How do I migrate my application from AS7 to WildFly

About this Document

The purpose of this guide is to document changes that are needed to successfully run and deploy AS 7 applications on WildFly. It provides information on to resolve deployment and runtime problems and how to prevent changes in application behavior. This is the first step in moving to the new platform. Once the application is successfully deployed and running on the new platform, plans can be made to upgrade individual components to use the new functions and features of WildFly.

Overview of WildFly

The list of WildFly new functionality is extensive, being the most relevant, with respect to server and application migrations:

- Java EE7 - WildFly is a certified implementation of Java EE7, meeting both the Web and the Full profiles, and already includes support for the latest iterations of CDI (1.2) and Web Sockets (1.1).

- Undertow - A new cutting-edge web server in WildFly, designed for maximum throughput and scalability, including environments with over a million connections. And the latest web technologies, such as the new HTTP/2 standard, are already onboard.
- Apache ActiveMQ Artemis - WildFly's new JMS broker. Based on an code donation from HornetQ, this Apache subproject provides outstanding performance based on a proven non-blocking architecture.
- IronJacamar 1.2 - The latest IronJacamar provides a stable and feature rich JCA & Datasources support.
- JBossWS 5 - The fifth generation of JBossWS, a major leap forward, brings new features and performances improvements to WildFly Web Services
- RESTEasy 3 - WildFly includes the latest generation of RESTEasy, which goes beyond the standard Java EE REST APIs (JAX-RS 2.0), by also providing a number of useful extensions, such as JSON Web Encryption, Jackson, Yaml, JSON-P, and Jettison.
- OpenJDK ORB - WildFly switched the IIOP implementation from JacORB, to a downstream branch of the OpenJDK Orb, leading to better interoperability with the JVM ORB and the Java EE RI.
- Feature Rich Clustering - Clustering support was heavily refactored in WildFly, and includes several APIs for applications
- Port Reduction - By utilising HTTP upgrade, WildFly has moved nearly all of its protocols to be multiplexed over just two HTTP ports: a management port (9990), and an application port (8080).
- Enhanced Logging - The management API now supports the ability to list and view the available log files on a server, or even define custom formatters other than the default pattern formatter. Deployment's logging setup is also greatly enhanced.

The support for some technologies was removed, due to the high maintenance cost, low community interest, and much better alternative solutions:

- CMP EJB - JPA offers a much more performant and flexible API
- JAX-RPC - JAX-WS offers a much more accurate and complete solution
- JSR-88 - With very little adoption, the more complete deployment APIs provided by vendors are preferred

Server Migration

Migrating an AS7 server to WildFly consists of migrating custom configuration files, and some persisted data that may exist.

JacORB Subsystem

WildFly ORB support is provided by the JDK itself, instead of relying on JacORB. A subsystem configuration migration is required.

JacORB Subsystem Configuration

The extension's module **org.jboss.as.jacorb *is replaced by module *org.wildfly.iiop-openjdk**, while the subsystem configuration namespace **urn:jboss:domain:jacorb:2.0** is replaced by **urn:jboss:domain:iiop-openjdk:1.0**.

The XML configuration of the new subsystem accepts only a subset of the legacy elements/attributes. Consider the following example of the JacORB subsystem configuration, containing all valid elements and attributes:

```
<subsystem xmlns="urn:jboss:domain:jacorb:1.3">
   <orb name="JBoss" print-version="off" use-imr="off" use-bom="off"  cache-typecodes="off"
       cache-poa-names="off" giop-minor-version ="2" socket-binding="jacorb" ssl-socket-binding="jacorb-
ssl">
       <connection retries="5" retry-interval="500" client-timeout="0" server-timeout="0"
           max-server-connections="500" max-managed-buf-size="24" outbuf-size="2048"
           outbuf-cache-timeout="-1"/>
       <initializers security="off" transactions="spec"/>
   </orb>
   <poa monitoring="off" queue-wait="on" queue-min="10" queue-max="100">
       <request-processors pool-size="10" max-threads="32"/>
   </poa>
   <naming root-context="JBoss/Naming/root" export-corbaloc="on"/>
   <interop sun="on" comet="off" iona="off" chunk-custom-rmi-valuetypes="on"
       lax-boolean-encoding="off" indirection-encoding-disable="off" strict-check-on-tc-creation="off"/>
   <security support-ssl="off" add-component-via-interceptor="on" client-supports="MutualAuth"
       client-requires="None" server-supports="MutualAuth" server-requires="None"/>
   <properties>
       <property name="some_property" value="some_value"/>
   </properties>
</subsystem>
```

Properties that are not supported and have to be removed:

- <orb/>: client-timeout, max-managed-buf-size, max-server-connections, outbuf-cache-timeout, outbuf-size, connection retries, retry-interval, name,server-timeout
- <poa/>: queue-min, queue-max, pool-size, max-threads

On-off properties: have to either be removed or in off mode:

- <orb/>: cache-poa-names, cache-typecodes, print-version, use-bom, use-imr
- <interop/>: all except sun
- <poa/>: monitoring, queue-wait

In case the legacy subsystem configuration is available, such configuration may be migrated to the new subsystem by invoking its `migrate` operation, using the CLI management client:

```
/subsystem=jacorb:migrate
```

There is also a `describe-migration` operation that returns a list of all the management operations that are performed to migrate from the legacy subsystem to the new one:

```
/subsystem=jacorb:describe-migration
```

Both `migrate` and `describe-migration` will also display a list of migration-warnings if there are some resource or attributes that can not be migrated automatically. The following is a list of these warnings:

- Properties X cannot be emulated using OpenJDK ORB and are not supported
  This warning means that mentioned properties are not supported and won't be included in the new subsystem configuration. As a result of that admin must be aware that any behaviour implied by those properties would be inexistent. Admin has to check whether subsystem is able to operate correctly without that behaviour on the new server.Unsupported properties: cache-poa-names, cache-typecodes, chunk-custom-rmi-valuetypes, client-timeout, comet, indirection-encoding-disable, iona, lax-boolean-encoding, max-managed-buf-size, max-server-connections, max-threads, outbuf-cache-timeout, outbuf-size, queue-max, queue-min, poa-monitoring, print-version, retries, retry-interval, queue-wait, server-timeout, strict-check-on-tc-creation, use-bom, use-imr.

