Upgrading Custom Java EE Applications from Oracle Application Server to WebLogic Server

Frances Zhao
Oracle Corporation
Portland, Oregon, United States

Keywords: Upgrade, Migration, Oracle Application Server, WebLogic

Introduction

As outlined in the Oracle Fusion Middleware upgrade strategy illustration below, Oracle WebLogic Server is the strategic application server infrastructure for the Oracle Fusion Middleware 11g platform. Oracle Internet Application Server will continue to be developed, maintained, and released according to Oracle's standard lifetime support policies. To ensure long-term investment protection and a seamless upgrade, key components such as Oracle HTTP Server, Oracle Web Cache, Oracle Forms, Reports, Portal, Discoverer, B2B and other layered products within Oracle Application Server (OracleAS) have been converged and certified on WebLogic Server.

As part of the Oracle Fusion Middleware 11gR1 release, Oracle has also delivered upgrade tooling to ensure that the upgrade for these products remains largely unaffected by the adoption of WebLogic Server. At a lower level, the Java server runtime underlying OracleAS, Oracle Containers for J2EE (OC4J), has also had key technologies integrated into WebLogic Server.

To enable the upgrade to WebLogic Server and the redeployment of custom Java EE applications from OC4J to the converged server infrastructure, customers will need to build a plan for the upgrade like they would for any major new application server release. Further, they will need to understand any changes to administrative processes associated with running those applications. To ensure that this impact remains minimal and well understood, Oracle has delivered a comprehensive set of documentation and tooling to guide customers through this process.

More specifically Oracle has delivered the WebLogic SmartUpgrade tool which greatly simplifies the upgrade of custom Java EE applications to WebLogic Server. The release of Oracle Fusion Middleware 11gR1 also includes an Upgrade Guide for Java EE that provides a detailed description of the work needed to upgrade existing OC4J based custom Java EE applications and their environments to WebLogic Server. The WebLogic SmartUpgrade tool and the Upgrade Guide for Java EE along with associated best practices as well as targeted consulting and educational offerings will allow for a systematic and seamless upgrade of custom Java EE applications to WebLogic Server and will facilitate any required adjustment to administrative processes.

The target audience of this paper is existing OracleAS 10g R3 customers who are responsible for the development of custom OC4J Java EE applications or the Upgrading Custom Java EE Applications from Oracle Application Server to WebLogic Server maintenance of an OC4J environment. The purpose of this paper is to introduce WebLogic Server concepts to these OC4J users and provide a preliminary understanding of the effort involved when upgrading custom Java EE applications from OC4J 10g R3 to WebLogic Server 11g (10.3.1). Key WebLogic Server terms are italicized on their first use and linked to their associated WebLogic Server documentation entry.

Benefit of Upgrading to WebLogic Server

Oracle WebLogic Server offers significant additional capabilities to the Oracle Application Server customer base beyond that which was available within Oracle Application Server including:

1. **Runtime**. Within the area of the core container, beyond the fully certified Java EE 5.0 compatibility, WebLogic Server extends the runtime in numerous areas including the Java Message Service (message ordering with unit of order, unit of work, scale out with distributed destinations and store and forward infrastructure, C and .NET JMS clients), Web services (conversational Web services, buffered Web services, asynchronous Web services, SOAP over JMS), built-in Tuxedo integration and runtime tuning (self tuning work managers) amongst others.
2. **Development**. Within the area of development capabilities above that in basic servers, WebLogic includes Java class FastSwap capabilities for fast test/debug cycles without server restarts, split development for tightly integrated IDE development, HTTP Publish/Subscribe Server for AJAX applications, filtering class loader to handle multiple versions of class libraries, Ant tasks for development, deployment and configuration automation and deep Eclipse and JDeveloper integration amongst others.
3. **Operations and Administration**. Within the area of operations and administration, WebLogic Server extends basic server capabilities in numerous areas including transactional and batch configuration, zero-downtime re-deployment, domain template builder and configuration wizard for simplified cloning, consistent scripting and command line tooling for all server configuration with WebLogic Scripting Tool, sophisticated lifecycle control over applications and the container, built in diagnostics framework amongst others.
4. **High Availability**. Within the area of high availability WebLogic Server extends basic core availability capabilities in a number of areas including clustered JNDI, whole server migration, service migration, high availability for singleton services (JMS, JTA, user defined) and rolling patch upgrades amongst others.

