microcontainer.

4.8 Modèle de sécurité de JBoss AS

Les problématiques de sécurisation n'ont pas échappé aux concepteurs de JBoss AS et ont intégré dés son lancement un bon nombre de mécanismes de sécurité allant de la gestion des autorisations à la mise en place de sandboxes pour les différents composants hébergés. L'objectif de ces dispositifs est clairement d'éviter tout accès illégitimes à une

ressource particulière mais également d'empêcher un composant de réaliser des opérations malveillantes.

un sandbox est un mécanisme qui permet l'exécution de logiciel(s) avec moins de risques pour le système d'exploitation. Ces derniers sont souvent utilisés pour exécuter du code non testé ou de provenance douteuse.

4.9 Gestion des autorisations : JBossSX

Afin d'assurer la gestion des autorisations pour les accès aux composants, JBoss AS intègre *la JBoss Security Extension (JBossSX)*. Cette extension est par défaut, basée sur l'API JAAS (*Java Authentification and Autorisation Service*) fournissant une indépendance entre l'applicatif et la sécurité. Cette API étant relativement complexe, les notions importantes sont :

- **les domaines de sécurité (Security Domain).**

Pour visualiser les domaines de sécurité présents par défaut sur JBoss AS, il suffit de consulter le fichier JBOSS_SERVER/conf/login-config.xml dont voici un aperçu pour le domaine de sécurité jmx-console :

```
<application-policy name="jmx-console">
<authentication>
<login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule" flag="required">
<module-option name="usersProperties">props/jmx-console-users.properties</module-
option>
<module-option name="rolesProperties">props/jmx-console-roles.properties</module-
option>
</login-module>
</authentication>
</application-policy>
```

Ce domaine utilise par exemple le Login Module correspondant à la classe Java org.jboss.security.auth.spi.UsersRolesLoginModule, fournissant une authentification par login/mot de passe avec une gestion des droits d'accès à l'aide de rôles. Les comptes et les rôles se trouvent respectivement dans les fichiers dédiés jmx-console-users.properties et jmx-console-roles.properties.

- **les rôles :**

Pour leur part, sont associés à un domaine de sécurité et permettent de spécifier qui posséde les droits d'accès aux composants auxquels s'appliquent le domaine. Si l'utilisateur ou l'objet authentifié possède un rôle définit dans le domaine de sécurité appliqué, alors celui-ci pourra accéder aux composants associés sinon l'accès lui sera refusé (même si l'authentification a réussi).

- **les Principals (les identifiants entre autres) :**

Peuvent être globalement considérés comme des identifiants permettant d'identifier de manière unique un utilisateur ou un objet aux vues de son authentification. Ces Principals sont ensuite à des rôles afin de leur attribuer des droits d'accès.

- **les Credentials (mot de passe, certificat ...) :**

Les Credentials permettent de prouver que l'utilisateur ou l'objet demandant les accès est bien celui qu'il prétend être. Les Credentials peuvent être des mots de passe ou des certificats par exemple mais plus généralement des objets Java.
Concernant le stockage de ces éléments, JBoss permet de multiples supports :
base de données, fichiers texte, base LDAP, etc.

4.9.1 Intégration de la sécurité Java 2 :

Etant donné que le risque ne vient pas forcement des utilisateurs mais peut aussi venir des composants hébergés, JBoss AS intègre également l'ensemble des mécanismes de sécurité de Java 2 et notamment le Security Manager. Le premier objectif de ce mécanisme est de *sandboxer* des composants, c'est-adire de restreindre les privilèges d'un code Java lors de son exécution afin de prévenir d'éventuelles opérations à risque (accès au système d'exploitation, ouverture de sockets, lancement de processus, création de class loader, etc.). Cette protection permet d'empêcher toute atteinte à l'intégrité et à l'éventuelle confidentialité des données manipulées par le serveur. Cela est particulièrement utile dans le cas d'une porte dérobée qui aurait été insérée au sein du serveur. Concernant la configuration du Security Manager, celle-ci est généralement unique suivant le système et les applications mises en place.
La politique de sécurité à appliquer est donc propre à chaque serveur.

4.9.2 JBoss AS et sa configuration par défaut :

Un rapide coup d'œil au sein de la configuration de JBoss AS suffit pour trouver son principal défaut : il n'active aucun mécanisme de sécurité par défaut. Cela signifie que malgré une déclaration effective des différents domaines de sécurité dans la configuration de JBoss AS, ceux-ci ne sont rattachés à rien, ouvrant ainsi toutes les portes aux éventuels attaquants. Dans la pratique, il faut également noter que bon nombre d'administrateurs n'activent pas ces sécurités même les plus simples à mettre en place, soit par négligence soit par leur incompréhension des mécanismes de sécurité. Lorsque quand bien même certaines sécurités sont mises en place, les mots de passe par défaut sont généralement laissés (à savoir admin/admin).

4.10 JBoss AS du point de vue de l'attaquant :

À la vue de la configuration par défaut des serveurs JBoss en terme de sécurité, il est facile de comprendre l'intérêt que portent les pirates à ces serveurs, quelles fonctionnalités si intéressantes ces serveurs apportent-ils, Et bien en réalité, elles peuvent être de plusieurs ordres allant de l'obtention d'information sur le serveur lui-même à l'exécution de code arbitraire en passant par la modification à la volée de la configuration du serveur ; en d'autres termes : une vrai mine d'or pour les pirates. La seule obligation est d'avoir un accès aux MBeans fournissant ces fonctionnalités et donc plus généralement au MBean Server.
Pour cette partie, nous allons nous placer du point de vue d'un attaquant étant parvenu, à prendre le contrôle du MBean Server de JBoss AS et désirant réaliser des opérations malveillantes.

4.10.1 Récupération d'informations :

Avant d'intruse un système quel qu'il soit, un attaquant passe nécessairement par une phase de reconnaissance visant à identifier de manière précise sa cible afin de détecter d'éventuelles vulnérabilités exploitables. Pour cela, JBoss AS propose par défaut un certain nombre de MBeans donnant tous les outils à l'attaquant pour réaliser cette opération. Il peut notamment obtenir une description complète de l'arborescence du serveur JBoss par l'intermédiaire du MBean prévu à cet effet jboss.system:type ServerConfig (emplacement des répertoires de déploiement, des fichiers de journalisation, des répertoires temporaires, des fichiers de configuration, etc.). La figure 4.17 donne un aperçu des attributs qu'il est possible de récupérer par l'intermédiaire de ce MBean.

Name	Type	Access	Value	Description
ServerDataDir	java.io.File	R	/opt/jboss-4.2.0.GA/server/default/data	MBean Attribute.
ExitOnShutdown	boolean	RW	(•) True ( ) False	MBean Attribute.
ServerLogDir	java.io.File	R	/opt/jboss-4.2.0.GA/server/default/log	MBean Attribute.
HomeURL	java.net.URL	R	file:/opt/jboss-4.2.0.GA/	MBean Attribute.
ServerConfigURL	java.net.URL	R	file:/opt/jboss-4.2.0.GA/server/default/conf/	MBean Attribute.
ServerTempDeployDir	java.io.File	R	/opt/jboss-4.2.0.GA/server/default/tmp/deploy	MBean Attribute.
RequireJBossURLStreamHandlerFactory	boolean	RW	(•) True ( ) False	MBean Attribute.
PlatformMBeanServer	boolean	R	False	MBean Attribute.
ServerNativeDir	java.io.File	R	/opt/jboss-4.2.0.GA/server/default/tmp/native	MBean Attribute.
ServerTempDir	java.io.File	R	/opt/jboss-4.2.0.GA/server/default/tmp	MBean Attribute.
ServerName	java.lang.String	R	default	MBean Attribute.
ServerHomeURL	java.net.URL	R	file:/opt/jboss-4.2.0.GA/server/default/	MBean Attribute.
RootDeploymentFilename	java.lang.String	RW	jboss-service.xml	MBean Attribute.
PatchURL	java.net.URL	R	null	MBean Attribute.
BlockingShutdown	boolean	RW	( ) True (•) False	MBean Attribute.
SpecificationVersion	java.lang.String	R	4.2.0.GA	MBean Attribute.
ServerHomeDir	java.io.File	R	/opt/jboss-4.2.0.GA/server/default	MBean Attribute.
ServerLibraryURL	java.net.URL	R	file:/opt/jboss-4.2.0.GA/server/default/lib/	MBean Attribute.
ServerBaseDir	java.io.File	R	/opt/jboss-4.2.0.GA/server	MBean Attribute.
ServerBaseURL	java.net.URL	R	file:/opt/jboss-4.2.0.GA/server/	MBean Attribute.
LibraryURL	java.net.URL	R	file:/opt/jboss-4.2.0.GA/lib/	MBean Attribute.
HomeDir	java.io.File	R	/opt/jboss-4.2.0.GA	MBean Attribute.

