Torry Harris
Distributed e-enterprise Computing

Cold Fusion to WebSphere Migration

Authors: Sunil Prasad

Intended Audience: This paper is primarily intended for managers and architects considering options for migrating applications on ColdFusion to WebSphere.

Further Inquiries and Feedback: torry_harris@thbs.com

Notices:

1 Introduction

In this document, we discuss a general strategy for migrating applications off the tag-based ColdFusion application server from Allaire to WebSphere application server from IBM. The ColdFusion application server was one of the first application servers to hit the market and that was very popular and rapidly became the platform of choice for web applications, the reason being it was very simple to use. But, to develop large-scale, business applications which are highly scalable, it may not be suitable since it is more of an extended web-server than an application server. The WebSphere application server on the other hand, is a highly robust and scalable offering from IBM that is based on the J2EE standards.

2 ColdFusion Applications

ColdFusion is a first generation application server built on a tag-based programming model. ColdFusion was primarily designed for quick generation of HTML interfaces to Database tables. The focus of the product is on using the tag extensions and build HTML pages. Since the product supports a rich set of tag extensions and supporting function libraries along with a proprietary scripting language CFScript (which is similar to JavaScript), it is very easy to develop simple applications in a very short timeframe. Design and development of ColdFusion applications is generally (though not necessary) done through the ColdFusion Studio.

ColdFusion has three important pieces of functionality – HTML tag extensions, function library, a scripting language. Though it does not have a concept of a servlet engine or a EJB container, you can execute servlet/EJBs by using specific tags.

In a typical ColdFusion project, we will have a set of CFML pages which will have business-logic embedded in them along with the presentation logic as well as logic to interact with database to pull/put data. It is this mixing up of business-logic with database/presentation logic which reduces the scalability factor of ColdFusion application.

3 WebSphere Applications

WebSphere is a standards based application server. It implements the highly popular Java 2 Platform Enterprise Edition (J2EE) programming model. This model is based on the Model-View-Controller (MVC) design pattern for user driven applications. In the MVC pattern, a user driven application is considered to have a set of Views, which allow the user to interact with the application. A Controller handles user inputs collected from a View and passes them to the appropriate Model. Once the Model has processed these inputs and produced outputs, the Controller uses them to populate the next View for the user. In the J2EE programming model, Java Server Pages (JSPs) or HTML pages serve as Views, Java Servlets or logic only JSPs serve as Controllers and Enterprise JavaBeans (EJBs) or Java classes serve as Models. WebSphere applications can be developed using the Visual Age for Java IDE, the WebSphere Studio IDE or any other Java development environment including a simple text editor.

WebSphere applications consist of a set of Java classes conforming to the J2EE standards and using the allowed set of J2EE APIs. Applications have to be deployed into the WebSphere container using a set of tools provided with the product.

In a typical project, WebSphere developers create a set of Entity EJBs or Java classes to encapsulate their SQL statements and other data access logic, a set of Session EJBs or Java classes to encapsulate their business logic, a set of Servlets that contain data validation logic, and, a set of JSPs that contain presentation logic.

WebSphere developers typically use popular J2EE APIs like Java Database Connectivity (JDBC), Java Messaging Service (JMS), and the Java Transaction API (JTA).

4 Migration Methodology

Since a typical ColdFusion application is an integrated one with business/presentation/database logic mixed up, we can take two approaches in migrating such an application,

- Re-architect the application to fit into the Model-View-Control (MVC) architecture. This will involve a lot of rework since it is almost as good as re-development and hence can be very expensive w.r.t. both time and money.
- Write JSP tag implementations for each of the ColdFusion tags and convert the CFML pages into equivalent JSP pages. The ColdFusion functions will also get converted into the Java equivalents. We will also have to map the CFScript to Java/JavaScript.

During our discussion further, we will assume that we will be going with the second option. This approach though faster, has a lot of issues associated with it like,

1. **Implementing CFML tags** can get very complicated. Good example for this could be the CFOBJECT tag which handles COM/CORBA/EJB objects.

   CFOBJECT allows you to create and use COM (Component Object Model) objects. Any automation server object type that is currently registered on a machine can be invoked. You can use a utility like Microsoft's OLEView to browse COM objects.

