1 - Installation et utilisation

Cas d'utilisation

Le scanner est un outil de ligne de commande qui permet d'analyser des fichiers binaires (war, ear ...) en vue d'une migration vers un nouveau serveur et/ou une nouvelle version java.

Cet outil permet de générer un rapport de migration consolidé incluant un rapport d'évaluation technologique, un rapport d'inventaire d'application, un rapport d'analyse détaillée de la migration ainsi que la configuration de la liste de fonctions.

Installation de l'outil

Pour l'installation, suivez les étapes ci-dessous :  
1. Téléchargez le kit d'outils de migration à partir du site Web WASdev, https://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/wasdev/downloads/wamt/ApplicationBinaryTP/binaryAppScannerInstaller.jar.  
2. Exécutez la commande ci-dessous et suivez les invites du kit d'outils de migration des fichiers binaires d'application.  
java -jar binaryAppScannerInstaller.jar  
Une fois que vous avez accepté l'accord et spécifié le répertoire cible d'installation, l'outil sera installé dans le répertoire *<répertoire cible>/wamt.*

**NB** : Cet outil requiert l'exécution d'un JRE/JDK Java 7 ou 8.

Lancer l'analyse

Afin de générer le rapport de migration, ouvrez une ligne de commande et pointez vers le répertoire d'installation de l'outil *<répertoire cible>/wamt*  
Tapez la commande *java -jar binaryAppScanner.jar --help --all* pour consulter toutes les options offertes par l'outil de migration

Les options principales à utiliser sont :

* *binaryInputPath* : correspond aux fichiers binaire .war ou .ear candidats à la migration, c'est le seul paramètre obligatoire pour lancer la commande d'analyse.
* *sourceAppServer* : correspond au serveur utilisé avant la migration
* *sourceJava* : correspond à la version java actuelle de l'application
* *sourceJavaEE* : correspond à la version javaEE actuelle de l'application
* *targetAppServer* : correspond au sereur cible de la migration
* *targetJava* : correspond à la version java cible de la migration
* *targetJavaEE* : correspond à la version javaEE cible de la migration

**Exemple de commande**  
*java -jar binaryAppScanner.jar C:\workspace\java.angular-2-tiers-archetype\server\yourapplication\_PRES\_EAR\target\yourapplication\_PRES\_EAR-1.0.0-0-SNAPSHOT.ear --all*  
*--sourceAppServer=was855 --sourceJava=ibm7 --sourceJavaEE=ee7 --targetAppServer=libertyCore --targetJava=ibm8 --sourceJavaEE=ee8*

2 - Classification des alertes de la migration WAS - Liberty Core

Le rapport d'analyse généré par le scanner permet de vérifier la compatibilité de la migration de votre application vers le nouveau serveur et la nouvelle version Java.

Dans cette partie, nous allons nous intéresser aux principales alertes que peut générer ce scanner concernant la migration d'un serveur **WAS** vers **Liberty Core.**

API non supportées par LC

Une grande partie des alertes potentielles de cette migration concernerait des librairies qui sont supportées par le serveur de départ, Web sphere 8.5.5 dans notre cas, mais qui ne sont pas supportées par Liberty Core.

Pour la plupart de ces librairies, le scanner proposerait d'utiliser des implémentations équivalentes offertes par LC et pour d'autres le scanner indiquerait juste que LC ne les supporte pas, sans pour autant proposer des technologies équivalentes, c'est le cas des EJB par exemple.

Pour pallier à ces problématiques, il y'a deux solutions possibles :

1- Migration applicative :

Faire évoluer le code de l'application pour utiliser des librairies/technologies supportées par LC.

Cette solution permettrait de moderniser les technologies de l'application et d'être en cohérence avec le serveur LC. En contrepartie la charge de ces changements peut s'avérer importante.

2- Ajouter les jars manquants :

Même si LC ne supporte pas certaines librairies, il est toujours possible de les inclure manuellement.

- Soit en les faisant porter par le livrable de l'application (war ou ear).