Figure 4.17- Attributs du MBean jboss.system :type=ServerConfig. [5]

L'attaquant peut déduire de ces informations, le système d'exploitation sous-jacent, la version du serveur JBoss, les répertoires potentiellement exploitables pour l'écriture de fichiers et les éventuelles vulnérabilités propres à la version du serveur JBoss. Pour obtenir des informations encore plus précises sur le serveur (branche SVN, date de compilation . . .), l'attaquant peut également interroger le MBean jboss.system:type=Server.
Le MBean jboss.system:type=ServerInfo, quant à lui, fournit une description très précise de l'OS (version du système, architecture 32 ou 64 bits, nombre de processeurs, mémoire disponible, etc.) et de la machine virtuelle Java (version, éditeur, etc.). De ces informations, l'attaquant peut également déduire d'éventuelles vulnérabilités au sein du système d'exploitation en vue d'une future élévation de privilèges par exemple (cf. figure 4.18).

Name	Type	Access	Value	Description
ActiveThreadCount	java.lang.Integer	R	48	MBean Attribute.
AvailableProcessors	java.lang.Integer	R	1	MBean Attribute.
OSArch	java.lang.String	R	i386	MBean Attribute.
MaxMemory	java.lang.Long	R	532742144	MBean Attribute.
HostAddress	java.lang.String	R	127.0.0.1	MBean Attribute.
JavaVersion	java.lang.String	R	1.6.0_0	MBean Attribute.
OSVersion	java.lang.String	R	2.6.24-19-generic	MBean Attribute.
JavaVendor	java.lang.String	R	Sun Microsystems Inc.	MBean Attribute.
TotalMemory	java.lang.Long	R	133365760	MBean Attribute.
ActiveThreadGroupCount	java.lang.Integer	R	7	MBean Attribute.
OSName	java.lang.String	R	Linux	MBean Attribute.
FreeMemory	java.lang.Long	R	105237600	MBean Attribute.
HostName	java.lang.String	R	vmtomcat	MBean Attribute.
JavaVMVersion	java.lang.String	R	1.6.0_0-b11	MBean Attribute.
JavaVMVendor	java.lang.String	R	Sun Microsystems Inc.	MBean Attribute.
JavaVMName	java.lang.String	R	OpenJDK Client VM	MBean Attribute.

Figure 4.18- Attributs du MBean jboss.system :type=ServerInfo. [5]

4.10.2 Déni de service :

Une fois en interaction avec le MBean Server, un attaquant ne s'arrête généralement pas à une simple phase de reconnaissance et peut commencer à mener des actions bien plus néfastes. L'un des objectifs peut notamment être la mise hors service du serveur dans le cadre d'un « contrat » avec un concurrent de la victime par exemple. Pour mener une telle attaque, il faut savoir que JBoss AS permet, à l'aide du MBean jboss.system:type=Server, d'arrêter purement et simplement le serveur à distance par l'invocation des méthodes shutdown(), halt() ou exit() du MBean, nécessitant un redémarrage « à la main » du serveur.
Une autre possibilité, qui se rapproche de celle offerte par les serveurs Tomcat, est d'arrêter les applications et composants sensibles du serveur (applications et composants d'administration, points d'entrée au serveur, applicatif métier, etc.). Pour

cela, JBoss fournit une méthode générique stop() valable sur la quasi-totalité des MBeans permettant l'arrêt du MBean sur lequel elle est appliquée. Cette méthode est beaucoup plus vicieuse car elle n'arrête pas le serveur en temps que tel et il sera nécessaire d'analyser les différents logs, souvent peu précis sur l'origine du problème.
La façon la plus simple et la plus rapide pour les administrateurs sera, de nouveau, le redémarrage « à la main » du JBoss AS s'ils ne veulent pas passer plusieurs heures à chercher quel composant il est nécessaire de redémarrer.
Ces possibilités s'adressent généralement à des script kiddies car la simple manipulation du serveur JBoss chez soi suffit à les découvrir.

4.11 Conclusion :

Dans ce chapitre nous avons présenté le serveur d'application JBoss as7, commençons par une brève historique, et donné les raison de popularité de JBoss, nous avons parlé de l'architecture générale et le modèle de sécurité de JBoss, puis nous avons faire une étude comparatif entre JBoss as et autres serveurs d'applications, et à la fin, nous avons montré l'installation, la configuration, le déploiement sur la version de JBoss choisi (JBoss7.1.1.Final).

Chapitre 05:

Mise en œuvre de JBoss pour la gestion centre de calcul 5

Dans ce chapitre on va faire la mise en œuvre du serveur d'application JBoss et la réalisation des fonctions qui aident à la gestion de centre de calcul de manière efficace et transparente, pour cela on va commencer par expliquer nos applications réalisées, ensuite nous allons montrer l'étape de configuration sur les deux niveaux administrateur et client, et on termine par l'étape de déploiement de nos applications sur JBoss.

5.1 L'utilité de serveur d'application pour un centre de calcul:

Pour simplifier le travail de l'administrateur du centre de calcul, le serveur d'application doit être capable de satisfaire les actions de l'administrateur vers les postes clients et des postes clients vert le serveur, donc les fonctions les plus important qui nous avons proposé sont :

5.1.1 Au niveau serveur :

- **Depuis la machine serveur, l'administrateur peut éteindre la machine Cliente.**

A l'aide de l'outil **éclipse** on a crée un projet web dynamique «turnoff » ce projet contient un fichier de type jsp :

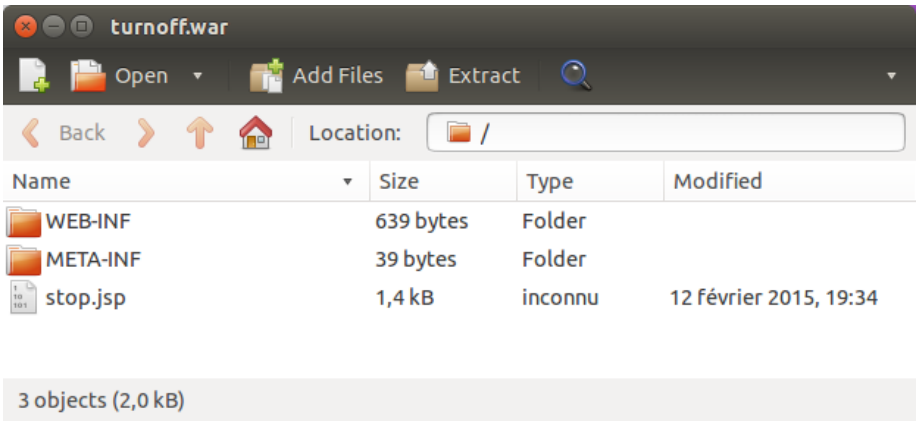


Figure 5.1- le fichier turnoff.war.

Pour éteindre le poste client le on doit passer par le mot de passe de cette machine, donc une configuration nécessaire pour éviter ce problème.