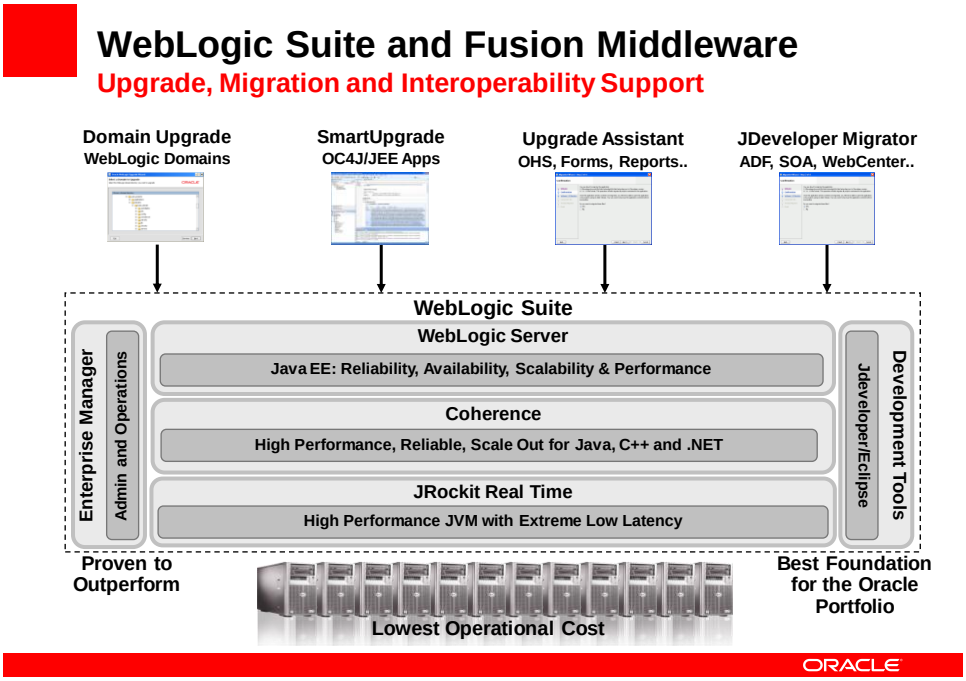

Illustration. 1: Oracle Fusion Middleware Upgrade Strategy

Upgrade Methodology

The upgrade of custom Java EE applications from OC4J to WebLogic Server should be executed within the context of a regular application enhancement project which preferably follows an established software development methodology and appropriate best practices. Figure 1 shows the categorization and flow of the specific upgrade activities that should be considered within the life-cycle of such a project.

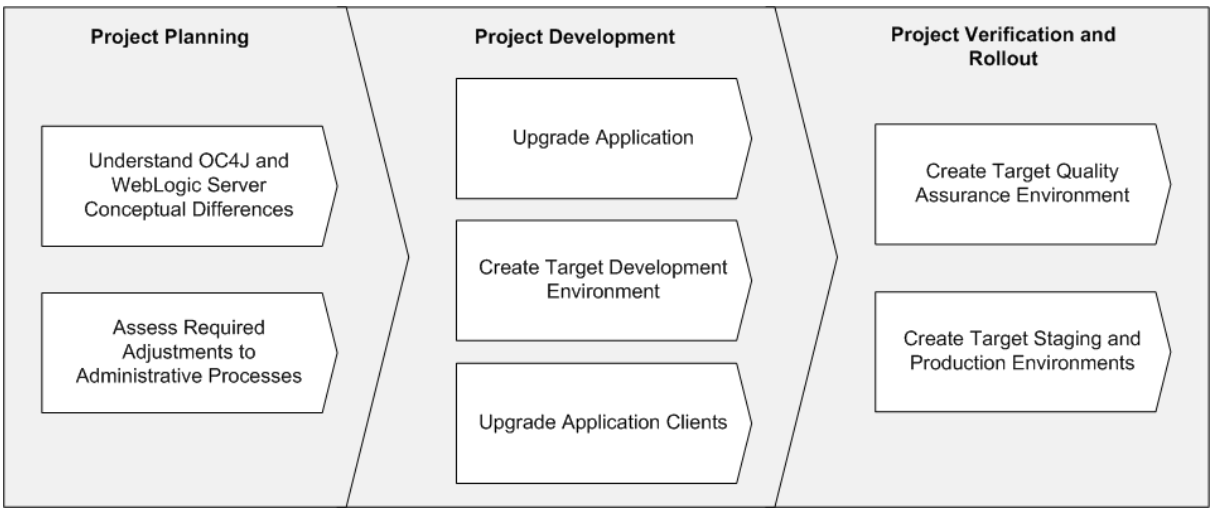

Figure 1 - OC4J to WebLogic Server Upgrade Activities within Project Life-Cycle

General WebLogic Server Concepts for OC4J Users

A good starting point for the comparison of WebLogic Server and OC4J Java EE container technologies is at the level of a server instance. In WebLogic, a WebLogic Server instance is a Java VM process executing the code that makes up the Oracle WebLogic Server container which provides all the APIs and services mandated by the Java EE specification as well as WebLogic specific extensions. OC4J users will recognize that in this way a WebLogic Server instance is very comparable to an OC4J instance. With this similarity in mind, this section begins by outlining some of the most fundamental conceptual differences between WebLogic Server and OC4J.

A WEBLOGIC SERVER INSTANCE IS ALWAYS PART OF A WEBLOGIC SERVER DOMAIN

WebLogic Server has no equivalent concept to a standalone or OracleAS OC4J configuration. Instead, a WebLogic Server instance has to always run within the context of a WebLogic Server domain. The closest equivalent OC4J concept to a WebLogic Server domain is a set of OC4J instances configured as part of the Oracle Process Manager and Notification (OPMN) service either within a single or multiple ORACLE_HOME(s). A domain is however a more general administrative grouping and configuration scoping mechanism which can include a set of servers and/or clusters. A domain's configuration is stored within a domain directory.