- Soit en les plaçant dans un répertoire coté serveur (shared lib).

Dans les deux cas, vous serez amené à ajouter, en plus de ces jars manquants, les implémentations et les dépendances correspondantes.

Voici les alertes les plus récurrentes et leurs descriptions associées pour des API non supportées par LC

**L'utilisation des API tierces fournies par le système nécessite une configuration**

Pour utiliser les API tierces fournies par le système dans les applications Liberty, vous devez configurer les applications de manière à inclure les API.

Sur le serveur WebSphere Application Server Traditional, ces API sont disponibles sans configuration.

Les API tierces fournies par le système comportent des packages d'Apache Wink, d'Apache OpenJPA, d'Apache Aries et d'Eclipse Equinox.

**Certaines API tierces ne sont pas disponibles dans Liberty / Les API tierces org.apache ne sont pas disponibles dans Liberty**

Cette règle marque l'utilisation des API tierces non disponibles afin que vous puissiez évaluer la complexité de votre migration.

Si votre application exige des API tierces, placez les fichiers JAR correspondants à un emplacement visible pour le chargeur de classe de l'application, tel le répertoire WEB-INF/lib.

Si vous ne disposez pas des fichiers JAR ou si la licence ne vous permet pas de les placer dans votre application, modifiez cette dernière afin de supprimer la dépendance sur les API tierces.

**Java Message Service (JMS) n'est pas disponible**

Cette règle marque le code Java ayant des références au package javax.jms.

Cette règle marque également les références suivantes dans les fichiers application.xml, ejb-jar.xml et web.xml :

* <res-type>javax.jms.ConnectionFactory</res-type>
* <res-type>javax.jms.QueueConnectionFactory</res-type>
* <res-type>javax.jms.TopicConnectionFactory</res-type>

Java Message Service (JMS) est pris en charge dans toutes les éditions WebSphere Application Server, à l'exception de Liberty Core.

**Les Enterprise JavaBeans 1.x/2.x ne sont pas disponibles**

Cette règle marque le code Java ayant des références aux classes suivantes :

* javax.ejb.EntityContext
* javax.ejb.Handle
* javax.ejb.HomeHandle
* javax.ejb.RemoteHome
* javax.ejb.EJBHome
* javax.ejb.EJBLocalHome
* javax.ejb.EJBLocalObject
* javax.ejb.EJBMetaData
* javax.ejb.EJBObject

Cette règle marque également les références suivantes dans le fichier ejb-jar.xml :

* <ejb-jar version="2.1"></ejb-jar>
* Les noms DTD sont http://java.sun.com/j2ee/dtds/ejb-jar\_1\_1.dtd ou http://java.sun.com/dtd/ejb-jar\_2\_0.dtd

Les beans Enterprise JavaBeans (EJB) 1.x et 2.x ne sont pas pris en charge dans l'édition Liberty Core. Si votre application utilise des beans JavaBeans de version 1.x ou 2.x, vous pouvez utiliser la prise en charge de Java Platform, Enterprise Edition (Java EE) 7 EJB 3.2 dans d'autres éditions de Liberty. Pour activer EJB 3.2, configurez la fonction ejb-3.2 dans votre configuration de serveur Liberty. La fonction ejb-3.2 inclut la fonction ejbHome-3.2.

**Les bundles Enterprise JavaBeans (EJB) ne sont pas prises en charge**

Les bundles EJB, qui sont des applications OSGi qui exposent les EJB sous forme de service, ne sont pas pris en charge dans Liberty ou Liberty Core.

Cette règle marque l'en-tête Export-EJB dans le fichier MANIFEST.MF avec une valeur autre que NONE. La valeur NONE déclare qu'aucun EJB n'est exposé sous forme de service.

**Les métadonnées de services Web pour la plateforme Java ne sont pas disponibles**

Cette règle marque le code Java ayant des références aux classes suivantes :

* javax.jws.WebService
* javax.jws.WebMethod
* javax.jws.OneWay
* javax.jws.WebParam
* javax.jws.WebResult
* javax.jws.HandlerChain
* javax.jws.soap.SOAPBinding

La spécification Web Services Metadata for the Java Platform est prise en charge par la fonction JAX-WS disponible dans toutes les éditions de WebSphere Application Server, à l'exception de Liberty Core.

