

Making the Case for

There are some compelling reasons

Do you really know what it means to run CFML applications on a J2EE server? How does it work, and why would you bother? There are many benefits you may never have considered. In this article, the first of a series, I'll answer these questions.



By Charlie Arehart

You may have heard that you can deploy your CFML on a J2EE server such as WebLogic, WebSphere, JRun, or JBoss – or on a servlet engine such as ServletExec or Tomcat. But what does this mean to a CFML developer, especially one who's not familiar with (or even interested in) J2EE?

Whether you're an organization with multiple applications running on a single server, or a hosting company, or even if you have just one application on your current CF server, I hope this article will help you see when and where CFML on J2EE can make sense. I'll conclude by pointing you to resources for learning more about this topic.

The Two Main Reasons to Run CFML on J2EE

Running CFML on J2EE really isn't all that different from running CFML the way you're already used to, as I'll show in the next article. For now, just take my word for it that as a CFML developer, you really don't need to worry about most of the details of the J2EE platform, nor do you need to understand anything about Java to appreciate the most significant reasons for considering this alternative. It's just your CFML running on a J2EE server.

The bigger question, of course is simply why a CFML developer would bother – or why a shop moving to J2EE would care – about preserving their CFML investment. They're two sides of the same coin, but very different perspectives, and they hint at the two compelling reasons for running CFML on J2EE.

I'll start with the more obvious reason: you're in a shop moving to J2EE anyway. But I'll conclude (and spend more time) with an entirely different motivation: even if you don't know or care about J2EE, there are many benefits to deploying

CFML on a J2EE server that we just don't get on a stand-alone CFML server.

The thing is, you won't hear any of these reasons explained to you this way by the typical advocates for J2EE.

Reason 1: You're Facing an Organizational Move to J2EE

The first and most obvious reason to consider this alternative is simply if you work in a shop where you're being told that the organization is making a strategic move to J2EE. They're bringing in BEA WebLogic or IBM WebSphere (or choosing any of a number of alternatives like ServletExec, Sun ONE, Oracle, Tomcat, JRun, etc.), and word has come down that it's time to either rewrite or retire the CFML apps.

What does this mean to you as a CFML developer? And what about all that CFML you've written?

Some organizations have developed very complex, enterprise-class CFML applications that they rely on – or perhaps a small part of the organization relies on them, even if the larger IT organization does not. Simply rewriting those as JSPs and servlets (the lingua franca of J2EE servers) is not going to be easy, nor will it happen quickly.

Such organizations may face expenses of millions of dollars and several man-years of effort to do such a conversion. While there are tools that purport to do such conversions automatically, many have found them wanting and inadequate for complex applications. Then, too, what happens to the ever-growing requests for new functionality during this downtime?

CFML on to go this route @ J2EE

Moving CFML Developers to J2EE?

What happens to CFML developers in such a move? They may hold a tremendous repository of domain understanding. If an organization shifts to J2EE, what do they plan to do with the CFML developers? Let them go? Retrain them?

Becoming a professional-class J2EE developer isn't something you can learn in a couple of weeks, or even several months. Beyond learning the core Java language (and its libraries), you also need to learn JSPs and servlets (and their APIs), and typically also EJBs (Enterprise JavaBeans). Then there are the debates about which of those to use and when, how to apply design patterns, use of MVC and/or frameworks like Struts, etc.

I discussed this challenge in more detail in a *Java Developer's Journal* article entitled, "The Many Sides of J2EE Development," (*JDJ*, Vol. 6, issue 11) available at www.sys-con.com/story/?storyid=36739. The complexity of J2EE development using pure Java is proving to be its Achilles heel.

Many organizations find that their Web app backlog grows significantly when they go the pure J2EE route – this despite its fundamental focus on reuse. But its adherents simply don't think there's any viable alternative, and few of them have heard of the possibility of deploying CFML on J2EE – if indeed their purist intentions would let them even consider it.

As a consultant you can find lots to learn and keep yourself ever-growing and ever in business serving a J2EE environment, but if you've got a job to do (and/or have apps that need to be built) you won't be doing it quickly in a pure J2EE development mode.