A WEBLOGIC SERVER DOMAIN IS MANAGED, CONFIGURED, AND MONITORED BY AN ADMINISTRATION SERVER

Unlike OC4J instances within a cluster which are managed, configured, and monitored through a combination of the Oracle Enterprise Manager Application Server Control and OPMN infrastructure, a WebLogic Server domain is managed, configured, and monitored through a central WebLogic Server instance called the domain administration (admin) server.
The admin server is just like any other WebLogic Server instance with the exception that it is configured with a distinctive set of applications which allow it to:

- Maintain a repository of configuration information for all servers within the domain and provide automatic synchronization capabilities for this information between the domain servers.
- Act as a centralized application deployment server for any subset of servers and clusters within the domain.
- Provide a browser-based administrative console application similar to the OC4J Oracle Enterprise Manager Application Server Control but spanning multiple servers and clusters which form the domain.

Other than the single admin server, all other WebLogic Server instances within the domain are called *managed servers*.
It is also worth noting that a domain could consist of a lone admin server (and no managed server), which would itself be used as a target for the deployment of Java EE applications. Such a scenario would closely resemble a standalone OC4J configuration.

A WEBLOGIC SERVER INSTALLATION CAN BE USED TO CONFIGURE MULTIPLE DOMAINS

An installation of OC4J is directly associated with a set of OC4J instance configurations which reside within sub-directories of the ORACLE_HOME directory containing the installation binaries. Not only is this direct file system association required, the relationship between the OC4J binaries and the configuration of instances is at a per instance level.
In contrast, WebLogic server provides a clear separation between the installed software and its different configuration instances. This separation is achieved through WebLogic Server domains. A WebLogic server installation can be used to create multiple domains – with different set of servers each - the configuration directory of which can reside anywhere within the file system. For a WebLogic Server instance to run from a domain directory, all that is required is for the WebLogic Server installation directory to be accessible. Note also that in the WebLogic Server model, the relationship between the WebLogic Server binaries and the configuration of instances is at a per domain (set of instances) level.

A WEBLOGIC SERVER INSTANCE IS ALWAYS ASSOCIATED WITH A SINGLE JAVA VM PROCESS

Using the OPMN numproc setting, an OC4J instance can be configured to run multiple copies of itself (with an identical configuration and deployed applications) on multiple Java VMs. There is no equivalent setting associated with a WebLogic Server instance which always runs on top of a single Java VM. However, the same outcome can be achieved by ensuring an equivalent configuration across multiple managed servers running on the same host.

A WEBLOGIC SERVER INSTANCE PROCESSES ALL APPLICATION REQUESTS ON THE SAME PORT BY DEFAULT

An OC4J instance uses a different set of listen ports for each protocol for which it can accept requests. OC4J "web sites" are for example used to configure specific HTTP, HTTPS, AJP, or AJPS ports (or port range) on an OC4J instance. Similarly, dedicated ports (and again port ranges) can be configured on an OC4J instance for RMI and JMS traffic. These ports are typically found and configured within the opmn.xml configuration file within the OracleAS Oracle Home or, if running standalone OC4J, within the server.xml, rmi.xml, *-web-site.xml and jms.xml files.
The WebLogic Server request port and protocol management model is by default different from OC4J's in two important ways. First, a WebLogic Server instance is configured to have only two listen ports for incoming application requests: One port for accepting plain non-encrypted requests (which has to be set) and the other for accepting SSL encrypted requests (which is optional). Second, these ports are configured to accept requests for all supported request protocols as opposed to being dedicated to a specific protocol as is the case with OC4J instances.
Although in general this default model has proven to be sufficient for a majority of use cases, a WebLogic Server feature called *network channels* does provide the capability for a WebLogic Server instance to be configured with additional ports and for these ports to be dedicated to specific protocols. This feature can be used for the special use cases where such a configuration is required.

A WEBLOGIC SERVER INSTANCE IS ALWAYS CONFIGURED WITH AN HTTP LISTENER AND DOES NOT SUPPORT AJP

Oracle Application Server enables the OC4J instances within it to be fronted by Oracle HTTP Server (OHS) or to run, local to each OC4J instance, an in-process HTTP server. The most common configuration is for customers to use OHS versus the embedded OC4J HTTP Server. In the OHS configuration, the HTTP traffic coming into OHS is translated into an optimized protocol - AJP - between the OHS server and the OC4J server.
A WebLogic Server instance on the other hand must always accept HTTP requests and has no support for AJP. WebLogic Server domains can be (and frequently are) however fronted by a web tier for security and scalability purposes. A number of web servers, including OHS, are certified to act as a web tier to servers within a WebLogic Server domain. When WebLogic Server is fronted by such a web tier the protocol between the two tiers remains HTTP

Administrative Processes for WebLogic Server Based Environments

Although the general administrative capabilities of OC4J and WebLogic Server are similar, there are some important differences between the two administrative models which will require an adjustment to administrative processes of existing OC4J based environments when upgrading to WebLogic Server. The following sections outline the various aspects of these differences.