Allons au fichier /etc/sudoers de la machine clientes, et commenter la ligne :

%sudo ALL = (ALL : ALL) ALL

Et on ajoute la ligne: %sudo ALL = (ALL :ALL) NOPASSWD: ALL

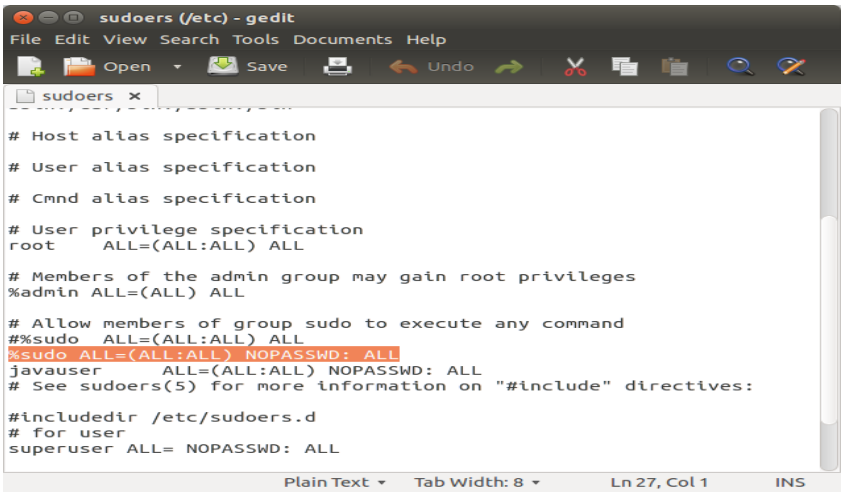


Figure 5.2- le fichier sudoers.

donc on peut exécuter notre script sans passer par le mot de passe de la machine cliente.

après Deployment de notre projet sous JBoss, dans notre navigateur on tape l'url suivant : 192.168.1.4 :8330/turnoff/stop.jsp

un message s'affiche au navigateur de l'administrateur : « cette machine va éteindre maintenant » et la machine cliente s'arrêtera tout de suite.

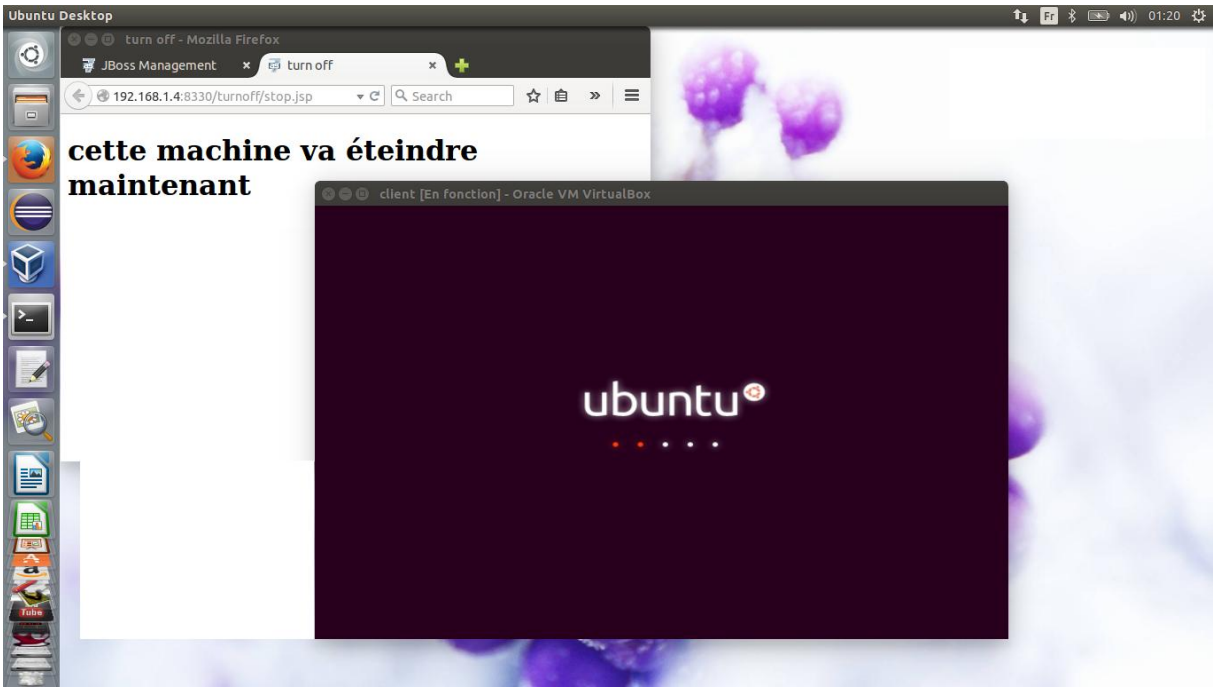


Figure 5.3- le résultat de l'application turnoff.

- **Afficher la date et l'heur des machines clientes au niveau serveur.**

A l'aide de l'outil **éclipse** on a crée un projet web dynamique «date_et_heur.war» ce projet contient un script timedate.jsp :

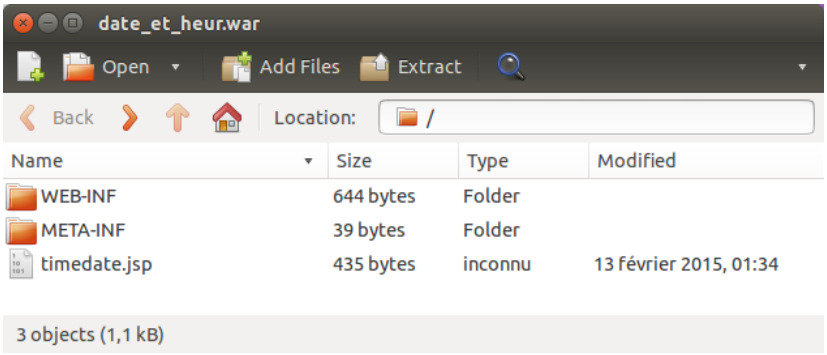


Figure 5.4- le contenu de fichier date_et_heur.war.

Après le déploiement de notre projet sur JBoss, dans le navigateur on tape l'url suivant : 192.168.1.4 :8330/date_et_heur/timedate.jsp

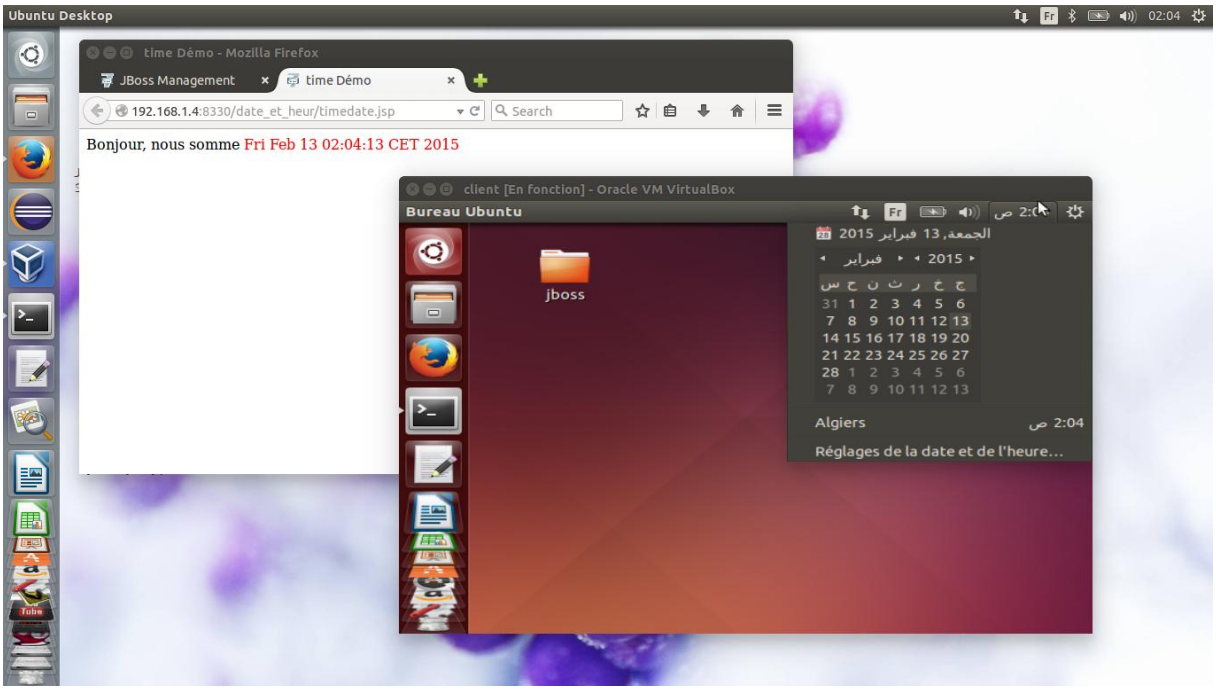


Figure 5.5- le résultat de l'application date_et_heur.

La date et l'heur de la machine cliente s'affiche sur le navigateur de l'administrateur comme l'image ci-dessous est indiqué.