- The properties X use expressions. Configuration properties that are used to resolve those expressions should be transformed manually to the new iiop-openjdk subsystem format
  Admin has to transform all the configuration files to work correctly with the jacorb subsystem. f.e. jacorb has a property giop-minor-version whereas openjdk uses property giop-version. Let's suppose we use '1' minor version in jacorb and have it configured in standalone.conf file as system variable: -Diiop-giop-minor-version=1. Admin is responsible for changing this variable to 1.1 after the migration to make sure that the new subsystem will work correctly.

JBoss Web Subsystem

JBoss Web is replaced by Undertow in WildFly, which means that the legacy subsystem configuration should be migrated to WildFly's Undertow subsystem configuration.

JBoss Web Subsystem Configuration

The extension's module **org.jboss.as.web *is replaced by module *org.wildfly.extension.undertow**, while the subsystem configuration namespace **urn:jboss:domain:web:*** is replaced by **urn:jboss:domain:undertow:3.0**.

The XML configuration of the new subsystem is relatively different. Consider the following example of the JBoss Web subsystem configuration, containing all valid elements and attributes:

```
<?xml version="1.0" encoding="UTF-8"?>
<subsystem xmlns="urn:jboss:domain:web:2.2" default-virtual-server="default-host" native="true" default-
session-timeout="30" instance-id="foo">
    <configuration>
        <static-resources listings="true"
                          sendfile="1000"
                          file-encoding="utf-8"
                          read-only="true"
                          webdav="false"
                          secret="secret"
                          max-depth="5"
                          disabled="false"
                />
        <jsp-configuration development="true"
                           disabled="false"
                           keep-generated="true"
                           trim-spaces="true"
                           tag-pooling="true"
                           mapped-file="true"
                           check-interval="20"
                           modification-test-interval="1000"
                           recompile-on-fail="true"
                           smap="true"
                           dump-smap="true"
                           generate-strings-as-char-arrays="true"
                           error-on-use-bean-invalid-class-attribute="true"
                           scratch-dir="/some/dir"
                           source-vm="1.7"
                           target-vm="1.7"
                           java-encoding="utf-8"
                           x-powered-by="true"
                           display-source-fragment="true" />
        <mime-mapping name="ogx" value="application/ogg" />
        <welcome-file>titi</welcome-file>
    </configuration>
    <connector name="http" scheme="http"
               protocol="HTTP/1.1"
               socket-binding="http"
               enabled="true"
               enable-lookups="false"
               proxy-binding="reverse-proxy"
               max-post-size="2097153"
               max-save-post-size="512"
               redirect-binding="https"
               max-connections="300"
               secure="false"
               executor="some-executor"
            />
    <connector name="https" scheme="https" protocol="HTTP/1.1" secure="true" socket-binding="https">
        <ssl certificate-key-file="${file-base}/server.keystore"
             ca-certificate-file="${file-base}/jsse.keystore"
             key-alias="test"
             password="changeit"
             cipher-suite="SSL_RSA_WITH_3DES_EDE_CBC_SHA"
             protocol="SSLv3"
             verify-client="true"
             verify-depth="3"
             certificate-file="certificate-file.ext"
             ca-revocation-url="https://example.org/some/url"
             ca-certificate-password="changeit"
             keystore-type="JKS"
             truststore-type="JKS"
             session-cache-size="512"
             session-timeout="3000"
             ssl-protocol="RFC4279"
                />
    </connector>
    <connector name="http-vs" scheme="http" protocol="HTTP/1.1" socket-binding="http" >
        <virtual-server name="vs1" />
        <virtual-server name="vs2" />
    </connector>
    <virtual-server name="default-host" enable-welcome-root="true" default-web-module="foo.war">
        <alias name="localhost" />
        <alias name="example.com" />
        <access-log resolve-hosts="true" extended="true" pattern="extended" prefix="prefix" rotate="true" >
            <directory relative-to="jboss.server.base.dir" path="toto" />
        </access-log>
        <rewrite name="myrewrite" pattern="^/helloworld(.*)" substitution="/helloworld/test.jsp" flags="L"
/>
        <rewrite name="with-conditions" pattern="^/helloworld(.*)" substitution="/helloworld/test.jsp"
flags="L" >
            <condition name="https" pattern="off" test="%{HTTPS}" flags="NC"/>
            <condition name="user" test="%{USER}" pattern="toto" flags="NC"/>
            <condition name="no-flags" test="%{USER}" pattern="toto"/>
        </rewrite>
        <sso reauthenticate="true" domain="myDomain" cache-name="myCache"
             cache-container="cache-container" http-only="true"/>
    </virtual-server>
    <virtual-server name="vs1" />
    <virtual-server name="vs2" />
    <valve name="myvalve" module="org.jboss.some.module" class-name="org.jboss.some.class" enabled="true">
        <param param-name="param-name" param-value="some-value"/>
    </valve>
    <valve name="accessLog" module="org.jboss.as.web" class-
name="org.apache.catalina.valves.AccessLogValve">
        <param param-name="prefix" param-value="myapp_access_log." />
        <param param-name="suffix" param-value=".log" />
        <param param-name="rotatable" param-value="true" />
        <param param-name="fileDateFormat" param-value="yyyy-MM-dd" />
        <param param-name="pattern" param-value="common" />
        <param param-name="directory" param-value="${jboss.server.log.dir}" />
        <param param-name="resolveHosts" param-value="false"/>
        <param param-name="conditionIf" param-value="log-enabled"/>
    </valve>
    <valve name="request-dumper" module="org.jboss.as.web" class-
name="org.apache.catalina.valves.RequestDumperValve"/>
    <valve name="remote-addr" module="org.jboss.as.web" class-
name="org.apache.catalina.valves.RemoteAddrValve">
        <param param-name="allow" param-value="127.0.0.1,127.0.0.2" />
        <param param-name="deny" param-value="192.168.1.20" />
    </valve>
    <valve name="crawler" class-name="org.apache.catalina.valves.CrawlerSessionManagerValve"
module="org.jboss.as.web" >
        <param param-name="sessionInactiveInterval" param-value="1" />
        <param param-name="crawlerUserAgents" param-value="Google" />
    </valve>
    <valve name="proxy" class-name="org.apache.catalina.valves.RemoteIpValve" module="org.jboss.as.web" >
        <param param-name="internalProxies" param-value="192\.168\.0\.10|192\.168\.0\.11" />
        <param param-name="remoteIpHeader" param-value="x-forwarded-for" />
        <param param-name="proxiesHeader" param-value="x-forwarded-by" />
        <param param-name="trustedProxies" param-value="proxy1|proxy2" />
    </valve>
</subsystem>
```