**L'API Java Connector Architecture (JCA) n'est pas disponible**

Cette règle marque le code Java qui fait référence au package javax.resource.

Cette règle marque également les éléments <connector> dans le fichier *ra.xml* et les éléments <res-type>javax.resource.cci.ConnectionFactory</res-type> dans les fichiers *application.xml*, *ejb-jar.xml* et *web.xml*.

L'API Java Connector Architecture (JCA) est prise en charge dans toutes les éditions WebSphere Application Server à l'exception de Liberty Core. Si votre application est déployée dans Liberty, vous devez activer la fonction pour JCA.

**Les API CommonJ Timer et Work Manager ne sont pas disponibles dans Liberty**

Cette règle signale les API des packages commonj.   
Ces API ne sont pas fournies dans Liberty et sont signalées comme obsolètes dans WebSphere Application Server Traditional version 9.0. Les API CommonJ Timer et Work Manager ont été remplacées par JSR 236, Concurrency Utilities for Java EE.

**Remarque :** Concurrency Utilities for Java EE ne fournit pas de remplacement pour les API ommonj.sdo

**Les API WebSphere Management sont indisponibles**

Les API WebSphere Management ci-dessous ne sont pas disponibles sous Liberty :

* com.ibm.websphere.ant.tasks
* com.ibm.websphere.hamanager.jmx
* com.ibm.websphere.interrupt
* com.ibm.websphere.management
* com.ibm.websphere.naming
* com.ibm.websphere.product

Vous devez modifier l'application de sorte qu'elle puisse être exécutée sous Liberty.

**L'interface SPI et l'API WebSphere Batch ne sont pas disponibles**

L'interface SPI et l'API WebSphere Batch ne sont pas disponibles dans Liberty. Cette règle marque les packages suivants :

* com.ibm.websphere.batch\*
* com.ibm.websphere.ci
* com.ibm.websphere.grid.spi
* com.ibm.websphere.longrun
* com.ibm.wsspi.batch\*

**API WebSphere Asynchronous Beans**

Cette règle signale les API des packages com.ibm.websphere.asyncbeans. Ces API ne sont pas fournies dans Liberty et sont signalées comme obsolètes dans WebSphere Application Server Traditional version 9.0. Les API WebSphere Asynchronous Beans API sont remplacées par JSR 236, Concurrency Utilities for Java EE.

Concurrency Utilities for Java EE ne fournit pas de remplacement pour les API com.ibm.websphere.asyncbeans.pool.

**Le service Activity Session n'est pas disponible**

Le service Activity Session n'est pas disponible dans Liberty. Cette règle marque les éléments suivants :

* Références aux API dans le package com.ibm.websphere.ActivitySession
* Nom JNDI de session d'activité java:comp/websphere/UserActivitySession
* Eléments liés aux sessions d'activité dans les descripteurs de déploiement d'extension Java EE (Java Platform, Enterprise Edition)

**L'API Java pour le RPC basé sur XML (JAX-RPC) n'est pas disponible**

 L'API Java pour le RPC basé sur XML (JAX-RPC) n'est pas prise en charge dans Liberty ou Liberty Core. Cette technologie est obsolète dans WebSphere Application Server Traditional version 9.0 et pourrait être retirée dans une version ultérieure. Si votre application utilise JAX-RPC, l'option de migration privilégiée consiste à utiliser JAX-WS, bien que vous puissiez envisager les alternatives suivantes:

* Option 1 : Migration des services Web JAX-RPC vers les services Web JAX-WS
* Option 2 : Utilisation du moteur Apache Axis 1 JAX-RPC dans Liberty
* Option 3 : Utilisation de WebSphere Application Server Traditional avec son moteur JAX-RPC natif

**L'API Java Connector Architecture (JCA) n'est pas disponible**

Cette règle marque le code Java qui fait référence au package javax.resource.