   To use CFOBJECT with COM components, you need to know the program ID or filename of the object, the methods and properties available through the IDispatch interface, and the arguments and return types of the object's methods. The OLEView utility can give you this information for most COM objects. On UNIX, COM objects are not currently supported by CFOBJECT.

   ColdFusion Enterprise version 4.0 and above supports CORBA through the Dynamic Invocation Interface (DII). To use CFOBJECT with CORBA objects, you need to know either the name of the file containing a stringified version of the IOR or the object's naming context in the naming service. You also need to know the object's attributes, method names and method signatures. User-defined types (for example, structures) are not supported. These CORBA objects must already have been defined and registered for use.

   CFOBJECT allows you to create and use JAVA objects including EJB objects. This support is currently only for NT. To be able to call Java objects, ColdFusion uses a JVM embedded in the process. The loading, location and the settings for the JVM are configurable using the ColdFusion Administrator pages. Any Java class available in the class path specified in the ColdFusion Administrator can be loaded and used from ColdFusion using the CFOBJECT tag.

   We need to use the following steps to access Java methods and fields:
   - Call CFOBJECT to load the class.
   - Use the init method with appropriate arguments to call a constructor explicitly.
     For example, <CFSET ret = myObj.init(arg1, arg2)>

     Calling a public method on the object without first calling the "init" method results in an implicit call to the default constructor. Arguments and return values can be any valid Java type (simple, arrays, objects). ColdFusion does the appropriate conversions when strings are passed as arguments, but

not when they are received as return values. Overloaded methods are supported as long as the number of arguments are different.

2. Security framework

ColdFusion supports an extensive security framework with some features which do not have equivalents in WAS.

- When administering a server used by multiple customers (e.g. ISP servers) a security hazard may arise from the use of several ColdFusion tags. CFFILE directly allows users to upload and save files on your server. CFFTP allows programmers and users to send and receive files via the FTP protocol to and from your ColdFusion Server. CFPOP allows users to receive files attached to mail messages and save them on your server.

  Each of these tags allows users to upload (send) their files to your web application hosting server. This kind of operation must be carefully considered and managed. Allowing unknown users to upload *any* file can allow an unknown attacker to carry out many unwanted changes to your hosting server.

  The following are examples of harmful actions allowed by unmanaged and untrusted uploading:
  - Attacker could overwrite your server's configuration files.
  - Attacker could put a Trojan horse (a program that compromises security settings) on your server.
  - Attacker could overwrite or place new CFML pages on your server. This could put your database(s) at risk as well.
  - Attacker could overwrite your server's registry or security (password and policy) files.

  With this kind of support, important consideration is how you go about protecting the use of CFML templates that incorporate the CFFILE, CFFTP, and CFPOP tags. If you do not carefully manage the files that your users work with when using your application, Allaire strongly recommends that you protect such templates with application-based security so that you can control who is permitted to execute these operations. It's important to ask yourself whether you trust your users not only to understand but also respect your security policies. Not only can a malicious user abuse your trust, but so can one who is uninformed.

  ColdFusion basic security lets you disable execution of seven CFML tags that could present security hazards. You can, however, specify a special directory, called the Unsecured Tags Directory; this is the only directory from which ColdFusion will execute tags you disable with Basic security. By default, this is the directory in which the ColdFusion Administrator is installed. ColdFusion displays an error message when it encounters a restricted tag in an application. Tags you disable with basic security remain disabled if you switch to ColdFusion advanced security. This feature is not directly supported in WAS and hence need to be handled programmatically.

  Using ColdFusion Server 4.5.1 Enterprise, you can configure Advanced Security and Security Sandboxing to make sure that each programmer cannot abuse the right to use a set of tags that you specifically allow the programmer to use. Sandboxing creates a limited space and context for each programmer to use, and enforces functional security limits with programmer permissions and limited file-system paths for each programmer. These kinds of limitations can eliminate many of the previously mentioned risks.

In WAS, we security sandboxing is supported by means of having separate class-loaders for each web-application and everything else must be handled programmatically.

- Using ColdFusion Administrator, you can configure the types of SQL commands that can be performed against a data source. This is not supported in WAS, but we should be able to handle it at the database level by GRANTing specific permissions to different users.