FIXME compare with Undertow, list unsupported features

It's possible to do a migration of the legacy subsystem configuration, and related persisted data. , by invoking the legacy's subsystem's `migrate` operation, using the CLI management client:

```
/subsystem=web:migrate
```

There is also a `describe-migration` operation that returns a list of all the management operations that are performed to migrate from the legacy subsystem to the new one:

```
/subsystem=web:describe-migration
```

Both `migrate` and `describe-migration` will also display a list of migration-warnings if there are some resource or attributes that can not be migrated automatically. The following is a list of these warnings:

- Could not migrate resource X
  This warning means that mentioned resource configuration is not supported and won't be included in the new subsystem configuration. As a result of that admin must be aware that any behaviour implied by those resources would be inexistent. Admin has to check whether subsystem is able to operate correctly without that behaviour on the new server.
  FIXME must document which are the resources that trigger this
- Could not migrate attribute X from resource Y.
  This warning means that mentioned resource configuration property is not supported and won't be included in the new subsystem configuration. As a result of that admin must be aware that any behaviour implied by those properties would be inexistent. Admin has to check whether subsystem is able to operate correctly without that behaviour on the new server.
  FIXME must document which are the properties that trigger this

- Could not migrate SSL connector as no SSL config is defined
- Could not migrate verify-client attribute %s to the Undertow equivalent
- Could not migrate verify-client expression %s
- Could not migrate valve X
  This warning means that mentioned valve configuration is not supported and won't be included in the new subsystem configuration. As a result of that admin must be aware that any behaviour implied by those resources would be inexistent. Admin has to check whether subsystem is able to operate correctly without that behaviour on the new server. This warning may happen for :
  - org.apache.catalina.valves.RemoteAddrValve : must have at least one allowed or denied value.
  - org.apache.catalina.valves.RemoteHostValve : must have at least one allowed or denied value.
  - org.apache.catalina.authenticator.BasicAuthenticator
  - org.apache.catalina.authenticator.DigestAuthenticator
  - org.apache.catalina.authenticator.FormAuthenticator
  - org.apache.catalina.authenticator.SSLAuthenticator
  - org.apache.catalina.authenticator.SpnegoAuthenticator
  - custom valves
- Could not migrate attribute X from valve Y
  This warning means that mentioned valve configuration property is not supported and won't be included in the new subsystem configuration. As a result of that admin must be aware that any behaviour implied by those properties would be inexistent. Admin has to check whether subsystem is able to operate correctly without that behaviour on the new server. This warning may happen for :
  - org.apache.catalina.valves.AccessLogValve : if you use the following parameters *resolveHosts*, *fileDateFormat*, *renameOnRotate*, *encoding*, *locale*, *requestAttributesEnabled*, *buffered*.
  - org.apache.catalina.valves.ExtendedAccessLogValve : if you use the following parameters *resolveHosts*, *fileDateFormat*, *renameOnRotate*, *encoding*, *locale*, *requestAttributesEnabled*, *buffered*.
  - org.apache.catalina.valves.RemoteIpValve:
    - if *remoteIpHeader* is defined and isn't set to "x-forwarded-for".
    - if *protocolHeader* is defined and isn't set to "x-forwarded-proto".
    - if you use the following parameters *httpServerPort* and *httpsServerPort* .

Also, please note that Undertow doesn't support JBoss Web **valves**, but some of these may be migrated to Undertow handlers, and JBoss Web subsystem's `migrate` operation do that too.

Here is a list of those valves and their corresponding Undertow handler:

Valve	**Handler**
org.apache.catalina.valves.AccessLogValve	io.undertow.server.handlers.accesslog.AccessLogHandler
org.apache.catalina.valves.ExtendedAccessLogValve	io.undertow.server.handlers.accesslog.AccessLogHandler
org.apache.catalina.valves.RequestDumperValve	io.undertow.server.handlers.RequestDumpingHandler
org.apache.catalina.valves.RewriteValve	io.undertow.server.handlers.SetAttributeHandler
org.apache.catalina.valves.RemoteHostValve	io.undertow.server.handlers.AccessControlListHandler
org.apache.catalina.valves.RemoteAddrValve	io.undertow.server.handlers.IPAddressAccessControlHandler
org.apache.catalina.valves.RemoteIpValve	io.undertow.server.handlers.ProxyPeerAddressHandler
org.apache.catalina.valves.StuckThreadDetectionValve	io.undertow.server.handlers.StuckThreadDetectionHandler
org.apache.catalina.valves.CrawlerSessionManagerValve	io.undertow.servlet.handlers.CrawlerSessionManagerHandler