Solving the Architecture Dilemma with CFML on J2EE

Here's the key: What if you could keep the CFML you've created and deploy it on a J2EE server? That's the focus of this series, and it at least saves you rewriting your CFML to get it to run in the J2EE environment. If your organization is moving to J2EE you can deploy your CFML as a J2EE Web application, which simply means running the CFML on the J2EE server. After that, you can slowly integrate your application with other J2EE applications and components. Heck, you might even be able to pare down that application backlog by doing some of the apps in CFML instead of pure J2EE.

One of the arguments you may face is that your IT management will not install CFMX or any other stand-alone CFML server (or they want to remove the existing one), but that doesn't mean you can't run CFML anymore.

Most CFML developers run their applications on a stand-alone CFML server. Whether that's an older 4.x/5 release of ColdFusion Professional or Enterprise; or CFMX Standard or Enterprise; or BlueDragon Server or Server JX – these run as a service on Windows or as a daemon on Linux or Unix.

However, what I'm talking about here is running your CFML on a J2EE server, which means you have no CF server (and no BlueDragon server). You just run your CFML as a J2EE Web application on a J2EE server such as WebLogic, Websphere, ServletExec, JRun, Tomcat, etc., using either the second or third installation options of CFMX Enterprise or BlueDragon/J2EE edition.

I'll explain in the next article what all this means, but for now just know that to the J2EE folks (including the server administrators), it really can be transparent to them that

you're even running CFML. All you give them is a J2EE Web application, and they deploy it like any other J2EE application. (Actually, that's how it is with BlueDragon/J2EE. With CFMX running on J2EE there's more involved both in building the Web app and after deploying it, so it's not quite as transparent. But each achieves the end result of running CFML on a J2EE server.)

KEEPING THE CFML DEVELOPERS DOING CFML

An important point to keep in mind as a CFML developer is that if you have to (or want to) run your CFML on a J2EE server, you don't need to learn anything about Java, JSPs, servlets, EJBs, or any other J2EE-based features. You can continue doing development in CFML (and using your preferred editor). It's just that your application is "deployed" on a J2EE server.

I'm not saying that pure J2EE development is bad – indeed, for a Java developer it's the most natural Web application development platform possible. You may even want to learn more about it just to add to your toolkit. In fact, there are great things you can do with CFML to integrate with JSPs, servlets, EJBs (and even JMS and other APIs) if you like, whether you write them or someone else in your shop does. It's just that you don't *need* to understand those things to deploy your CFML on a J2EE server.

Reason 2: To Leverage Benefits of Running CFML on J2EE

Perhaps you're reading this and thinking, "What's all this got to do with me? My shop isn't moving to J2EE." I'd like to show you the benefits you can obtain by running your CFML on J2EE – benefits that you just don't get running CFML as you do on stand-alone servers.

For now, just accept that running CFML on J2EE is easy. I will demonstrate the benefits more clearly in the future article(s) as well. In the meantime, note that you can get started with the documentation from Macromedia and New Atlanta Communications, as well as in the resources I list at the end of this article.

With your CFML running as a J2EE Web application it can leverage all the same benefits any J2EE Web application would reap from running in a J2EE server environment. Some of those benefits are unique and not available when running CFML on a stand-alone server.

Application Management Features

J2EE servers are designed and built to support enterprise-class application processing. As such, they typically include features that would be useful to CFML developers deploying their applications on such a J2EE server. These include the following.

CLUSTERING, LOAD BALANCING, AND FAILOVER

Most J2EE servers offer mechanisms for defining a set of physical servers as a cluster such that they can operate in conjunction with one another, cooperatively handling requests (typically in a high-volume environment) so that no single server becomes overwhelmed (load balancing) and taking over requests being handled by a server that goes down (failover).

Some J2EE servers offer only one or another of these features (if any), perhaps leaving such things as load balancing to external devices, protocols, round-robin DNS servers, or even external Web servers. This brings up another point: some J2EE servers contain a built-in Web server, which itself may or may not offer such features as load balancing and failover.