STARTING AND STOPPING SERVERS

Just as OC4J instances can be started and stopped using a variety of the OracleAS administration tools, so can the start/stop operations be performed on WebLogic server instances within a domain using the equivalent tooling as described by Table 2. Additionally, WebLogic server instances can also be started through the start-up scripts (startWebLogic.cmd|sh for admin server and startManagedWebLogic.cmd|sh for managed servers) which are available within the domain directory's /bin sub-directory. The possible approaches, and associated implications, of starting and stopping WebLogic Server instances are described within this WebLogic Server documentation page.

APPLICATION DIAGNOSTICS AND LOGGING

Diagnostics
The diagnostics capabilities of WebLogic Server are provided through the *WebLogic Diagnostics Framework* (WLDF). WLDF provides features that meet or exceed the capabilities of the *OracleAS Dynamic Monitoring Service*. These features - which include dynamic code instrumentation, image capture, watches, and notifications - result in extended application diagnostics capabilities which can greatly reduce the total cost of ownership of maintaining Java EE applications. Furthermore, WLDF capabilities can be exposed as custom dashboards within the WebLogic Server Administration console through the *WLDF Console Extension* features. Applications that currently use the OracleAS Dynamic Monitoring Service can continue to do so within Oracle Fusion Middleware 11g as described within the Upgrade of Custom Java EE Applications section of this paper.
It should also be noted that WebLogic Server has wide and deep support across 3rd party application management tools such as CA Wily Introscope and HP OpenView. Therefore, environments managed by such tools can typically be easily updated and continue operating as before against the upgraded applications running on WebLogic Server.

Logging
The *WebLogic Logging Services* provide a comprehensive set of logging features that match all OracleAS logging capabilities. The capabilities of the Oracle Diagnostics Logging (ODL) framework have also been integrated into WebLogic Server through the Oracle Java Required Files (JRF) template which is available in Oracle Fusion Middleware 11gR1 as described in the Upgrade of Custom Java EE Application section of this paper. Therefore, if an application is using the ODL framework directly for logging, it does not require to be modified for upgrade purposes. The JRF ODL integration into WebLogic Server is as follows:

- ODL log messages will be sent to a log file that is kept on the file system separate from the WebLogic server log files (with the <domain directory>/servers/<server name>/logs/<server name>-diagnostic.log file).
- Critical messages (errors) will be double-logged both in the ODL and WebLogic domain log file.
- ODL log query and configuration JMX MBeans are available in the domain's WebLogic admin server.

THREAD POOLS PERFORMANCE TUNING

OC4J Server instances have different thread pools for different purposes (system, http, and jca are the default startup thread pools). The parameters of each thread pool (such as max/min thread counts) can be individually tuned to achieve an optimal application request throughput for a particular environment.
A WebLogic Server instance has by default a single thread pool the thread count of which is automatically tuned to achieve maximum overall throughput. All requests are en-queued upon arrival in a common queue and prioritized according to administratively configured goals such as an application's desired response time or its fair-share usage of all available threads relative to other applications. This WebLogic Server feature - referred to as *WebLogic Work Managers* - effectively allows WebLogic Server instances to self-tune their thread counts for optimal request processing.

Upgrade of Custom Java EE Applications

To upgrade an OC4J based application so that it can be deployed to and run within a WebLogic Server domain, one must however ensure that the application's use of OracleAS extension APIs are properly

addressed and that the application's use of OC4J specific deployment descriptors are properly mapped to WebLogic Server specific deployment descriptors and/or features. The following sections provide more details on these upgrade tasks.

UPGRADE OF WEB SERVICES

In general the expectation of customers upgrading Web services from OC4J to WebLogic should be to use the matching Java Web Services API to that of OC4J - specifically JAX-RPC - on WebLogic Server. Generally this requires re-generating the Java artifacts on WebLogic Server to the identical underlying Java business logic using the WebLogic Server Web services tooling. For customers using older releases of OC4J where Web services standards did not exist within Java EE, it is recommended that they upgrade to the Java EE 5.0 standard JAX-WS on WebLogic Server. For absolute fidelity to a specific OC4J Web services public API – its WDSL - it is also possible to regenerate the Web services on WebLogic Server from the OC4J WSDL (top down development) and deploy the resulting Web service on WebLogic Server.

After producing the equivalent deployable Web service artifacts on WebLogic Server, the equivalent quality of service capabilities such as WS-Security and WS-ReliableMessaging should be applied as a secondary administrative operation.

WebLogic Server supports all of the web services standards and specifications, except for WS-Reliability, as those supported by OC4J.

WebLogic Server 11g also introduces support for database web Services which will allow OC4J based database web services to be seamlessly upgraded.

USAGE OF OC4J SPECIFIC DEPLOYMENT DESCRIPTORS

Security Deployment Descriptor Elements

Security configurations (e.g. authentication methods, security constraints, EJB method permission) contained in standard Java EE application deployment descriptors such as web.xml and ejb-jar.xml should remain untouched for upgrade and will continue to function – so long as the target WebLogic Server domain security has been configured as described in the Creation of WebLogic Server Domain for Upgraded Application section of this paper - when the application is deployed to WebLogic Server. For security configurations specified in OC4J specific descriptors (e.g. security role mappings), the WebLogic Server Security documentation should be consulted in order to map each configuration to an element within the equivalent WebLogic Server deployment descriptor.