The **org.apache.catalina.valves.JDBCAccessLogValve** can't be automatically migrated to **io.undertow.server.handlers.JDBCLogHandler** as the expectations differ.
The migration can be done manually thought :

1. create the driver module and add the driver to the list of available drivers
2. create a datasource pointing to the database where the log entries are going to be stored
3. add an **expression-filter** definition with the following expression: "jdbc-access-log(datasource='datasource-jndi-name')

```
<valve name="jdbc" module="org.jboss.as.web" class-
name="org.apache.catalina.valves.JDBCAccessLogValve">
    <param param-name="driverName" param-value="com.mysql.jdbc.Driver" />
    <param param-name="connectionName" param-value="root" />
    <param param-name="connectionPassword" param-value="password" />
    <param param-name="connectionURL" param-value="jdbc:mysql://localhost:3306/wildfly?
zeroDateTimeBehavior=convertToNull" />
    <param param-name="format" param-value="combined" />
</valve>
```

should become:

```
<subsystem xmlns="urn:jboss:domain:datasources:1.2">
    <datasources>
        <datasource jndi-name="java:jboss/datasources/accessLogDS" pool-name="ccessLogDS"
enabled="true" use-java-context="true">
            <connection-url>jdbc:mysql://localhost:3306/wildfly?
zeroDateTimeBehavior=convertToNull</connection-url>
            <driver>mysql</driver>
            <security>
               <user-name>root</user-name>
               <password>password</password>
            </security>
        </datasource>
...
        <drivers>
            <driver name="mysql" module="com.mysql">
                <driver-class>com.mysql.jdbc.Driver</driver-class>
            </driver>
...
        </drivers>
    </datasources>
</subsystem>
...
<subsystem xmlns="urn:jboss:domain:undertow:3.1" default-virtual-host="default-virtual-host" default-
servlet-container="myContainer"
           default-server="some-server" instance-id="some-id" statistics-enabled="true">
...
    <server name="some-server" default-host="other-host" servlet-container="myContainer">
...
        <host name="other-host" alias="www.mysite.com, ${prop.value:default-alias}" default-web-
module="something.war" disable-console-redirect="true">
            <location name="/" handler="welcome-content" />
            <filter-ref name="jdbc-access"/>
        </host>
    </server>
...
    <filters>
        <expression-filter name="jdbc-access" expression="jdbc-access-
log(datasource='java:jboss/datasources/accessLogDS')" />
...
    </filters>

</subsystem>
```

Please note that any custom valve won't be migrated at all and will just be removed from the configuration.
Also the authentication related valves are to be replaced by Undertow authentication mechanisms, and this have to be done manually.

FIXME how this last “manual” replacement is done? Need whole process documented and concrete example

WebSockets

In AS7, to use WebSockets, you had to configure the 'http' **connector** in the **web** subsystem of the server configuration file to use the NIO2 protocol. The following is an example of the Management CLI command to configure WebSockets in the previous releases.

```
/subsystem=web/connector=http/:write-
attribute(name=protocol,value=org.apache.coyote.http11.Http11NioProtocol)
```

WebSockets are a requirement of the Java EE 7 specification and the default configuration is included in WildFly. More complex WebSocket configuration is done in the **servlet-container** of the **undertow** subsystem of the server configuration file.

You no longer need to configure the server for default WebSocket support.
FIXME isn't <websockets /> required for that?

Messaging Subsystem

WildFly JMS support is provided by ActiveMQ Artemis, instead of HornetQ. It's possible to do a migration of the legacy subsystem configuration, and related persisted data.

Messaging Subsystem Configuration

The extension's module **org.jboss.as.messaging** is replaced by module **org.wildfly.extension.messaging-activemq**, while the subsystem configuration namespace **urn:jboss:domain:messaging:3.0** is replaced by **urn:jboss:domain:messaging-activemq:1.0**.

Management model

In most cases, an effort was made to keep resource and attribute names as similar as possible to those used in previous releases. The following table lists some of the changes.

HornetQ name	ActiveMQ name
hornetq-server	server
hornetq-serverType	serverType
connectors	connector
discovery-group-name	discovery-group

The management operations invoked on the new messaging-subsystem starts with **`/subsystem=messaging-activemq/server=X`** while the legacy messaging subsystem was at `/subsystem=messaging/hornetq-server=X`.

In case the legacy subsystem configuration is available, such configuration may be migrated to the new subsystem by invoking its `migrate` operation, using the CLI management client:

```
/subsystem=messaging:migrate
```

There is also a `describe-migration` operation that returns a list of all the management operations that are performed to migrate from the legacy subsystem to the new one:

```
/subsystem=messaging:describe-migration
```

Both `migrate` and `describe-migration` will also display a list of migration-warnings if there are some resource or attributes that can not be migrated automatically. The following is a list of these warnings:

- The migrate operation can not be performed: the server must be in admin-only mode
  The `migrate` operation requires starting the server in admin-only mode, which is done by adding parameter `--admin-only` to the server start command, e.g.
  ```
  ./standalone.sh --admin-only
  ```
- Can not migrate attribute local-bind-address from resource X. Use instead the socket-attribute to configure this broadcast-group.
- Can not migrate attribute local-bind-port from resource X. Use instead the socket-binding attribute to configure this broadcast-group.
- Can not migrate attribute group-address from resource X. Use instead the socket-binding attribute to configure this broadcast-group.
- Can not migrate attribute group-port from resource X. Use instead the socket-binding attribute to configure this broadcast-group.
  Broadcast-group resources no longer accept local-bind-address, local-bind-port, group-address, group-port attributes. It only accepts a socket-binding. The warning notifies that resource X has an unsupported attribute. The user will have to set the socket-binding attribute on the resource and ensures it corresponds to a defined socket-binding resource.