5.1.2 Au niveau des clients :

- **Un client peut télécharger une application qui se trouve au niveau de serveur.**

Dans ce cas, à l'aide d'**éclipse** on a créé un projet web dynamique « mainserver » puis on a exporté ce projet dans un fichier mainserver.war qui contient les fichiers suivantes :

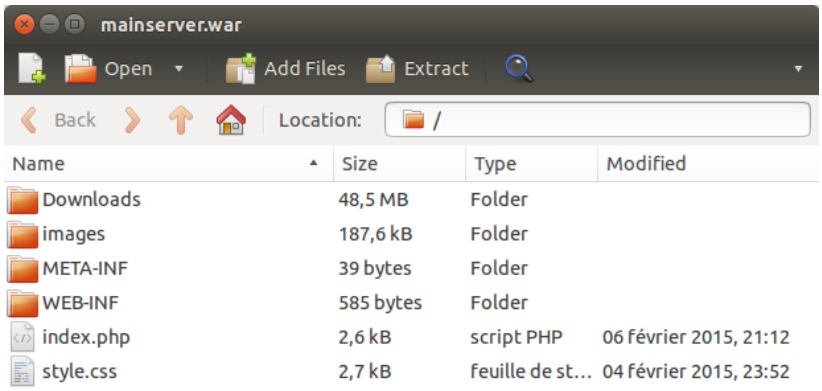


Figure 5.6- le contenu de fichier mainserver.war.

1. Download : contient les applications et les fichiers téléchargeables sur le serveur.
2. Image : contient les images utilisées dans les pages web.
3. META-INF
4. WEB-INF
5. Index.php
6. Style .css

Notre application est un simple site web au niveau du serveur, créé à l'aide de html et du PHP, ce site contient des fichiers et des applications à télécharger. Le client peut accéder à ce site pour télécharger ces applications après une authentification.

Donc au niveau de client, dans un navigateur on tape l'url suivant :

192.168.1.4:8330/mainserver/

Figure 5.7- l'accueil de site (demande de l'authentification).

Tandis que 192.168.1.4 est l'adresse IP de poste client.

Après l'authentification la page de téléchargement s'ouvrir :

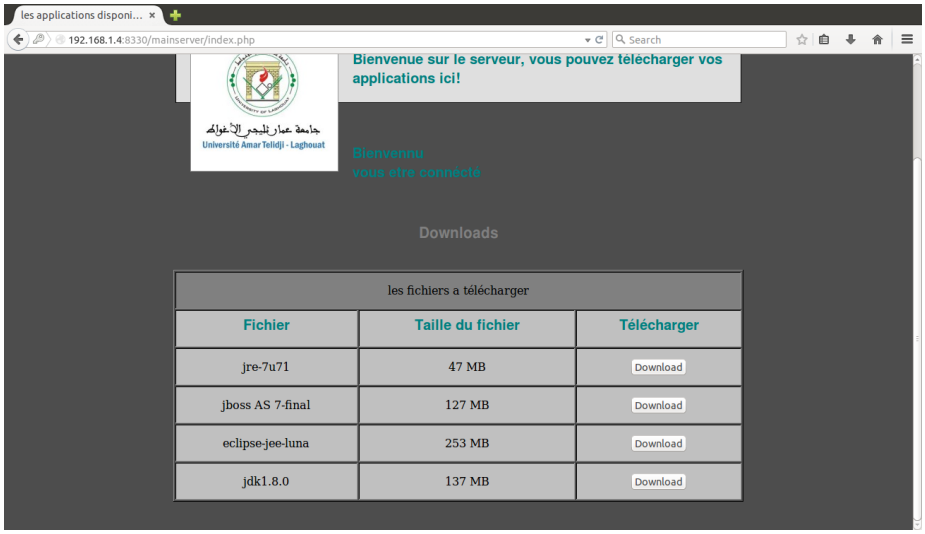


Figure 5.8- la page de téléchargement des applications sur le site.

Si on cliquer sur un bouton Download, on peut télécharger le fichier correspondant :

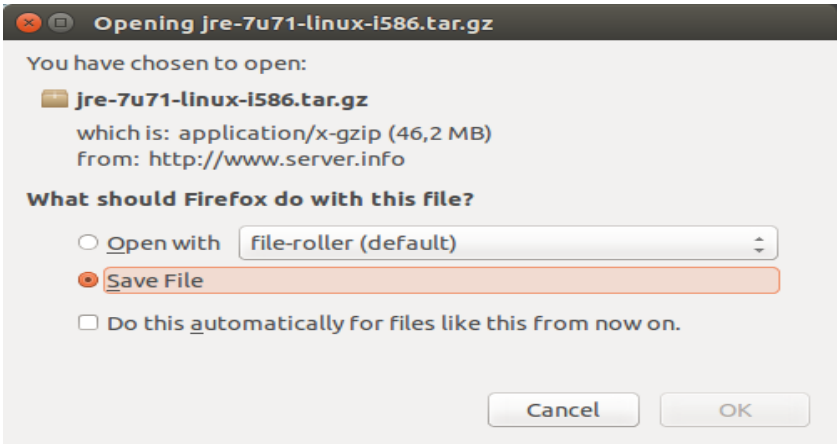


Figure 5.9- exemple de téléchargement d'un fichier.

5.2 L'environnement du travail :

Nous avons préparé deux hôtes. Nous allons utiliser ces deux hôtes comme suit:

- Les deux hôtes installer linux (ubuntu 14.04) comme système exploitation (nous avons choisir de travailler avec linux, comme on peut travailler avec Windows).
- on doit s'assurer que les hôtes sont en même réseau local (utilisons la commande ping).

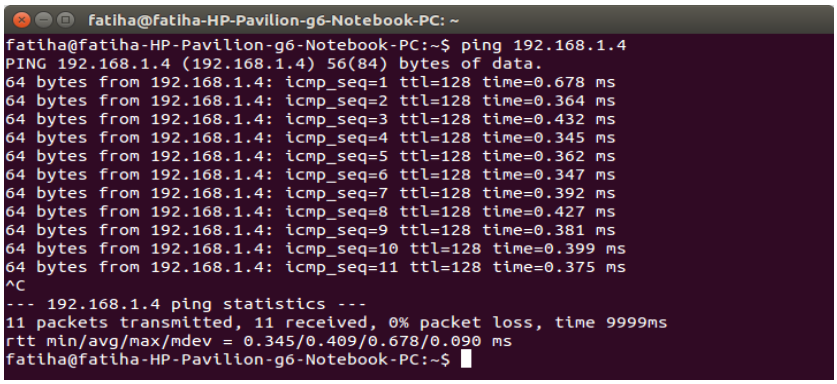


Figure 5.10- Ping de serveur vers la machine cliente.

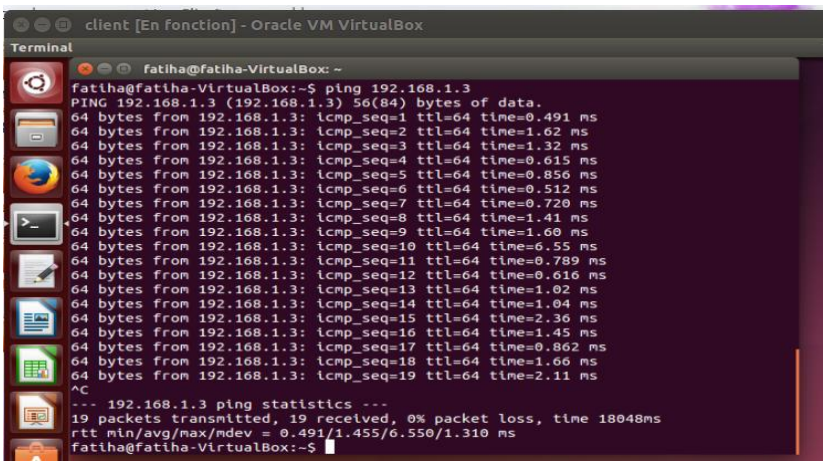


Figure 5.11- Ping de machine cliente vers la machine serveur

- on doit s'assure aussi qu'ils peuvent accéder l'un à l'autre via les différents ports UDP et TCP (les ports UDP et TCP doivent être ouvertes).