- Delete, optimize, purge, search, and update access to search verity collections. Verity collections are not supported by WAS and hence such functionality will require a lot of customization.

  Cold Fusion provides a mechanism (through the Cold Fusion Administrator) by which you can create Verity collections. A verity collection is actually a logical database made up of a number of physical files stored on your web server's hard drive(s). When you create a collection with the Cold Fusion Administrator this actually creates a certain file structure on disk which are the physical files that make up the Verity collection. A registry entry is also created in HKEY_LOCAL_MACHINE/SOFTWARE/Allaire/ColdFusion/CurrentVersion/Collections (NT). This entry associates the collection logical name with its physical residence on disk. Additionally, the name that you use is a logical name that you use when referring to these collections via the CFINDEX and CFSEARCH tags. This logical name is used in the COLLECTION attribute of both of these tags.

- ColdFusion supports usage of ODBC datasources to authenticate Cold Fusion developers. WAS does not support storing security profile information in a ODBC data source, but we should be able to simulate this by writing application extensions.

3. Custom error handling

In ColdFusion, using the CFERROR tag, we can display customized HTML pages when errors occur. This allows us to maintain a consistent look and feel within our application even when errors occur. It also allows us to optionally suppress the display of error information.

In WAS, this can be simulated using customized JSP error pages.

4. Debugging options

ColdFusion allows you to see important debug information by placing an url parm mode=debug in any ColdFusion url. You can turn on the veiwing of degug information by using the debug settings page in to the administrator. You can also restrict the debug information to certain IP addresses. Another option available is to use an url parameter mode=debug. The url would look like this www.allaire.com?mode=debug . This url will display debug information whether you have debug information turned on or off in the administrator.

The above settings are all valuable tools while developing and maintining a ColdFusion site. One of the areas you will want to be aware of is that anyone can enter mode=debug onto an url and see the debug information. The information is not a direct security risk but could help someone with bad intentions. In order to eliminate the ability for anyone to type mode=debug you should do the following:

- Go into Debugging in the CF Administrator
- Go to the green box that states: Restrict debug output to selected IP addresses
- Enter only one IP Address - 127.0.0.1

- Add
- Apply
- Restart ColdFusion

This will restrict the use of mode=debug. This should be done on all production machines. There is no need to select any of the checkboxes for returning debug information to enforce the IP restriction rule.

In WAS, we may not need to do anything to restrict the debug output to only the server since debugging messages will get written only to the server log by default. But, if we want only a few users to receive the debug output, this needs to be handled programmatically.

To help diagnose potential problems or bottlenecks in your site, you can have ColdFusion log the names of any pages that take over a specified length of time to return. When enabled, any output will be written to the "server.log".

WAS has an extensive logging mechanism wherein you can control the level of detail as well. It supports Object level trace feature as well, but this needs a Distributed debugger client to analyze.

5. Application framework

- Request processing in ColdFusion

  It is useful to understand how ColdFusion processes .cfm requests. In ColdFusion 4.0 requests for .cfm pages are processed in this way:
  - A .cfm page is requested by a web browser.
  - Due to an installed advanced Web Server mapping, setting the .cfm extension to one of the supported web server API "stubs", the .cfm request is sent to the appropriate stub (ISCF.dll, NS35CF.dll etc…).
  - The stub then sends the request to the ColdFusion server for processing
  - A listener thread receives the request and sends it to one of the waiting active simultaneous thread that will process the request. If all of the simultaneous threads are busy processing then the listener thread places the request on a waiting list. When one of the simultaneous request slots becomes available the listener thread sends the waiting request to the active slot.
  - The ColdFusion server processes all of the CFML tags and converts the results to HTML and sends them back to the web server to be sent to the browser.
  - The thread is not destroyed, but put back into the active pool to be used again.

- Code to be executed before and after a page's execution

  ColdFusion allows the user to specify an Application.cfm file, which is executed before each application page it governs, as well as a file named OnRequestEnd.cfm, which is executed after each application page in the same application. The OnRequestEnd.cfm file will not be executed if there is an error or an exception in the called page, or if the called page executes the CFABORT or CFEXIT tag.