- Classes providing the %s are discarded during the migration. To use them in the new messaging-activemq subsystem, you will have to extend the Artemis-based Interceptor.
  Messaging interceptors support is significantly different in WildFly 10, any interceptors configured in the legacy subsystem are discarded during migration. Please refer to the Messaging Interceptors section to learn how to migrate legacy Messaging interceptors.

- Can not migrate the HA configuration of X. Its shared-store and backup attributes holds expressions and it is not possible to determine unambiguously how to create the corresponding ha-policy for the messaging-activemq's server.
  If the hornetq-server X’s shared-store or backup attributes hold an expression, such as ${xxx}, then it’s not possible to determine the actual ha-policy of the migrated server. In that case, we discard it and the user will have to add the correct ha-policy afterwards (ha-policy is a single resource underneath the messaging-activemq's server resource).

- Can not migrate attribute local-bind-address from resource X. Use instead the socket-binding attribute to configure this discovery-group.Can not migrate attribute local-bind-port from resource X. Use instead the socket-binding attribute to configure this discovery-group.

- Can not migrate attribute group-address from resource X. Use instead the socket-binding attribute to configure this discovery-group.
- Can not migrate attribute group-port from resource X. Use instead the socket-binding attribute to configure this discovery-group.
  discovery-group resources no longer accept local-bind-address, local-bind-port, group-address, group-port attributes. It only accepts a socket-binding. The warning notifies that resource X has an unsupported attribute.
  The user will have to set the socket-binding attribute on the resource and ensures it corresponds to a defined socket-binding resource.

- Can not create a legacy-connection-factory based on connection-factory X. It uses a HornetQ in-vm connector that is not compatible with Artemis in-vm connector
  Legacy subsystem's remote connection-factory resources are migrated into legacy-connection-factory resources, to allow old EAP6 clients to connect to EAP7. However a connection-factory using in-vm will not be migrated, because a in-vm client will be based on EAP7, not EAP 6. In other words, legacy-connection-factory are created only when the CF is using remote connectors, and this warning notifies about in-vm connection-factory X not migrated.

- Can not migrate attribute X from resource Y. The attribute uses an expression that can be resolved differently depending on system properties. After migration, this attribute must be added back with an actual value instead of the expression.
  This warning appears when the migration logic needs to know the concrete value of attribute X during migration, but instead such value includes an expression that's can't be resolved, so the actual value can not be determined, and the attribute is discarded. It happens in several cases, for instance:
  - cluster-connection forward-when-no-consumers. This boolean attribute has been replaced by the message-load-balancing-type attribute (which is an enum of OFF, STRICT, ON_DEMAND)
  - broadcast-group and discovery-group's jgroups-stack and jgroups-channel attributes. They reference other resources and we no longer accept expressions for them.

- Can not migrate attribute X from resource Y. This attribute is not supported by the new messaging-activemq subsystem.
  Some attributes are no longer supported in the new messaging-activemq subsystem and are simply discarded:
  - hornetq-server's failback-delay
  - http-connector's use-nio attribute
  - http-acceptor's use-nio attribute
  - remote-connector's use-nio attribute
  - remote-acceptor's use-nio attribute

XML Configuration

The XML configuration has changed significantly with the new messaging-activemq subsystem to provide a XML scheme more consistent with other WildFly subsystems.
It is not advised to change the XML configuration of the legacy messaging subsystem to conform to the new messaging-activemq subsystem. Instead, invoke the legacy subsystem `migrate` operation. This operation will write the XML configuration of the new **messaging-activemq** subsystem as a part of its execution.

Messaging Interceptors

Messaging Interceptors are significantly different in EAP 7, requiring both code and configuration changes by the user. In concrete the interceptor base Java class is now **org.apache.artemis.activemq.api.core.interceptor.Interceptor**, and the user interceptor implementation classes may now be loaded by any server module. Note that prior to EAP 7 the interceptor classes could only be installed by adding these to the HornetQ module, thus requiring the user to change such module XML descriptor, its **module.xml**.
With respect to the server XML configuration, the user must now specify the module to load its interceptors in the new **messaging-activemq** subsystem XML config, e.g:

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:1.0">
    <server name=“default”>
       ...
        <incoming-interceptors>
            <class name="org.foo.incoming.myInterceptor" module="org.foo" />
            <class name="org.bar.incoming.myOtherInterceptor" module="org.bar" />
        </incoming-interceptors>
        <outgoing-interceptors>
            <class name="org.foo.outgoing.myInterceptor" module="org.foo" />
            <class name="org.bar.outgoing.myOtherInterceptor" module="org.bar" />
        </outgoing-interceptors>
   </server>
</subsystem>
```

JMS Destinations

In previous releases, JMS destination queues were configured in the <jms-destinations> element under the hornetq-server section of the **messaging** subsystem.

```
<jms-destinations>
<jms-queue name="testQueue">
<entry name="queue/test"/>
<entry name="java:jboss/exported/jms/queue/test"/>
</jms-queue>
</jms-destinations>
```

In WildFly, the JMS destination queue is configured in the default server of the messaging-activemq subsystem.