Other Deployment Descriptor Elements

To prepare it for WebLogic Server deployment, OC4J specific deployment descriptors used by any application must be removed and replaced with their equivalent WebLogic Server specific settings. To do this, the elements of each of the deployment descriptor used should be examined and one of the following two actions should be taken as appropriate:

1. The feature provided by the OC4J deployment descriptor could have a direct mapping within the equivalent WebLogic Server specific deployment descriptor listed in Table 6. In such cases, the equivalent WebLogic Server descriptor with the appropriate elements and values should be created within the application prior to deployment.
2. The capability enabled by the OC4J deployment descriptor could be achieved through specific configuration of the WebLogic Server domain (at any level such as server instance, server resources, etc…). In such cases, the WebLogic Server documentation should be researched for the required domain configurations and the target domain should be appropriately modified.

Oracle WebLogic SmartUpgrade tool provides the capability for the mapping of specific OC4J deployment descriptor elements to their WebLogic Server equivalents. The Oracle Fusion Middleware 11gR1 Upgrade Guide for Java EE also provides more detailed information of how each category of OC4J specific deployment descriptor elements map to specific WebLogic Server settings.

CREATION OF WEBLOGIC SERVER DOMAIN FOR UPGRADED APPLICATION

A WebLogic Domain targeted for the deployment of an upgraded application should have a comparable topology and set of resources as the application's OC4J based environment (from here on referred to as the source OC4J environment). Given that WebLogic Server generally has a matching set of features as OC4J, the creation of a comparable topology involves the mapping of the OC4J features to the WebLogic server ones.
The sections below describe at a high level how each of the different elements of an OC4J based environment map to their equivalent concept in WebLogic Server and how a comparable target domain can be obtained. All the steps described in the sections below can be achieved through the WebLogic Server tooling described in the Administrative Processes for WebLogic Server Based Environments section of this paper.
It is important to note that although not described in the sections below, in most cases WebLogic Server offers features that go above and beyond those matching OC4J. The reader is therefore encouraged to explore these features through the referenced WebLogic Server documentations in order to assess their suitability for the enhancement of their application’s capabilities as part of the upgrade process.

OC4J INSTANCES

In general a single managed server is required, within the target WebLogic Server domain, for each OC4J instance to which the application is deployed within the source environment. In cases where the numproc setting of any such OC4J instances is higher than 1, then an additional managed server is needed on the same machine for each increment of the numproc setting. Each application deployed to an OC4J instance in the source environment should then be deployed and targeted to the associated managed server within the target WebLogic Server domain.

OC4J CLUSTERS

Just as is the case in an OC4J based environment, a *WebLogic Server cluster* of server instances is grouping of WebLogic Server instances that provides load distribution and fault tolerance for deployed applications.
To map a source OC4J environment’s cluster configuration to a target domain, for each cluster configured within the source OracleAS environment, a WebLogic Server cluster should be created within the target domain. Each application deployed on to a cluster in the OracleAS environment should then be deployed and targeted to the associated cluster within the WebLogic Server domain. WebLogic Server clustering features are a superset of the OC4J clustering features and in general all application benefits obtained in an OC4J cluster should be obtainable through enablement of appropriate features within the associated WebLogic Server cluster.

OC4J WEB SITES

As described in the General WebLogic Server Concepts for OC4J Users section of this paper, a WebLogic Server instance does not require explicit web site configurations. As such, there is no need

to map website configurations from the source OC4J environment to the target domain. If explicit port and protocol configuration assignments are necessary for multiple applications deployed to the same server, the WebLogic Server *network channels* feature can be used.

JDBC DATA SOURCES AND CONNECTION POOLS

In general, equivalent data source configuration can be easily created on WebLogic Server as of that contained within OC4J based on the database connection information, pooling requirements and JDBC driver. Data sources for OC4J can be defined at the OC4J instance level or included in an Enterprise Application in a file named datasources.xml. WebLogic Server data sources are typically defined at a domain level and applied across the cluster or to specific managed servers within a domain. WebLogic Server data sources can also be packaged as a JDBC module within Enterprise Applications providing the equivalent capability of application level data sources within OC4J.

Just as is the case in an OC4J based environment, a *WebLogic Server JDBC data source* is an object bound to the environment's JNDI context which provides database connectivity through a pool of JDBC connections. Applications look up a data source in the JNDI context in order to use a database connection from this pool.

To map a source OC4J environment's JDBC data source configuration to a target domain, for each JDBC data source configured within the source OracleAS environment, a WebLogic Server JDBC data source with the same JNDI name should be created within the target domain. Also note that WebLogic Server JDBC data sources can be configured to take advantage of Oracle Real Application Clusters (RAC) as described by the following Oracle white paper.

There are two important differences between OC4J and WebLogic Server JDBC data sources which should be highlighted: First WebLogic Server JDBC data sources have an implicit connection pool associated with them and therefore no explicit connection pool needs to be created within the domain to match the connection pools within the OC4J environment. Second, WebLogic Server JDBC data sources always behave like managed Oracle data sources – there is no natural equivalent to OC4J native data sources.

JMS RESOURCES

Within an OC4J based environment, there are three JMS providers: in-memory, file based and Oracle DB Advanced Queuing (AQ). There is direct equivalence within WebLogic Server for in-memory and file based JMS providers. WebLogic Server 11g introduces support for a JMS AQ provider which, as described in the Upgrade of Client Applications section of this paper, should allow for a seamless upgrade of OC4J applications making the use of this functionality..