Cette règle marque également les éléments <connector> dans le fichier *ra.xml* et les éléments <res-type>javax.resource.cci.ConnectionFactory</res-type> dans les fichiers *application.xml*, *ejb-jar.xml* et *web.xml*.

**Les sessions JMS faisant l'objet d'une transaction locale portée ne fonctionnent pas dans Liberty**

Même si WebSphere Application Server Traditional permet les sessions JMS faisant l'objet d'une transaction locale, le portage d'une session JMS faisant l'objet d'une transaction locale de WebSphere Application Server Traditional vers Liberty n'est pas autorisé.

Cette règle marque le code Java avec des références aux méthodes ci-dessous lorsque la valeur booléenne transacted est true ou que la valeur sessionMode est Session.SESSION\_TRANSACTED ou JMSContext.SESSION\_TRANSACTED:

* javax.jms.Connection.createSession(int sessionMode)
* javax.jms.Connection.createSession(boolean transacted, int acknowledgeMode)
* javax.jms.ConnectionFactory.createContext(int sessionMode)
* javax.jms.JMSContext.createContext(int sessionMode)
* javax.jms.QueueConnection.createQueueSession(boolean transacted, int acknowledgeMode)
* javax.jms.TopicConnection.createTopicSession(boolean transacted, int acknowledgeMode)

L'appel de ces méthodes avec ces valeurs spécifiques peuvent aboutir à un comportement différent dans Liberty.

L'API Java Connector Architecture (JCA) est prise en charge dans toutes les éditions WebSphere Application Server à l'exception de Liberty Core. Si votre application est déployée dans Liberty, vous devez activer la fonction pour JCA.

Configuration requise pour LC

**L'utilisation des interfaces java.sql.Driver et java.sql.DriverManager requiert une configuration**

Dans Liberty, vous pouvez vous connecter facilement à une base de données en configurant un élément <dataSource> dans votre fichier server.xml et en utilisant une recherche JNDI dans votre application. Pour plus d'informations sur la configuration d'une source de données, voir Configuration de la connectivité de base de données relationnelle dans Liberty.

Toutefois, si votre application doit accéder à la base de données en utilisant une des méthodes suivantes, une étape de configuration supplémentaire est requise :

* java.sql.Driver.connect()
* java.sql.DriverManager.getConnection()
* java.sql.DriverManager.getDriver()

Si vous utilisez ces méthodes, vous devez configurer une bibliothèque partagée pour le pilote de base de données et la référencer sur l'élément <classloader> de l'application.

Dans le fichier server.xml, définissez une bibliothèque partagée pour le pilote de base de données dans l'élément <library> :

<library id="yourDBDriverLib">   
<fileset dir="yourDBDriverDir"includes="yourDBDriver.jar"/>   
</library>

Référencez la bibliothèque partagée en ajoutant un élément de chargeur de classe qui désigne la bibliothèque dans votre élément <application> :

<application ...>  
<classloader commonLibraryRef="yourDBDriverLib"/>  
</application>

**Les valeurs d'espace de nom dans le fichier web.xml doivent être cohérentes avec la version du descripteur**

Le serveur Liberty ne peut pas charger de modules web lorsque le descripteur de déploiement web.xml possède une valeur d'espace de nom qui ne correspond pas à la version définie.

Les modules Web de la version 2.4 doivent utiliser l'espace de nom http://java.sun.com/xml/ns/j2ee.

Les modules Web de la version 2.5 ou 3.0 doivent utiliser l'espace de nom http://java.sun.com/xml/ns/java.

Les modules Web de la version 3.1 ou des versions ultérieures doivent utiliser l'espace de nom http://xmlns.jcp.org/xml/ns/javaee.

**Les valeurs d'espace de nom dans le fichier application.xml doivent être cohérentes avec la version du descripteur**

Le serveur Liberty ne peut pas charger de modules d'application lorsque le descripteur de déploiement application.xml possède une valeur d'espace de nom qui ne correspond pas à la version définie.