```
<jms-queue name="testQueue" entries="queue/test java:jboss/exported/jms/queue/test"/>
```

Messaging Logging

The prefix of messaging log messages in WildFly is **WFLYMSGAMQ**, instead of **WFLYMSG**.

Messaging Data

The location of the messaging data has been changed in the new messaging-activemq subsystem:

- messagingbindings/ -> activemq/bindings/
- messagingjournal/ -> activemq/journal/
- messaginglargemessages/ -> activemq/largemessages/
- messagingpaging/ -> activemq/paging/

To migrate legacy messaging data, you will have to export the directories used by the legacy messaging subsystem and import them into the new subsystem's server by using its `import-journal operation:`

```
/subsystem=messaging-activemq/server=default:import-journal(file=<path to XML dump>)
```

The XML dump is a XML file generated by HornetQ `XmlDataExporter` util class.

Application Migration

Before you migrate your application, you should be aware that some features that were available in previous releases are now deprecated or missing.

EJBs

CMP Entity EJBs

Container-Managed Persistence entity beans support is optional in Java EE 7, and WildFly does not provide support for these.

CMP entity beans are defined in the **ejb-jar.xml** descriptor, in concrete an entity bean is CMP only if the **<entity/>**'s child element named **persistence-type** is included and has a value of **Container**. An example:

```
<?xml version="1.1" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/ejb-
jar_3_1.xsd"
         version="3.1">
    <enterprise-beans>
        <entity>
            <ejb-name>SimpleBMP</ejb-name>
            <local-home>org.jboss.as.test.integration.ejb.entity.bmp.BMPLocalHome</local-home>
            <local>org.jboss.as.test.integration.ejb.entity.bmp.BMPLocalInterface</local>
            <ejb-class>org.jboss.as.test.integration.ejb.entity.bmp.SimpleBMPBean</ejb-class>
            <persistence-type>Container</persistence-type>
            <prim-key-class>java.lang.Integer</prim-key-class>
            <reentrant>true</reentrant>
        </entity>
    </enterprise-beans>
</ejb-jar>
```

CMP entity beans should be replaced by JPA entities.

EJB Client

Default Remote Connection Port

The default remote connection port has changed from '4447' to '8080'.

In JBoss AS7, the **jboss-ejb-client.properties** file looked similar to the following:

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port=4447
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

In WildFly, the properties file looks like this:

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port=8080
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

Default Connector

In WildFly, the default connector has changed from "remoting" to "http-remoting". This change impacts clients that use libraries from one release of JBoss and to connect to server in a different release.

- If a client application uses the EJB client library from JBoss AS 7 and wants to connect to WildFly 10 server, the server must be configured to expose a remoting connector on a port other than "8080". The client must then connect using that newly configured connector.
- A client application that uses the EJB client library from WildFly 10 and wants to connect to a JBoss AS 7 server must be aware that the server instance does not use the http-remoting connector and instead uses a remoting connector. This is achieved by defining a new client-side connection property.

  ```
  remote.connection.default.protocol=remote
  ```

External applications using JNDI, to remotely lookup up EJBs in a WildFly 10 server, may also need to be migrated, please refer to Remote JNDI Clients section for further information.

JMS

Proprietary JMS Resource Definitions

The proprietary XML descriptors, previously used to setup JMS resources, are deprecated in WildFly. Java EE 7 (section EE.5.18) standardised such functionality.

The deprecated descriptors are files bundled in the application package, which name ends with **-jms.xml**.
Their namespace has been changed to **urn:jboss:messaging-activemq-deployment:1.0**.

External JMS Clients

JMS Resources are remotely looked up using JNDI, and looking up resources in a WildFly 10 server may require changes in the application code, please refer to Remote JNDI Clients section for further information.

JPA (and Hibernate)

Applications That Plan to Use Hibernate ORM 5.0

WildFly ships with Hibernate ORM 5.0 and those libraries are implicitly added to the application classpath when a persistence.xml is detected during deployment. If your application uses JPA, it will default to using the Hibernate ORM 5.0 libraries.

Hibernate ORM 5.0 introduces:

- Redesigned metamodel - Complete replacement for the current org.hibernate.mapping code
- Query parser - Improved query parser based on Antlr 3/4
- Multi-tenancy improvements - Discriminator-based multi-tenancy
- Follow-on fetches - Two-phase loading via LoadPlans/EntityGraphs

Applications that currently use Hibernate ORM 4.0 - 4.3

If your application needs second-level cache enabled, you should migrate to Hibernate ORM 5.0, which is integrated with Infinispan 8.0. Applications written with Hibernate ORM 4.x can still use Hibernate 4.x if you define a custom JBoss module with Hibernate 4.x JARs and exclude the Hibernate 5 classes from your application. It is recommended that you migrate your application to use Hibernate 5.

For information about the changes implemented between Hibernate 4 and Hibernate 5, see https://github.com/hibernate/hibernate-orm/blob/master/migration-guide.adoc

Applications that currently use Hibernate 3

The integration classes that made it easier to use Hibernate 3 in AS 7 were removed from WildFly 10. If your application still uses Hibernate 3 libraries, it is strongly recommended that you migrate your application to use Hibernate 5 as Hibernate 3 will no longer work in WildFly without a lot of effort. If you can not migrate to Hibernate 5, you must define a custom JBoss Module for the Hibernate 3 classes and exclude the Hibernate 5 classes from your application.

Web Applications

JBoss Web Valves

Undertow does not support the JBoss Web Valve functionality. This can be replaced by Undertow Handlers (see http://undertow.io/undertow-docs/undertow-docs-1.3.0/index.html#undertow-handler-authors-guide for more).

List of valves that were provided with JBoss Web, together with a corresponding Undertow handler, is provided above, in the section on the JBoss Web subsystem.

JBoss Web Valves are specified in the proprietary **jboss-web.xml** descriptor, through **<valve />** element(s). These can be replaced using the **<http-handler />** element(s). For example:

```
<jboss-web>
    <valve>
        <class-name>org.apache.catalina.valves.RequestDumperValve</class-name>
        <module>org.jboss.as.web</module>
    </valve>
</jboss-web>
```

can be replaced by