In OC4J, JMS connection factories and destinations are first configured at an individual OC4J instance's JMS server, and then mapped through the use of resource providers and/or JMS connectors. In WebLogic Server, these primary JMS resources are created within a *WebLogic JMS module*. JMS modules are targeted to a *WebLogic JMS Server* within a domain. WebLogic JMS servers provide a central point which allows for the configuration of message persistence, durable subscribers, message paging, and quotas for their targeted JMS destinations. This differs from OC4J where this configuration is done per destination for message persistence, through the use of OC4J specific JVM properties for message paging, through the use of JMX MBeans for durable subscriptions, and where message quotas is not supported.

To map a source OC4J environment's JMS configuration to a target domain, one must first create a set of WebLogic JMS servers with configurations that reflect the OC4J environment's JMS resource providers, connectors, connection factories and destination configurations. After this step, for each set of JMS connection factories and destinations with common configuration, a WebLogic Server JMS module should be created. The module should then be populated with JMS connection factories and

destinations which have the same JNDI name as their equivalent version in the OracleAS environment. Finally, the JMS modules should be targeted towards the appropriate WebLogic JMS server within the domain.

REMOTE JMS PROVIDERS

In an OC4J based environment, remote destinations and connection factories for 3rd party JMS providers such as WebSphereMQ, Tibco, and SonicMQ are configured as part of a JMS connector configuration. WebLogic Server remote destinations are instead accessed through the *WebLogic Server Foreign Server resources* which enable users to integrate external JMS providers with WebLogic Server by providing a mapping between a domain's JNDI tree and external remote JNDI names of JMS destinations and connection factories.
To map a source OC4J environment's external JMS provider configurations to a target domain, one must create a JMS module which contains a foreign server, as well as a set of foreign connection factories and foreign destinations that serve as a proxy to the remote destinations which need to be accessed from the domain.

STARTUP AND SHUTDOWN CLASSES

WebLogic Server provides startup/shutdown (SU/SD) class functionality as part of its *Application Life Cycle Events* feature. Any SU/SD class configured within the source OC4J environment should first be converted to a set of WebLogic Server SU/SD classes. Each class must then be configured within the target WebLogic Server domain and targeted to the WebLogic Server instances corresponding to the associated OC4J instances in the source environment.
Unlike OC4J SU/SD classes, a WebLogic Server SU/SD class does not need to implement any specific interface or provide pre/post deployment methods. The custom logic must instead be implemented within the standard main() method of the class. For startup classes, WebLogic Server allows for pre/post deployment execution of this logic by providing *configuration parameters* which must be set accordingly when configuring a domain with the startup class. To convert an OC4J startup class, it might therefore be necessary to create two WebLogic Server startup classes: One that contains the code from the original class's preDeploy method and another which contains the code from the postDeploy method.
Although pre-configured parameters can be passed to the main() method of a WebLogic Server SU/SD class, WebLogic Server startup classes have no access to similar arguments as the JNDI context and configuration hash table parameters passed to an OC4J startup class. If the custom logic within the startup class to be converted makes use of these parameters, then this logic should be modified to obtain the JNDI context from scratch and access the server configuration through the WebLogic Server JMX interfaces.

SECURITY CONFIGURATIONS

The target WebLogic Server's environment for an upgraded application should provide a similar security configuration as the application's source OC4J environment. Table 7 describes how each OC4J environment security configurations should be mapped to a WebLogic Server environment in order to achieve a comparable outcome.

CLASS LOADING CONFIGURATIONS

It is important to properly construct the target WebLogic Server's environment so that it's class loading configuration correctly mimics the one found within the application's source OC4J environment.

Due to the intricacies of the differences between the OC4J and WebLogic Server class loading models, it is important to develop a good understanding of *WebLogic Server Application Class Loading* prior to setting up the target WebLogic Server environment's class loading configuration. To facilitate this understanding for users familiar with options available for application class loading configuration in an OC4J based environment, Table 8 provides a high level mapping of the main OC4J approaches for making a class available to an application to the most comparable way of achieving the same outcome within a WebLogic Server environment.

WEB TIER

As is the case with OC4J server instances, WebLogic Server instances can be (and frequently are) fronted by a web tier for security and scalability purposes. This web tier effectively acts as a client of the web applications deployed within the application server tier.