Les modules d'application de la version 1.4 doivent utiliser l'espace de nom http://java.sun.com/xml/ns/j2ee. Les modules d'application de la version 5 ou 6 doivent utiliser l'espace de nom http://java.sun.com/xml/ns/javaee.

Les modules d'application de la version 7 ou des versions ultérieures doivent utiliser l'espace de nom http://xmlns.jcp.org/xml/ns/javaee.

**Nécessité de supprimer les classes JSP précompilées**

Les classes JSP (JavaServer Pages) précompilées générées avec la commande JspBatchCompiler sur le serveur WebSphere Application Server Traditional ne sont pas compatibles avec Liberty. Cette règle marque les fichiers JSP qui comportent des fichiers de classes correspondants dans le dossier WEB-INF/classes. Les fichiers de classes JSP précompilées doivent être supprimés de l'application.

Liberty ne possède pas de compilateur par lots unique, comme la commande JspBatchCompiler. Les deux attributs de configuration du serveur Liberty ci-dessous contrôlent la compilation JSP :

* Définissez prepareJSPs=0 sur l'élément <jspEngine> pour compiler tous les fichiers JSP au démarrage d'une application.
* Vous pouvez également définir deferServletLoad="false" sur l'élément <webContainer> pour démarrer toutes les applications au démarrage du serveur Liberty, ce qui entraîne de fait la compilation de tous les fichiers JSP au démarrage du serveur au lieu que ce soit le cas lors de la première demande de l'application.

**Différence de comportement pour la validation de nom d'hôte de service Web**

Lors de l'exécution des services Web sur le serveur Liberty, le nom d'hôte de l'URL HTTPS est validé avec le certificat par défaut, ce qui n'est pas le cas sur le serveur WebSphere Application Server Traditional. Si le nom d'hôte n'est pas valide, le serveur Liberty génère l'exception suivante :

java.io.IOException: Le nom d'hôte de l'URL https ne correspond pas au nom usuel (CN) sur le certificat du serveur dans le fichier de clés certifiées du client. Assurez-vous que le certificat du serveur est correct ou décochez cette case (NON recommandé en environnement de production) pour définir la propriété de configuration TLS disableCNCheck du client CXF à 'true'.

Cette différence de comportement a une incidence sur les applications client qui utilisent l'API Java pour services Web XML (JAX-WS), fournie par le package javax.xml.ws. Si les API JAX-WS sont référencées, le projet est marqué une fois.

La solution la plus sécurisée consiste à créer un certificat serveur avec le nom d'hôte correct. Sinon, vous pouvez personnaliser le comportement de validation en définissant en conséquence la propriété http.conduit.tlsClientParameters.disableCNCheck dans le fichier ibm-ws-bnd.xml.

<service-ref name="service/SimpleEchoService">

<port name="SimpleEchoPort">

<properties vendor="IBM"/>

</port>

<**properties http.conduit.tlsClientParameters.disableCNCheck="true"**/>

</service-ref>

**Les fichiers JAR des sous-dossiers ne sont pas chargés**

Le conteneur Web charge automatiquement les fichiers JAR dans le dossier WEB-INF/lib des fichiers WAR, de telle sorte que les classes de ces fichiers soient disponibles pour l'application. Sur le serveur WebSphere Application Server Traditional, le conteneur Web charge également automatiquement les fichiers JAR dans les sous-dossiers du dossier WEB-INF/lib.

Dans Liberty, les fichiers JAR dans les sous-dossiers du dossier WEB-INF/lib ne sont pas chargés automatiquement, et des exceptions java.lang.NoClassDefFoundError peuvent être générées pour l'application lors de l'exécution.

Cette règle détecte les fichiers JAR dans les sous-dossiers de WEB-INF/lib. L'action Afficher les résultats présente les fichiers JAR dans l'explorateur de packages. Pour éviter des exceptions dans votre application, déplacez le fichier JAR dans le dossier WEB-INF/lib.