  You can specify a single Application.cfm file for your application, or use different Application.cfm files that govern individual sections of the application.

  In WAS, we can have one include file for functionality similar to the one provided by having Application.cfm file and another include file equivalent to OnRequestEnd.cfm functionality (for each

application). In the first, we would also want to set variables with application-level scope with all the variables of the form <application name>.<variable name>, whether it is setting the value or retrieving the value of the variable.

- Client/Application/Session/Server level variables

  To start with, it is important to understand the differences between Application / Session variables and Client variables and how they are handled by Cold Fusion. The main differences between session/application and client variables are that Session and Application variables are stored in memory (the ColdFusion server's memory) rather than in the registry so they're faster to access and don't fill up the registry. Session and Application variables also have a much finer granularity of time-out associated with them. With client variables, you can specify in the Cold Fusion Administrator that you would like them to be deleted from the registry after a certain period of inactivity. That's basically the only way they could "time out." Session and application variables can be set to time out after a very precise period of inactivity either on a per page basis via the CFAPPLICATION tag or by setting defaults in the administrator.

  The whole point of this time out capability is to create the illusion of a **"session"**. In other words, the ColdFusion engine doesn't really know what a session is or when it might end. It doesn't know if the user has actually finished using a particular web application (i.e., accessing a certain set of web pages) or if he/she is just a slow reader and is moving slowing between pages. A ColdFusion programmer could use the timeout feature of session variables to define what a "session" is. For example, he/she could decide that a session ends when a new page has not been accessed for 5 minutes. After 5 minutes of inactivity, any session variables he has set would be discarded.

  Like client variables, however, session variables depend on the same client management mechanism to identify the user. So session variables still require a CFTOKEN and CFID to be returned with each page request so the engine knows whose session variables to access. Also, it is important to note that Session and application variables must be always be accessed via their complete names as there's no automatic name resolution associated with them.

  Cold Fusion provides Client Management option wherein values specific to each client session is stored. It provides the option for specifying the location for storing client variables. They are Registry (NT only), data-source or cookie. Each of them have their own limitations. In WAS, we can map this to Session-level variables, but will not be able to provide the Registry option. Data-source option is used in case the quantity of the client data is huge (say > 50K per client). The client variables should be accessed as though you are accessing Session variables. So, the accesses to client variables which are of the format Client.<variable name> should translated to the access the same from the Session. Even the Session variables in Cold Fusion referenced in the form Session.<variable name> must also be treated in the same way as the Client variables in J2EE.

  ColdFusion allows specifying application-level settings using CFAPPLICATION tag. In WAS, these map to PageContext settings in JSP using APPLICATION and SESSION scope for Application-level variables and Session/Client level variables. The default time-out for session variables is set to 20 minutes in Cold Fusion. This is 30 minutes in WAS. In WAS, session timeout cannot be set at the application (web-application) level, but only at the session manager level. But it is possible in ColdFusion using the SESSIONTIMEOUT attribute of the CFAPPLICATION tag. In WAS, there is no equivalent to APPLICATIONTIMEOUT attribute of CFAPPLICATION tag. This can be handled by means of a daemon thread which checks for application-level variables and resets them after the APPLICATIONTIMEOUT period has expired. In Cold Fusion, there are specific functions to get

a list of say, all application-level variables or delete an application-level variable. Such things have no direct equivalents in WAS. Hence they need to be handled programmatically.

WAS does not support server-level attributes at all, but Cold Fusion does. This can be simulated by using variables with Application scope with the variable name as say Server.<variable name>.

- Locking sections of code

  In ColdFusion, CFLOCK tag provides a means of implementing exclusive locking in ColdFusion applications. The reasons for using CFLOCK may be for,
  - Protecting sections of code that manipulate shared data, such as session, application, and server variables.
  - Ensuring that file updates do not fail because files are open for writing by other applications or ColdFusion tags.

  There is no direct equivalent for CFLOCK tag in the WAS world. But, We can achieve the same (though to a limited extent) by using the Synchronized blocks in Java. The concept of read-only locks supported by Cold Fusion may not be possible in WAS at all.