```
<jboss-web>
    <http-handler>
        <class-name>io.undertow.server.handlers.RequestDumpingHandler</class-name>
        <module>io.undertow.core</module>
    </http-handler>
</jboss-web>
```

Web Services

CXF Spring Webservices

The setup of web service's endpoints and clients, through a Spring XML descriptor, driving a CXF bus creation, is no longer supported in WildFly.

Any application containing a jbossws-cxf.xml must migrate all functionality specified in such XML descriptor, mostly already supported by the JAX-WS specification, included in Java EE 7. It is still possible to rely on direct Apache CXF API usage, loosing the Java EE portability of the application, for instance when specific Apache CXF functionalities are needed. Please refer to the Apache CXF Integration document for further information.

JAX-RPC

JAX-RPC is an API for building Web services and clients that used remote procedure calls (RPC) and XML, which was deprecated in Java EE 6, and is no longer supported by WildFly.

JAX-RPC Web Services may be identified by the presence of the XML descriptor named web services.xml, containing a **<webservice-description/>** element that includes a child element named **<jaxrpc-mapping-file/>**. An example:

```
<webservices xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://www.ibm.com/webservices/xsd/j2ee_web_services_1_1.xsd" version="1.1">
    <webservice-description>
        <webservice-description-name>HelloService</webservice-description-name>
        <wsdl-file>WEB-INF/wsdl/HelloService.wsdl</wsdl-file>
        <jaxrpc-mapping-file>WEB-INF/mapping.xml</jaxrpc-mapping-file>
        <port-component>
            <port-component-name>Hello</port-component-name>
            <wsdl-port>HelloPort</wsdl-port>
            <service-endpoint-interface>org.jboss.chap12.hello.Hello</service-endpoint-interface>
            <service-impl-bean>
                <servlet-link>HelloWorldServlet</servlet-link>
            </service-impl-bean>
        </port-component>
    </webservice-description>
</webservices>
```

Applications using JAX-RPC should be migrated to use JAX-WS, the current Java EE standard web service framework.

JAX-RS 2.0

JSR 339: JAX-RS 2.0: The Java API for RESTful Web Services specification is located here: https://jcp.org/en/jsr/detail?id=339

Some changes to the `MessageBodyWriter` interface may represent a backward incompatible change with respect to JAX-RS 1.X.

Be sure to define an @Produces or @Consumes for your endpoints. Failure to do so may result in an error similar to the following.