OC4J web tiers are typically composed of Oracle HTTP Server instances configured with the mod_oc4j module. A web tier fronting WebLogic Server instances must be configured with a *WebLogic Server web server plug-in* which is available for Oracle HTTP, Apache HTTP, as well as Microsoft's Internet Information Servers. The areas of difference between an Oracle HTTP server configured with mod_oc4j and web servers configured with the WebLogic Server web server plug-ins are as follows:

- Load Balancing: WebLogic Server web server plug-ins support a simple round robin load balancing where as an Oracle HTTP server configured with mod_oc4j supports a number of other load balancing methods including metric based, weighted round robin, random, as well as the ability to combine these algorithms with local affinity where processes on the local machine would be preferred over remote processes. If the application being upgraded has a dependency on these mod_oc4j load balancing capabilities, then the use of a load-balancer technology fronting the WebLogic Server domain web tier which provides similar capabilities might be required.
- Configuration: An Oracle HTTP server configured with mod_oc4j can dynamically discover OC4J instances and its deployed web applications. WebLogic Server web server plug-ins on the other hand have a more static configuration model where all web application URLs are configured statically. Although WebLogic Server web server plug-ins have the ability to discover container instances within a WebLogic Server cluster, the configuration must first be bootstraped with the host and port connection information for one or more members of the cluster. The benefit of this static configuration model is that it provides loose coupling between the web tier and the web applications due to the lack of dependency on WebLogic Server domain components.
- Transport Protocol: WebLogic Server web server plug-ins use HTTP as the transport protocol where as an Oracle HTTP server configured with mod_oc4j uses the Apache JServ Protocol (AJP). Therefore, environments in which a firewall is configured between the web tier and the web applications need to be modified to allow for HTTP traffic between these tiers.

Upgrade of Client Applications

The nature of the external interfaces exposed by a Java EE application could be impacted by the upgrade from OC4J to WebLogic Server. This impact can in some cases lead to differences in behavior and require modifications to the client applications. The sections below describe these possible impacts and related effects on application clients.

JAVA SERVER PAGES AND SERVLET CLIENTS

The main impact to JSP and Servlet clients when an application is upgraded to WebLogic Server could be caused by differences in HTTP session state replication model between WebLogic Server and OC4J. Where as it is possible to configure an OC4J cluster to contain any number of in-memory replicated copies of the HTTP session state, The *WebLogic Server in-memory HTTP session state replication* only supports a primary-secondary, two-copy model. In most cases, this difference should have no impact on JSP and Servlet clients, however for rare cases where an application might explicitly rely on more than two copies of the HTTP session state to be available for its clients, an *Oracle Coherence* based solution should be considered.

JAVA NAMING AND DIRECTORY INTERFACE CLIENTS

Clients of an upgraded application which use the OC4J Java Naming and Directory Interface (JNDI) provider to lookup application interfaces and/or resources will have to be modified to use the *WebLogic Server JNDI* provider instead. This modification must be done to the application's JNDI initial context creation code as follows: All instances of the OC4J JNDI URLs should be identified. Typically the OC4J URL is structured in the following format: <prefix>://<host>:<RMI or OPMN request port>:<oc4j_instance>/<application-name> An example URL for a full OracleAS installation with an OC4J instance named oc4j1 and an application deployed called myapplication would be opmn:ormi://127.0.0.1:6003:oc4j1/myapplication. Note that the prefix can be opmn:ormi within full Oracle Application Server installation using the Oracle Process Management and Notification infrastructure or just ormi: when using a standalone OC4J installation.

- The URL of the provider should be modified to point to the target WebLogic Server domain's administration server using the t3 protocol (e.g. t3://127.0.0.1:7001).
- The security credentials used must be valid within the target WebLogic Server domain.
- The initial context factory used must be changed to the WebLogic Server WLInitialContextFactory class. This class should also be made available to the client application's class loader through a *WebLogic Server client jar file*.

Additionally, if the client is running within an OC4J server instance, the server's server.xml environment-naming-url-factory-enabled attribute must be set to true in order to allow the use of multiple JNDI providers within the same OC4J instance.
Another important difference between the OC4J and WebLogic Server JNDI providers which might impact client applications is the scoping of JNDI namespaces. OC4J JNDI objects can have an explicit application scope. Therefore, when performing a lookup, OC4J JNDI clients can use a URL which identifies a specific OC4J server instance and includes the name of the target application. The WebLogic Server JNDI objects on the other hand always have a global namespace. Therefore, a WebLogic Server JNDI client performing a lookup cannot specify a URL identifying a target application explicitly. This implies that as part of the upgrade to WebLogic Server, one must ensure that the JNDI name of all JNDI resources deployed to the same WebLogic Server domain are unique regardless of the application they belong to and that if necessary, JNDI clients are modified to use their target object’s unique JNDI name.

ENTERPRISE JAVA BEAN CLIENTS

There are two cases where the upgrade of an application to WebLogic Server might have Enterprise Java Bean (EJB) client related impacts. In the first case, the EJB being upgraded has remote clients which are either stand-alone or are deployed to an OC4J server. In the second case, the application being upgraded remotely uses OC4J based EJB interfaces. The following two sections address these cases.

Clients of Upgraded Application

Remote clients of an upgraded application's EJB interfaces will need to be modified to use the WebLogic Server JNDI provider as described in the Java Naming and Directory Services Clients sub-section. The use of the WebLogic Server JNDI provider will lead to the client application obtaining WebLogic Server EJB client stubs which can potentially impact the client application as follows:

- RMI Protocol: Client applications will end-up using one of the *WebLogic Server RMI* transport protocols as opposed to OC4J's RMI transport protocol, ORMI. The default WebLogic RMI transport protocol used is the *WebLogic Server T3* protocol, but IIOP can also be used.
- Load Balancing: In OC4J client EJB requests can be configured to be load-balanced through the InitialContext JNDI object (random or sticky) across the OC4J cluster for each invocation of Context.lookup(). In WebLogic Server EJB client request load-balancing is handled automatically by remote EJB client stubs. The load-balancing behavior of these stubs is configured through WebLogic Server weblogic-ejb-jar.xml deployment descriptor level configurations and can be set to occur at InitialContext creation or EJB method invocation time.