- Custom-tag extensions

  ColdFusion allows you to build extensions or custom tags in two ways:
  - Using C++ to code DLLs (Windows) or shared objects (Solaris) that provide a custom tag you can use in your application pages.
  - Using CFML to create custom tags you invoke in your application pages.

  CFX tags are custom tags written against the ColdFusion Application Programming Interface. Generally, you create a CFX if you want to do something that's not possible in CFML, or if you want to improve performance of a task in CFML that's repetitive. Unlike CFML custom tags, CFXs are implemented as DLL files and can:
  - Handle any number of custom attributes.
  - Use and manipulate ColdFusion queries for custom formatting.
  - Generate ColdFusion queries for interfacing with non-ODBC based information sources.
  - Dynamically generate HTML to be returned to the client.
  - Set variables within the ColdFusion application page from which they are called.
  - Throw exceptions that result in standard ColdFusion error messages.

  You can build CFXs using C++ or Java. Then, to be able to use the CFX, you have to register it in the ColdFusion Administrator.

  Most of the functionality implemented using CFX tags can be implemented using JSP tag libraries in WAS as is done with most of the CFML tags themselves. But, while writing custom tags using CFX in Cold Fusion, we extend CustomTag which is a class supplied by Cold Fusion. So, while migrating Cold Fusion custom tags, implemented using CFX API, we need to concentrate on the functionality provided by the tag and make sure we don't use any of the Cold Fusion classes. Migrating CFX tags implemented using the CFX Java API must be quite straight forward, but not those implemented using the CFX C++ API since it involves language-translation or using JNI.

- Transactions

In ColdFusion, transaction support is by means of **CFTRANSACTION** tag. The transaction begins at the first CFML tag after the CFTRANSACTION tag that requires a data source. All subsequent CFML tags contained in the CFTRANSACTION block must use the same data source. An error is thrown if a different data source is specified. When the </CFTRANSACTION> is executed, the database transaction is committed and the transaction mode is ended. If a database action returns an error or the end of page processing finds an open database transaction, then the transaction is rolled back and the transaction mode is ended.

In WAS, this feature should be handled programmatically since there is no built-in tag for handling transactions. But WAS supports JTA APIs which can be used while writing an equivalent tag / function in WAS.

6. CFScript

ColdFusion offers a server-side scripting language, CFScript, that provides ColdFusion functionality in script syntax. This JavaScript-like language gives developers the same control flow, but without tags.

CFScript is based on JavaScript, there are some key differences:

- You can read and write ColdFusion variables inside CFScript
- CFScript uses ColdFusion expressions, which are neither a subset nor a superset of JavaScript expressions. For example, there is no < operator in CFScript.
- No user-defined functions or variable declarations are available.
- CFScript is case-insensitive.
- All statements end in a semi-colon, and line breaks in your code are insignificant.
- In CFScript, assignments are statements, not expressions.
- Some implicit objects are not available, such as Window and Document.

CFScript is not directly exportable to JavaScript. Only a limited subset of JavaScript can run inside CFScript. JavaScript supports server-side scripting as well. So, in WAS, we should be able to migrate CFScript to either JavaScript or Java in cases where JavaScript equivalents are not available (like ColdFusion expressions).

7. Windows NT Registry access

ColdFusion includes the CFREGISTRY tag, which allows you to get, set, and delete registry values. The equivalent functionality can be implemented in J2EE, but it may involve accessing the native system which may make the application platform-dependent since system-registry is bound to an individual system and hence will limit the scalability of the migrated application.

8. Performance monitoring

The ColdFusion Server statistics (a set of counters), available on Windows through the performance monitor, and on UNIX through the STAT command, provide one mechanism to monitor the server status while performing testing and tuning scenarios. On NT, this enhancement is installed automatically by the ColdFusion setup.

In case of WAS, we have the resource analyzer (which serves the same purpose, but has more monitors) which runs as a separate process (It is a technology preview in 3.5 release).

9. E-mail server for error-reporting

The administrator's e-mail address entered in the Administrator Logging page appears with any error messages generated by ColdFusion. This facility can help users report errors. This e-mail address can be overridden in the application framework page, Application.cfm. The administrator mail feature also requires configuring a mail server to handle sending automated mail messages from the server. In WAS, this feature, if required, can be made part of a JSP custom-error page. We should also configure a mail server to handle administrator mail messages.