5.3 Scénario du travail :

Voici quelques détails sur ce que nous allons faire :

- On va appeler un hôte «maître» la machine physique, et l'autre «esclave» la machine virtuelle.
- Les deux hôtes le maître et l'esclave se déroulera JBoss AS7, le maître se démarre en tant que contrôleur du domaine (domain controller), le hôte esclave sous la gestion de domaine du maître (l'esclave se connecté au maître via son adresse IP et ce qui est configurer dans le fichier host.xml de l'hôte esclave voir par la suite).
- Le JBoss as7 sur le maître et l'esclave va former un groupe (cluster) et être découvert par httpd.

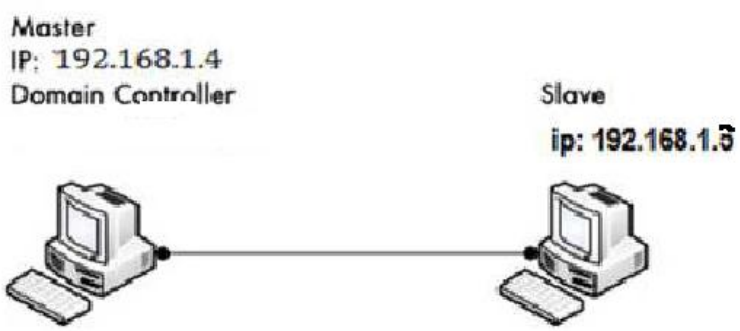


Figure 5.12- les deux hôtes maîtres et esclave.

- Nous allons déployer nos fonctions en mode domain et de vérifier que le projet est déployé à la fois maître et esclaves par le contrôleur de domaine.

5.4 La configuration du JBoss AS:

- **Décider entre l'exécution des serveurs autonomes (standalone) ou d'un domaine géré (mode domain) :**

  Le mode domaine de JBoss est tout au sujet de la gestion coordonnée des multi-serveurs avec elle JBoss AS 7 fournit un point central par lequel les utilisateurs peuvent gérer plusieurs serveurs, avec des capacités riches de garder les configurations de ces serveurs compatibles et la capacité à déployer des changements de configuration (vers les serveurs de manière coordonnée.

  Il est important de comprendre que le choix entre le mode domaine et le mode standalone Le mode domaine et le mode standalone déterminent la façon dont les serveurs sont gérés, pas ce que les capacités qu'ils fournissent :

  1) Pour une installation de serveur unique, ne gagne rien si on utilise le mode domaine, l'exécution d'un serveur standalone est un meilleur choix.
  2) Pour les environnements de production multi-serveurs, le meilleur choix entre le mode domaine par rapport mode standalone, et le mode domain si l'utilisateur veut utiliser les fonctionnalités de gestion centralisée.

- **Créer un cluster avec JBoss en mode domain:** comme nous avons une architecture centralisée, le bon choix dans ce cas est d'utiliser le mode domaine.

Le Clustering : Le clustering permet d'exécuter une application sur plusieurs serveurs en parallèle (les nœuds du cluster) tout en offrant une vue unique aux clients de l'application. La charge est répartie entre les différents serveurs, et même si un ou plusieurs des serveurs tombe en panne, la demande est toujours accessible via les nœuds du cluster survivants. Le

Clustering est crucial pour les applications d'entreprise évolutives, puisque on peu améliorer les performances en ajoutant des nœuds au cluster.

5.4.1 Configuration d'interface sur l'hôte maître :

D'abord on va ouvrir le fichier host.xml par l'exécution de la commande suivante :

$ sudo gedit /domain/configuration/host.xml

Les paramètres par défaut de l'interface dans ce fichier, comme suit :

```
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
    </interface>
    <interface name="public">
       <inet-address value="${jboss.bind.address:127.0.0.1}"/>
    </interface>
    <interface name="unsecured">
       <inet-address value="127.0.0.1" />
    </interface>
</interfaces>
```

Nous devons changer l'adresse de l'interface de gestion pour que l'esclave peut se connecter au maître. L'interface publique permet à l'application d'être accessible par http non local, et l'interface non sécurisée (unsecured) permet l'accès à distance RMI. L'adresse IP de notre maître est 192.168.1.4, donc on changer la configuration par:

```
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:192.168.1.4 }"/>
    </interface>
    <interface name="public">
       <inet-address value="${jboss.bind.address: 192.168.1.4 "/>
    </interface>
    <interface name="unsecured">
       <inet-address value="192.168.1.4 " />
    </interface>
</interfaces>
```

5.4.2 Configuration d'interface sur hôte esclave :

Maintenant, nous allons configurer l'interface de configuration sur l'esclave. Nous devons d'abord éliminer le fichier la domain.xml de l'esclave, car l'esclave ne doit pas agir en tant que contrôleur de domain car il est sous la direction du maître. Donc on renommer le domain.xml par domain.xml.move, on à renommer le domain.xml pour il ne sera pas traitée par JBoss car l'hôte esclave n'est pas le contrôleur de domain.

La commande est la suivante :

$ mv configuration/domain.xml configuration/domain.xml.move

Puis nous allons modifier host.xml. Même étapes comme celui du maître,

ouvrir host.xml:

$ vi domain/configuration/host.xml

La configuration que nous allons utiliser pour l'hôte esclave est un peu différente parce que nous devons le laisser connecter au maître. Nous devons d'abord définir le nom d'hôte. Nous changeons la propriété du nom de:

```
<host name="master" xmlns="urn:jboss:domain:1.1">
```

Par:

```
<host name="slave" xmlns="urn:jboss:domain:1.1">
```

Ensuite, nous devons modifier la section domaine contrôleur afin que slave peut se connecter au port de gestion de master:

```
<domain-controller>
    <remote host="192.168.1.4"port="9999"/>
</domain-controller>
```

où l'adresse 192.168.1.4 est l'adresse IP du master.

Enfin, on doit configurer section interfaces et exposer les ports de gestion à l'adresse public :

```
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:192.168.1.5}"/>
    </interface>
    <interface name="public">
        <inet-address value="${jboss.bind.address: 192.168.1.5 }"/>
    </interface>
    <interface name="unsecured">
        <inet-address value="192.168.1.5 " />
    </interface>
</interfaces>
```

où l'adresse 192.168.1.5 est l'adresse IP de l'esclave.

5.4.3 Création des utilisateurs :

Dans le répertoire bin de l'hôte master il ya un script appelé add-user.sh, nous allons l'utiliser pour ajouter de nouveaux utilisateurs au fichier de propriétés utilisé pour l'authentification de gestion du domain:

```
./add-user.sh
Enter the details of the new user to add.
Realm (ManagementRealm) :
Username : admin
Password : 123123
Re-enter Password : 123123
The username 'admin' is easy to guess
Are you sure you want to add user 'admin' yes/no? yes
About to add user 'admin' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'admin' to file '/home/weli/projs/jboss-as-
7.1.0.CR1b/standalone/configuration/mgmt-users.properties'
Added user 'admin' to file '/home/weli/projs/jboss-as-
7.1.0.CR1b/domain/configuration/mgmt-users.properties'
```

Comme indiqué ci-dessus, nous avons créé un utilisateur nommé «admin» avec un mot de passe .Ensuite, nous ajoutons un autre utilisateur appelé «slave»:

```
./add-user.sh
Enter the details of the new user to add.
Realm (ManagementRealm) :
Username : slave
Password : 123123
Re-enter Password : 123123
About to add user 'slave' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'slave' to file '/home/weli/projs/jboss-as-
7.1.0.CR1b/standalone/configuration/mgmt-users.properties'
Added user 'slave' to file '/home/weli/projs/jboss-as-
7.1.0.CR1b/domain/configuration/mgmt-users.properties'
```

Nous allons utiliser cet utilisateur pour que l'hôte esclave se connecte à hôte maître.