```
org.jboss.resteasy.core.NoMessageBodyWriterFoundFailure: Could not find MessageBodyWriter for response
object of type: <OBJECT> of media type: <CONTENT_TYPE>
```

REST Client API

Some REST Client API classes and methods are deprecated, for example: org.jboss.resteasy.client.ClientRequest and org.jboss.resteasy.client.ClientResponse. Instead, use org.jboss.resteasy.client.jaxrs.ResteasyClient and javax.ws.rs.core.Response. See the `resteasy-jaxrs-client quickstart` for an example of an external JAX-RS RestEasy client that interacts with a JAX-RS Web service.

Application Clustering

HA Singleton

JBoss AS7 introduced singleton services - a mechanism for installing an service such that it would only start on one node in the cluster at a time, a HA Singleton. Such mechanism required usage of a private JBoss EAP Clustering API, designed around the class **org.jboss.as.clustering.singleton.SingletonService**, and was documented in detail at https://access.redhat.com/documentation/en-US/JBoss_Enterprise_Application_Platform/6.4/html/Development_Guide/Implement_an_HA_Singleton.html , and while not difficult to implement, the installation process suffered from a couple shortcomings:

- Installing multiple singleton services within a single deployment caused the deployer to hang.
- Installing a singleton service required the user to specify several private module dependencies in /META-INF/MANIFEST.MF

WildFly 10 introduces a new public API for building such services, which significantly simplifies the process, and solves the issues found in the legacy solution. The JBoss EAP 7 Quickstart application named **cluster-ha-singleton** examples a HA Singleton implementation using the new API, and may be found at https://github.com/jboss-developer/jboss-eap-quickstarts/tree/7.0.x-develop/cluster-ha-singleton .
FIXME: community URLs instead

Stateful Session EJB Clustering

WildFly 10 no longer requires Stateful Session EJBs to use the **org.jboss.ejb3.annotation.Clustered** annotation to enable clustering behaviour. By default, if the server is started using an HA profile, the state of your SFSBs will be replicated automatically. Disabling this behaviour is achievable on a per-EJB basis, by annotating your bean using **@Stateful(passivationCapable=false)**, which is new to the EJB 3.2 specification; or globally through the configuration of the EJB3 subsystem, in the server configuration.

Note that the **@Clustered** annotation, if used by an application, is simply ignored, the application deployment will not fail.

Web Session Clustering

WildFly 10 introduces a new web session clustering implementation, replacing the one found in AS7, which has been around for ages (since JBoss AS 3.2!), and was tightly coupled to the legacy JBoss Web subsystem source code. The most relevant changes in the new implementation are:

- Introduction of a proper session manager SPI, and an Infinispan implementation of it, decoupled from the web subsystem implementation
- Sessions are implemented as a facade over one or more cache entries, which means that the container's session manager itself does not retain a separate reference to each HttpSession
- Pessimistic locking of cache entries effectively ensures that only a single client on a single node ever accesses a given session at any given time
- Usage of cache entry grouping, instead of atomic maps, to ensure that multiple cache entries belonging to the same session are co-located.
- Session operations within a request only ever use a single batch/transaction. This results in fewer RPCs per request.
- Support for write-through cache stores, as well as passivation-only cache stores.

With respect to applications, the new web session clustering implementation deprecates/reinterprets much of the related configuration, which is included in JBoss's proprietary web application XML descriptor, **jboss-web.xml**:

- **<max-active-sessions/>**
  Previously, session creation would fail if an additional session would cause the number of active sessions to exceed the value specified by <**max-active-sessions/>**.
  In the new implementation, **<max-active-sessions/>** is used to enable session passivation. If session creation would cause the number of active sessions to exceed **<max-active-sessions/>**, then the oldest session known to the session manager will passivate to make room for the new session.

- **<passivation-config/>**
  This configuration element and its sub-elements are no longer used in WildFly.

- **<use-session-passivation/>**
  Previously, passivation was enabled via this attribute, yet in the new implementation, passivation is enabled by specifying a non-negative value for **<max-active-sessions/>**.

- **<passivation-min-idle-time/>**
  Previously, sessions needed to be active for at least a specific amount of time before becoming a candidate for passivation. This could cause session creation to fail, even when passivation was enabled.
  The new implementation does not support this logic and thus avoids this DoS vulnerability.

- **<passivation-max-idle-time/>**
  Previously, a session would be passivated after it was idle for a specific amount of time.
  The new implementation does not support eager passivation - only lazy passivation. Sessions are only passivated when necessary to comply with **<max-active-sessions/>**.

- **<replication-config/>**
  The new implementation deprecates a number of sub-elements:

- **<replication-trigger/>**
  Previously, session attributes could be treated as either mutable or immutable depending on the values specified by **<replication-trigger/>**:
  - SET treated all attributes as immutable, requiring a separate HttpSession.setAttribute(...) to indicate that the value changed.
  - SET_AND_GET treated all session attributes as mutable.
  - SET_AND_NON_PRIMITIVE_GET recognised a small set of types (i.e. strings and boxed primitives) as immutable, and assumed that any other attribute was mutable.
    The new implementation replaces this configuration option with a single, robust strategy. Session attributes are assumed to be mutable unless one of the following is true:
  - The value is a known immutable value:
    - null
    - java.util.Collections.EMPTY_LIST, EMPTY_MAP, EMPTY_SET
  - The value type is or implements a known immutable type:
    - Boolean, Byte, Character, Double, Float, Integer, Long, Short
    - java.lang.Enum, StackTraceElement, String
    - java.io.File, java.nio.file.Path
    - java.math.BigDecimal, BigInteger, MathContext
    - java.net.InetAddress, InetSocketAddress, URI, URL
    - java.security.Permission
    - java.util.Currency, Locale, TimeZone, UUID
  - The value type is annotated with @org.wildfly.clustering.web.annotation.Immutable

- **<use-jk/>**
  Previously, the instance-id of the node handling a given request was appended to the jsessionid (for use by load balancers such as mod_jk, mod_proxy_balancer, mod_cluster, etc.) depending on the value specified for **<use-jk/>**. In the new implementation, the instance-id, if defined, is always appended to the jsessionid.

- **<max-unreplicated-interval/>**
  Previously, this configuration option was an optimization that would prevent the replicate of a session's timestamp if no session attribute was changed. While this sounds nice, in practice it doesn't prevent any RPCs, since session access requires cache transaction RPCs regardless of whether any session attributes changed. In the new implementation, the timestamp of a session is replicated on every request. This prevents stale session meta data following failover.

- **<snapshot-mode/>**
  Previously, one could configure **<snapshot-mode/>** as INSTANT or INTERVAL. Infinispan's replication queue renders this configuration option obsolete.

- **<snapshot-interval/>**
  Only relevant for **<snapshot-mode>INTERVAL</snapshot-mode>**. See above.

- **<session-notification-policy/>**
  Previously, the value defined by this attribute defined a policy for triggering session events. In the new implementation, this behaviour is spec-driven and not configurable.

Other Specifications and Frameworks

Remote JNDI Clients

WildFly 10's default JNDI Provider URL has changed, which means that external applications, using JNDI to lookup remote resources, for instance an EJB or a JMS Queue, may need to change the value for the JNDI **InitialContext** environment's property named **java.naming.provider.url**. The default URL scheme is now **http-remoting**, and the default URL port is now **8080**.

As an example, considering the application server host is **localhost**, then clients previously accessing JBoss AS7 would use

```
java.naming.factory.initial=org.jboss.naming.remote.client.InitialContextFactory
java.naming.provider.url=remote://localhost:4447
```

while clients now accessing WildFly should use instead

```
java.naming.factory.initial=org.jboss.naming.remote.client.InitialContextFactory
java.naming.provider.url=http-remoting://localhost:8080
```

JSR-88

The specification which aimed to standardise deployment tasks got very little adoption, due to much more "feature rich" proprietary solutions already included in every vendor application server. It was no surprise that JSR-88 support was dropped from Java EE 7, and WildFly followed that and dropped support too.

A JSR-88 deployment plan is identified by a XML descriptor named **deployment-plan.xml**, bundled in a zip/jar archive.

Module Dependencies

Applications defining dependencies to WildFly modules, through the application's package MANIFEST.MF or jboss-deployment-structure.xml, may be referencing missing modules. When migrating an application, relying on such functionality, the presence of the referenced modules should be validated in advance.

6 Comments

Sande Gilda Dec 04, 2015

Eduardo, re "Scoped EJB Client Contexts". See https://issues.jboss.org/browse/JBEAP-2080. It states that this section is not correct and that scoped EJB client contexts are still in WildFly 10.

Ladislav Thon Dec 07, 2015

The value in `remote.connection.default.protocol=remoting` (name of the remoting protocol) should in fact be `remote`, no?

Sande Gilda Dec 07, 2015

Good catch! Yes, you are correct. Fixed!

Marek Novotny Feb 29, 2016

> *<snapshot-interval/>*

Only relevant for <snapshot-mode>INTERVAL</snapshot-mode>. See above.
The explanation is misleading. There should be written that since `<snapshot-mode>` is no longer valid, this option is no longer valid as well and not pointing to "see above".

Mahesh G Apr 06, 2016

Quick question, can this document be used to migration application from AS7 to any WildFly version or only to WildFly10 ?

Thanks!!

Sande Gilda Apr 06, 2016

I believe it targets migration to WildFly 10.