10. Source-code management

On Windows systems, we can configure ColdFusion Server to use Microsoft Visual SourceSafe for source code management. This allows ColdFusion Studio users to access source files through a ColdFusion RDS server.

In WAS, the development tools Visual Age for Java and WebSphere studio have source-code control built-in (not WAS itself) as well as allow integrating with external source-code control software.

11. Distributed configuration

The CF server (prior to 4.5 release) must run on the web server machine. You can access ODBC data sources on remote machines if your database is capable of being accessed remotely, for example if you have the Open Client/C network libraries from Sybase installed you can connect to a Sybase System 11 database anywhere on the internet.

ColdFusion (4.5 onwards) can be configured in a distributed manner where the ColdFusion engine is running on a separate computer from the Web server.

WAS also supports this configuration, but the only caveat is that the communication between the web-server and application server is not SSL enabled if WAS is used in Remote OSE configuration.

12. Scheduling execution of pages

The ColdFusion Administrator includes a scheduling facility that allows us to schedule the execution of ColdFusion pages and to generate static HTML pages. The scheduling facility can be very useful for applications that do not require user interactions or customized output. Often, ColdFusion developers use this facility to schedule daily sales reports, corporate directories, statistical reports, and so on: Information that is more often read than written.

Instead of executing a query every time the page is requested, the static page is served up to users containing information generated by the scheduled event. Response time is faster since there is no database transaction, just an HTML page. ColdFusion allows us to schedule pages to execute on a daily, weekly, or monthly basis. We can specify a time of day for execution, and we can schedule a page to be run only once on a specified date. ColdFusion writes information about all scheduled events to a log file you can view to verify that events occurred or to troubleshoot scheduled events that did not execute properly.

WAS does not support scheduling actions, and hence we need to simulate such a thing by writing special programs which execute as daemon processes and perform these specific actions for us at a scheduled time/under certain conditions.

13. Clustering

ClusterCATS for ColdFusion is a Web server clustering technology that provides load balancing and failover services that assure high availability for your Web server and ColdFusion Server (CFS). ClusterCATS lets you cluster distributed servers into a single, high-performance, highly available environment of Web server resources.

A cluster consists of two or more Web servers located on a LAN or across a WAN. Web servers included in a cluster operate as a single entity to provide rapid and reliable access to resources on those Web servers. A cluster can help your Web site avoid the consequences of busy and failed servers -- slow networks. With ClusterCATS you can avoid bandwidth, latency, and congestion problems.

In WebSphere, load-balancing at the web-server level is done using WebSphere Performance pack(software level load-balancing) or Cisco Local Director (for hardware level load-balancing). Load-balancing at the application-server level is built into WebSphere itself. The functionality provided by WebSphere Performance pack closely matches that of ClusterCATS. ClusterCATS can augment an existing hardware-based load balancing solution by easily integrating with established devices, such as Cisco's LocalDirector. This particular feature is not supported by WebSphere Performance pack.

If a server within a Cold Fusion cluster fails or becomes unavailable and is no longer transmitting an IP signal, another available server in the cluster will automatically assume the IP address of the failed server. Subsequent HTTP requests that are bound for the downed server will be redirected to the server that has taken over for its failed counterpart. When the failed server is fixed and comes back online, the server that has assumed the failed server's IP address relinquishes it to the fixed server. This process is known as IP aliasing. WebSphere performance pack does not support this.

ClusterCATS also lets you configure automatic e-mail-based alarms for a wide range of failure types. This feature ensures that you can attend to failed servers quickly and minimize server downtime. During regular business hours, you can scan and read your e-mail as usual. For off-hour periods, you can configure your pager to receive e-mail notifications for 24x7 notification capability. This is also not supported by WebSphere performance pack.

5 Summary

In this document, we have outlined the issues involved in performing a ColdFusion to WebSphere migration. Some possible solutions have also been suggested. As described, the migration effort is highly dependent on the extent of usage of ColdFusion proprietary stuff in the applications. In some cases, the migration effort could be considerable and exceed the original effort.