It should also be noted that the considerations mentioned above are with regards to either stand-alone clients or clients currently running within an OC4J server instance but also being upgraded to run within a WebLogic Server instance.
For EJB clients that are not a stand-alone application and that will continue running within an OC4J server instance making remote invocations to an upgraded WebLogic Server EJB application the following two additional implications of an upgrade should also be considered: First, the application's security context will not be automatically propagated. If this security propagation is necessary, the client will require modification in order to explicitly use the existing security context's credentials at the creation of the WebLogic Server JNDI initial context. Second, JTA transaction propagation and XA recovery within the context of the remote EJB invocations will not be possible and if needed the client application itself will require upgrade.

Clients of OC4J Enterprise Java Beans within Upgraded Application

Applications being upgraded might need to act as a remote client to EJB components that will continue running within an OC4J server. If not already the case, such clients will require modification in order to use the OC4J RMIInitialContextFactory JNDI initial context factory located in the oc4jclient.jar file. This jar file will also need to be made available to the WebLogic Server class loaders for the application through one of the mechanisms highlighted in the Creation of WebLogic Server Domain for Upgraded Application section of this paper. Also note that the propagation of security context will require the configuration of the target OC4J server instance as a *WebLogic Server SSL* client. Furthermore, JTA transaction propagation and XA recovery within the context of the

remote EJB invocations will not be possible and if needed the target EJB application itself will require upgrade.

JMS CLIENTS

It should first be noted that the information contained in this section is with regards to JMS clients that are not Message Driven Beans (MDB). MDBs are usually tightly coupled to the JMS provider's resources and as such should always be upgraded together with these resources. Specific *WebLogic Server MDB* capabilities and behavior should be considered during the upgrade of MDB applications. There are two cases where the upgrade of an application to WebLogic Server might have JMS client related impacts. In the first case, the JMS provider - and its related resources such as destinations and connection factories - is upgraded to WebLogic Server and both upgraded client applications and non-upgraded OC4J client applications still running within an OC4J server will require adjustments. In the second case, the JMS provider remains within an OC4J infrastructure and client applications being upgraded to WebLogic Server will require adjustments. The following two sections address these cases.

JMS Provider is Upgraded to WebLogic Server

Stand-alone JMS clients or JMS clients currently running within an OC4J server instance but also being upgraded to run within a WebLogic Server instance will need to be modified to use the WebLogic Server JNDI provider as described in the Java Naming and Directory Services Clients sub-section. The use of the WebLogic Server JNDI provider will lead to the client applications obtaining JMS resources from the WebLogic Server JMS provider. As such, the following important differences between the OC4J and WebLogic JMS provider behavior and capabilities should be considered as they could potentially impact client applications: Message Ordering: Just as is the case with the OC4J JMS provider, WebLogic Server provides the capability to guarantee strictly ordered message processing. Additionally, the *WebLogic Server JMS Unit of Order* feature which allows for additional message order processing capabilities.

- Connection Pooling: Unlike the OC4J JMS provider, the WebLogic Server JMS provider provides no pooling capability for stand-alone clients. If this feature is required, stand-alone clients should be modified to implement this capability through explicit re-use of JMS resources.
- Network Connections: Clients using the WebLogic Server JMS provider use a single network connection per client virtual machine regardless of the number of JMS connection objects used. This behavior is slightly different from that of the OC4J JMS provider which associates each JMS connection object with a separate network connection.

JMS clients that are not stand-alone applications and that continue running within an OC4J server instance while using JMS resources within a WebLogic Server environment should use the *OC4J Oracle Enterprise Messaging Server JMS Connector feature*.

JMS Provider Remains within OC4J

Applications being upgraded might need to continue using JMS resources within an OC4J JMS provider. Such applications should treat the OC4J JMS resources as remote JMS providers and use the *WebLogic Server Foreign Server* feature in order to provide access to the OC4J JMS resources to the WebLogic Server deployed JMS clients.

AQ JMS CLIENT

Oracle has introduced support for AQ as part of WebLogic Server 11g. This functionally allows for the configuration of a *WebLogic Server Foreign Server* that can reference an AQ JMS provider. This mechanism enables AQ JMS client applications to directly access Oracle Streams Advanced Queuing (AQ) system from WebLogic Server without any required modifications.

Summary

Oracle Fusion Middleware moves to its next generation with the stable, proven WebLogic Server as the strategic application server runtime. Oracle Application Server continues forward with new patchsets and capabilities based on its lifetime support policy preserving the investment Oracle customers have made with Oracle Fusion Middleware. This paper has outlined the process and work involved for the upgrade of custom Java EE applications from Oracle Application Server's OC4J to WebLogic Server.

Contact address:

Frances Zhao
Oracle Corporation
1211 SW 5TH AVE, SUITE 800
PORTLAND, OR 97229
U.S.A

Phone: +1503-276-2514
Email Frances.Zhao@oracle.com
Internet: www.oracle.com