Dans l'hôte esclave dont nous avons besoin pour configurer host.xml pour l'authentification.
Nous devons changer la section security-realms de la manière suivante :

```
<security-realms>
   <security-realm name="ManagementRealm">
       <server-identities>
           <secret value="MTIzMTIz="/>
       </server-identities>
       <authentication>
           <properties path="mgmt-users.properties" relative-
to="jboss.domain.config.dir"/>
       </authentication>
   </security-realm>
</security-realms>
```

Nous avons ajouté server-identities en security-realm, qui est utilisé pour l'authentification de l'hôte quand l'esclave tente de se connecter au maître.
Parce que le nom d'hôte de l'esclave est défini comme ‘slave ‘, de sorte que nous devrions utiliser le mot de passe de l'utilisateur ‘slave’ sur le maître. Dans la propriété ‘**secret value**’ que nous avons 'MTIzMTIz =', qui est le code de base64 pour le mot de passe.
Ensuite, dans la section de contrôleur de domaine, nous devons aussi ajouter la propriété à security-realm :

```
<domain-controller>
<remote host="10.211.55.7" port="9999" security-realm="ManagementRealm"/>
</domain-controller>
```

Donc l'hôte esclave pouvait utiliser les informations d'authentification, qui nous avons fourni dans ManagementRealm.
dans le fichier domain.xml du maître on trouve:

```
<subsystem xmlns="urn:jboss:domain:messaging:1.3">
    <hornetq-server>...</hornetq-server>
</subsystem>
```

Il s'agit de la configuration pour la section hornetq, nous devons mettre 'cluster-user' et 'cluster-password' en elle:

```
<subsystem xmlns="urn:jboss: domain:messaging:1.3">
    <hornetq-server>
        <cluster-user>jms-user</cluster-user>
        <cluster-password>simple-pass</cluster-password>
    </hornetq-server>
</subsystem>
```

Le 'cluster-user' utilisateur et les paramètres 'cluster-password' ne sont que des valeurs définies par l'utilisateur; Le schéma d'authentification utilisé par HornetQ ici est simple: lorsqu'un hôte se connecte au contrôleur de domaine, l'instance HornetQ fonctionner sur contrôleur de domaine dira l'hôte de son 'cluster-user' et les paramètres 'cluster-password'.
Ensuite, l'instance HornetQ de l'hôte connecté utilisera ce paramètre pour communiquer et former un groupe avec d'autres instances de HornetQ cours d'exécution dans le domaine.
Il s'agit d'un moyen simple mais efficace pour prévenir certains étrangers de participer au cluster HornetQ de ce domaine.
Maintenant, tout est réglé pour les deux hôtes de s'exécuter en mode du domain. Commençons par lancer domain.sh sur les deux hôtes. Si tout va bien, nous devrions voir ce qui suit dans le journal sur l'hôte master.

```
[Host Controller] 21:30:52,042 INFO  [org.jboss.as.domain] (management-
handler-threads - 1) JBAS010918: Registered remote slave host slave
```

Cela signifie que toutes les configurations sont correctes et les deux hôtes sont en cours d'exécution en mode de domain comme prévu.
Maintenant on peut accéder à la console de gestion contrôleur de domain, dans un navigateur on tape l'URL suivant :
http://192.168.1.4:9990
Une fenêtre d'authentification s'affiche pour saisir le nom d'utilisateur et le mot de passe :

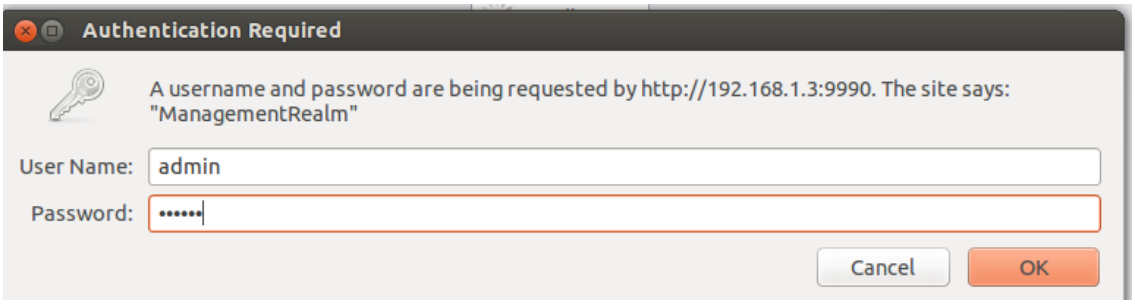


Figure 5.13 – Demande d'authentification.

Après l'authentification nous passons directement à l'interface « Admin console » de JBoss AS.

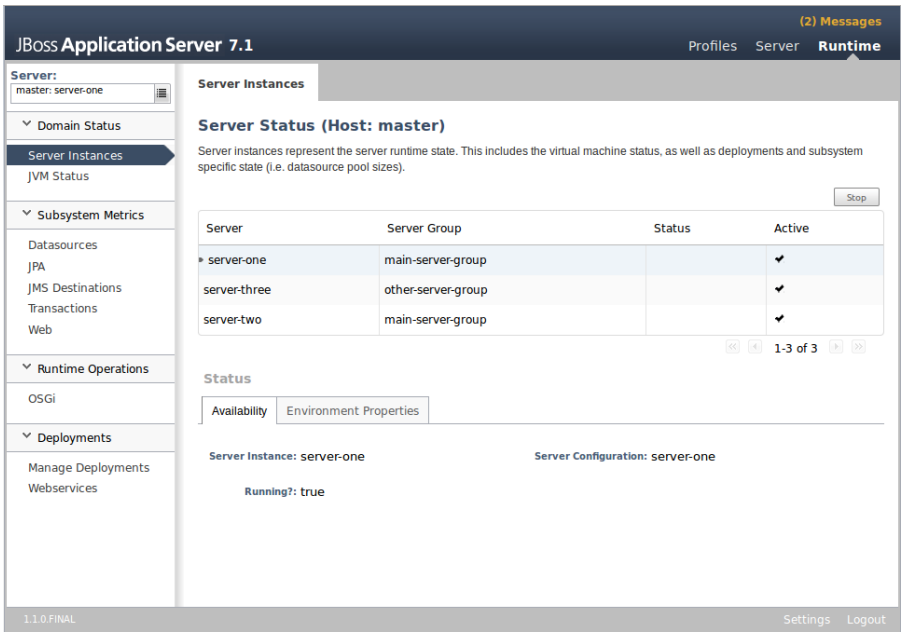


Figure 5.14 – Interface Admin Console de JBoss AS.

5.5 Déploiement :

A l'aide de l'outil éclipse nous avons créé nos projets et les exporter dans des fichiers de type war. Il existe plusieurs manières de déployer les applications sur le serveur d'applications JBoss, La méthode approuvé dans notre travail, est le déploiement sur l'interface web console de JBoss.

Donc pour déployer une application (un fichier .war), on se localise dans Manage Deployment sur l'interface console, et cliquer sur le bouton « Add Content » comme la figure 5.15 indique :

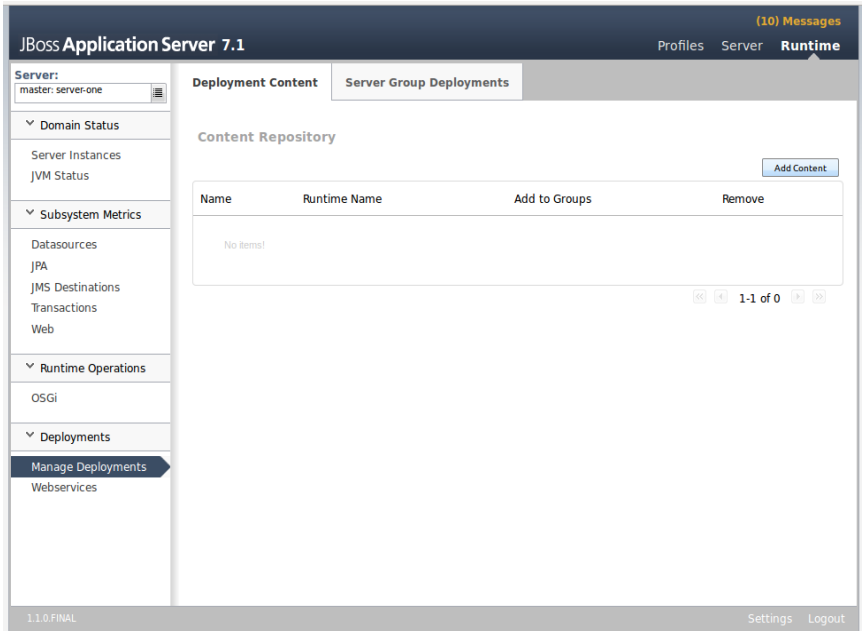


Figure 5.15 – l'ajout de nouvelle application (add content).

Puis, on va importer notre fichier turoff.war :

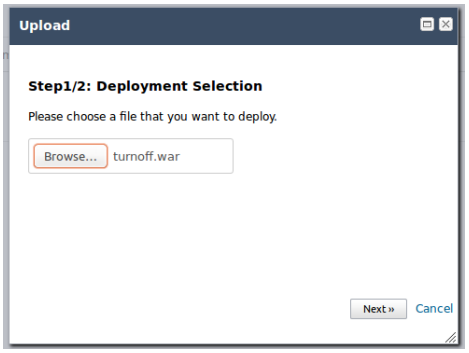


Figure 5.16 – importer le fichier war.

Et sélectionnée les groupes de serveurs pour notre application.

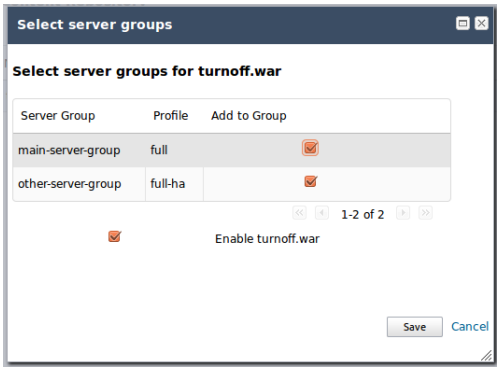


Figure 5.17- sélection de groupe de serveurs.

Cliquer sur enregistrer, et assurer que les serveurs sont active (enable) :

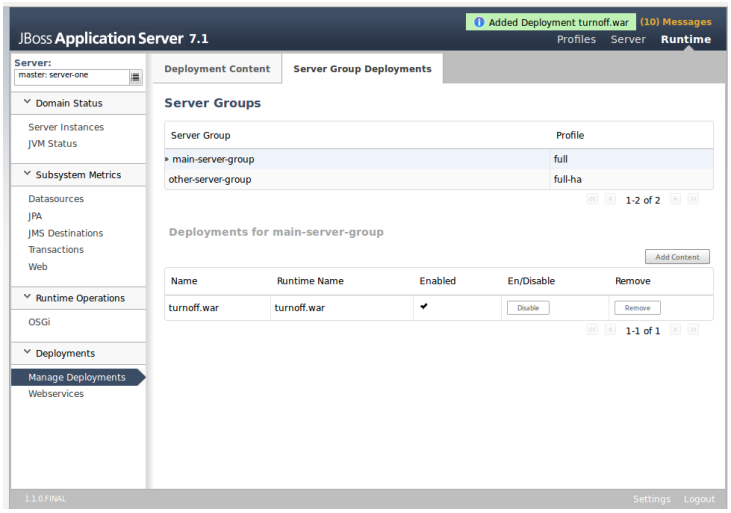


Figure 5.18- déploiement de l'application sur le groupe de serveur (main-server-group).

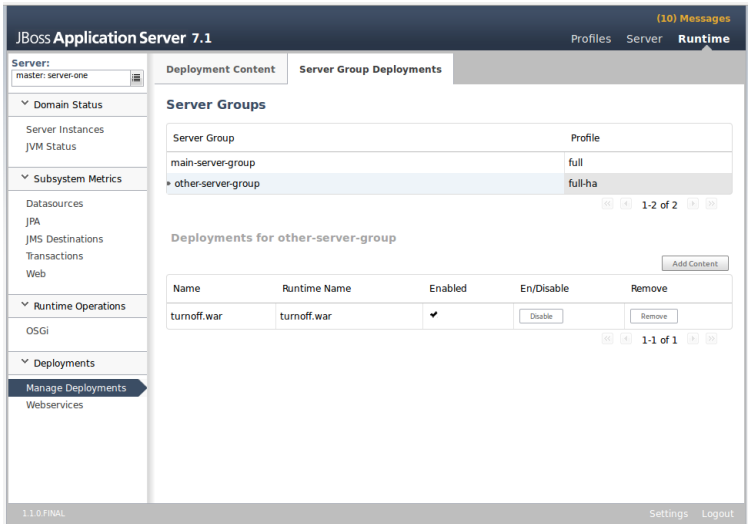


Figure 5.19- déploiement de l'application sur le groupe de serveur (other-server-group).

Donc l'application est bien déployer.

- **Groupe de serveurs (server groupe) :**

Dans le mode domain chaque instance de serveur d'application est un membre d'un groupe de serveurs. (Même si le groupe ne dispose que d'un seul serveur, le serveur est toujours membre d'un groupe). C'est la responsabilité du contrôleur de domaine (Domain Controller) et les contrôleurs d'hôte (Host Controllers) pour s'assurer que tous les serveurs dans un groupe de serveurs ont une configuration cohérente. Ils devraient tous être configurés avec le même profil et ils devraient avoir le même déploiement de contenu déployé.

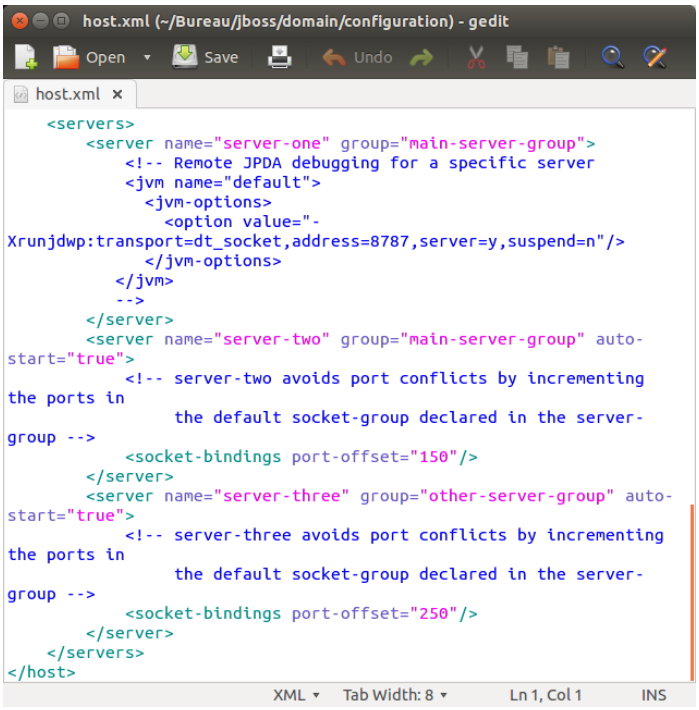


Figure 5.20- configurations des serveurs sur le fichier host.xml.

Une configuration serveur-groupe comprend les attributs requis suivants:

- name -- le nom du groupe de serveurs.
- profile -- le nom du profil ou les serveurs du groupe devraient fonctionner.
- socket-binding-group--spécifie le nom du groupe de liaison du socket par défaut à utiliser sur les serveurs du groupe.
- deployments -- le contenu de déploiement qui devrait être déployé sur les serveurs du groupe.
- system-properties -- propriétés du système qui doit être mis sur tous les serveurs dans le groupe.
- jvm -- paramètres JVM par défaut pour tous les serveurs du groupe.

5.6 Retrait d'une application :

Pour retirer une application, on doit tout d'abord supprimer les 2 groupes de serveurs main-server-groupe et other-server-groupe par le bouton « Remove » et puis supprimer l'application (ici date_et_heur.war) par le bouton « Remove » :

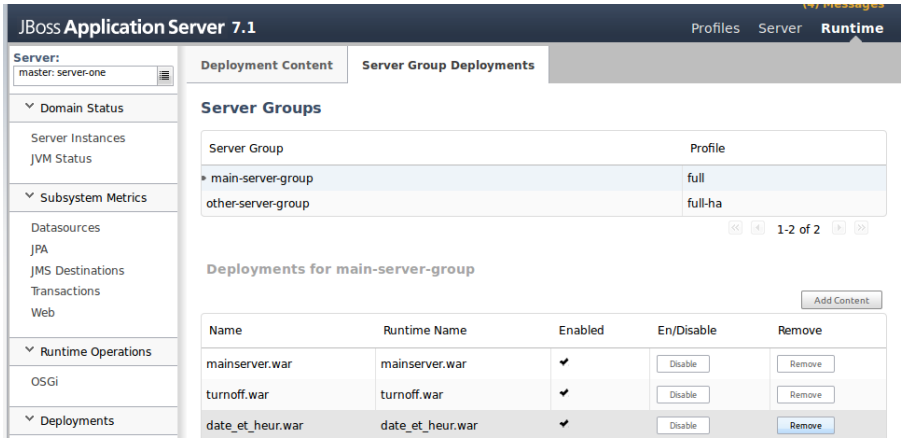


Figure 5.21- suppression de main-server-group.

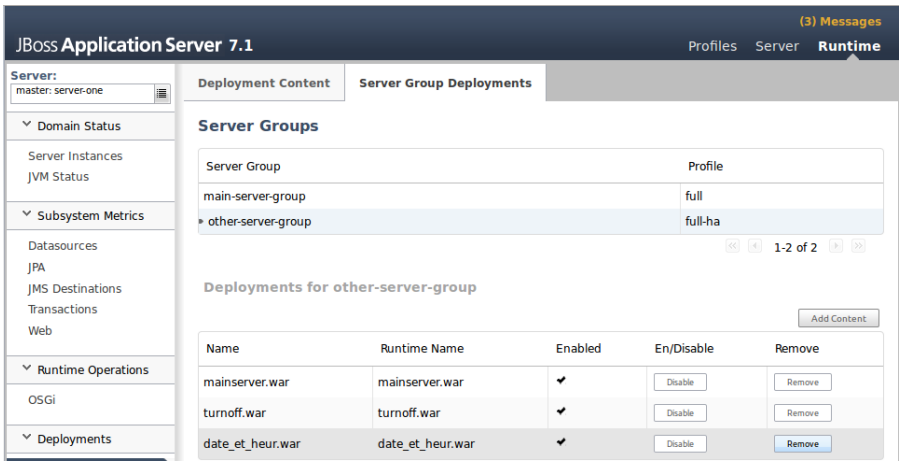


Figure 5.22- suppression de other-server-group.

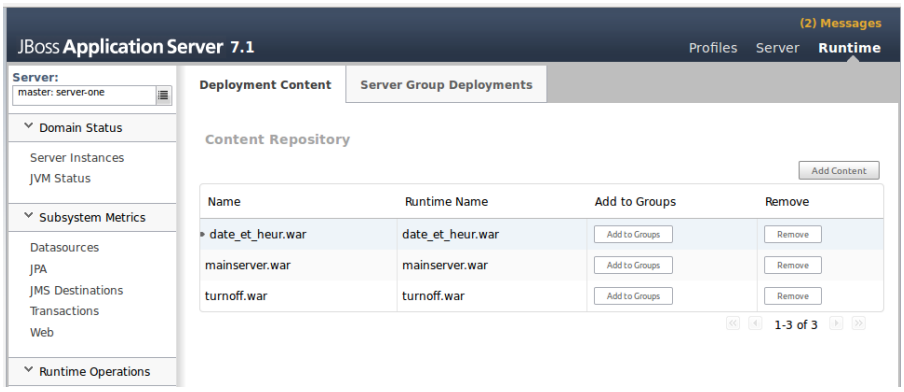


Figure 5.23- suppression de l'application date-rt-heur.war.

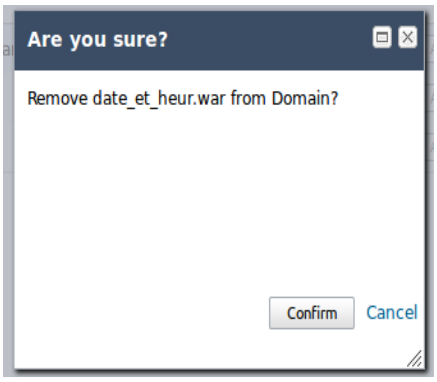


Figure 5.24- confirmation de suppression.

Cliquer sur Confirm et l'application est retirée.

5.7 Conclusion :

Au cours de ce chapitre, nous avons vue L'utilité de serveur d'application pour la gestion d'un centre de calcul, puis nous avons discuté la mise en œuvre de serveur d'application JBoss et expliqué les fonctions réalisées, l'environnement de travail et la configuration de JBoss as7 sur les hôtes administrateur et client, et finalement le déploiement des applications.

Conclusion générale et perspectives

Au cours de ce mémoire, nous avons présenté les résultats de notre recherche sur les serveurs d'applications, En raison de son utilité et son importance dans le domaine de
La gestion à distance.
Dans cette conclusion, nous résumons les contributions de notre mémoire, nous présentons des fonctions perspectives qui ont été déduites d'après les fonctions réalisées dans le chapitre cinq.
Il serait envisageable de faire la mise en œuvre réel de serveur d'application JBoss as7 pour la gestion de centre de calcul de département mathématique et informatique, pour rendre le travail de l'administrateur facile et transparent , et garantir une bonne gestion et organisation des sales de centre de calcul .les fonction réaliser dans le Cinquième chapitre sont comme suit :

- Éteindre une ou plusieurs machines clientes à distance : cette fonction aider l'administrateur à arrêté les poste des ses sale depuis son bureau à la fin de la journée.
- Afficher l'heure et la date système du machines clientes au niveau de serveur : cette fonction permettre l'administrateur de consulter l'heur de système des machines clientes et la date pour assurer une horloge précise. Les systèmes d'exploitation utilisent l'heure pour un certain nombre de tâches. En particulier, les fichiers disposent de plusieurs dates (date de création, date d'accès et date de dernière modification), qui sont utilisées par différents programmes. Les programmes de sauvegarde en font évidemment partie, parce qu'ils se basent sur les dates de modification des fichiers pour déterminer quels sont les fichiers qui doivent être sauvegardés depuis la dernière sauvegarde. Les programmes de maintenance sont également lancés à des dates précises, et les applications normales des utilisateurs peuvent utiliser la date système pour l'intégrer dans leurs documents. En clair, il est important que votre système soit à l'heure.

- Les machines cliente accéder au serveur pour télécharger une ou plusieurs applications à distance : dans ce car le client accéder au serveur pour télécharger une application parmi les applications disponible au serveur.

- **des fonctions perspectives :**
  - Démarrer et arrêter une machine à distance (amélioration**) :** dans le cas de centre de calcule une fonction nécessaire pou gérer l'ouverture / la fermeture des machines clientes à un temps définis par l'administrateur par exemple à 8h du matin les machines de sale 01 seront démarrées ou à 17h35 les poste clientes seront éteint.
  - Installation et désinstallation une application dans un poste client : l'administrateur peut accéder à une machine cliente pour installer ou désinstaller une application par exemple un antivirus.
  - Redémarrer une machine cliente : si une application installée par l'administrateur dans une machine client nécessite le redémarrage de la machine, donc l'administrateur doit lancer cette fonction à partir de serveur d'application.
  - L'administrateur lancer JBoss as7 dans les postes client : dans note travail on a lancés un script au démarrage de machine qui démarrer JBoss sur chaque machines, clientes ou JBoss et lancer dans un terminale ou un console, un problème se pose si un utilisateur de poste client fermer ce terminale donc JBoss s'arrêtera directement, et cette machine client ne sera pas enregistrer dans le serveur d'application JBoss, la fonction proposé est de lance une application a distance qui démarrer JBoss directement dans les machines clientes. Au lieu de redémarrer la machine.

Références

1. **Yves LE Monnier, Philippe Dartois(2002)**. «Les Serveurs d'applications » Sur le site de TELECOM LILLE [En ligne]. http://wapiti.telecom-lille1.eu.
2. **Prabhat Jha** , Site officiel du projet JBoss AS : URL http://www.jboss.org/jbossas/
3. **Michaël Tranchant**, «Java Web Server, Tomcat, JBoss, JRun, JOnAS», Décembre 2008.
4. **Virginie Salas**, «Serveurs d'application BEA, Websphere, Inprise et DNA» Janvier 2009.
5. **Renaud Dubourguais** (2008), «JBoss Application Server : exploitation et sécurisation ».
6. **Vincent (2011).** «Java, C/C++, Linux et plus : JBoss AS 7 vs Glassfish3»,sur le site[En ligne].http://the-fruits-rouges.blogspot.com.
7. **Jean-François Masler (2002).** «Comprendre le rôle des serveurs d'application» site de documentation informatique [En ligne]. http://www.01net.com.
8. JBoss (entreprise). Dans Wikipédia. Consulté le 27/05/2014, tiré de : http://fr.wikipedia.org/wiki/JBoss_(entreprise).
9. Site officiel du projet JBoss : URL http://www.jboss.com/.
10. Conférence de la RedTeam au Hack.lu, 2008 : URL http://wiki.hack.lu/
11. Documentation de JBoss AS https://www.jboss.org/community/docs/DOC12898.
12. Documentation sur l'implémentation JMX : URL http://jcp.org/en/jsr/detail?id=3.