The stand-alone versions of ColdFusion MX and BlueDragon each offer their own built-in Web server, but both describe it as being intended for development and testing only. As for clustering, ColdFusion MX Enterprise does offer that capability as well. See the Macromedia documentation for more details on when and how this works.

SESSION REPLICATION

Still another benefit offered by enterprise-class J2EE servers when a number of server instances are grouped together, as in the discussion of clusters above, is the ability to replicate sessions among the servers. In other words, the servers are declared to share their sessions with each other (or with one or more of the others backing it up), so that as changes are made to a session on one server, those changes are replicated to the others that back it up.

The goal, of course, is to permit one or more servers to serve as a backup or failover server to handle requests for a server should it crash or otherwise become unavailable. With simpler failover systems it's not possible to replicate sessions, so you must instead declare that the servers in the cluster use "sticky sessions" (as with the clustering in CFMX Enterprise). In that approach, the clustering system simply ensures that each user remains associated with just a single server, and if that server fails there is no way to preserve the user's session via failover.

Clearly, session replication offers a substantial improvement in application reliability and availability. Like many such benefits, it comes at a cost: the overhead of keeping sessions replicated among the participating servers. Some J2EE servers offer a means to temper this cost (perhaps varying how often sessions are replicated or what session data is replicated).

SESSION PERSISTENCE

A subtle twist on the previous discussion – but an important one, is the notion that active sessions in a server instance can be preserved over server restarts. It's not about replication to another server. Indeed, it's a feature that's particularly useful if you're *not* using clusters and session replication. Session persistence could be useful for any application that uses sessions. This way, if your server goes down, when it comes back up the active sessions are restored to memory and still available. If that restart happened quickly enough it's possible that an end user might never experience a problem (if the user doesn't make a request while the server is down).

The sessions may be persisted to a file system, a database, or something else. Again, there's certainly a cost to enable this feature, since the persistence operation must again take place either on every session change or at some interval. The question is simply whether the benefit is worth the price. But it's another benefit available only when running CFML on a J2EE server.

—continued on page 40

One more point: the ability to leverage both session replication and persistence is something that comes only when you enable “J2EE sessions” in the administration console of either CFMX or BlueDragon when deploying a CFML Web application on a J2EE server. When you do this, you’re in effect telling the J2EE server to take over control of session management, rather than having the CF or BlueDragon CFML engine handling sessions.

This reflects one of many kinds of mechanisms that previously we’d look to the CFML engine to provide, but the J2EE server may provide it as well – and perhaps do it better – or at least handle it better in a clustered environment. The next feature is another example.

DATASOURCES DEFINED AT THE J2EE SERVER LEVEL

Just as we could give up control in our applications to have the J2EE server manage our sessions, we could also give up control to the J2EE server to define and manage our datasources. This can have many benefits. It helps first, though, to clarify that what this means is simply that rather than define a datasource in the CF or BlueDragon admin console, you would instead define it in the J2EE server’s admin console. The features and interface are generally very similar.

And you would still refer to the datasource name in CFQUERY (and related tags) just as you do now, but instead of being a datasource name defined in the CF or BlueDragon admin, you would refer to the datasource name defined in the J2EE server.

The difference is that the J2EE server may offer some features that you don’t get in either CF or BlueDragon’s admin console or datasource definition. For instance, the J2EE server may perform connection pooling better, it may offer more updated database drivers or support additional databases, or it may offer additional benefits in a clustered environment.

And, as I will detail later here, you can arrange to run several applications on a single server. If you define the datasource at the J2EE server level, then that one configuration can be leveraged by any of the several applications.

One last thought on this topic is that since the J2EE servers may have far more customers (there are a lot more Java developers in the world than CFML developers), you may also see improvements coming about more quickly in some areas (in the area of J2EE datasources, for instance) than you would in the CFML world.

BUILT-IN APPLICATION TRACKING

J2EE servers often offer mechanisms to track and manage applications as they run. This can include reporting how many requests have been made, what kind of processing they’ve performed, how long they’ve been running, how many sessions have been started and their runtimes, what sort of datasource processing has occurred, etc. (I’ll add that BlueDragon offers this information about CFML requests in all editions, including the stand-alone server ones. While ColdFusion Enterprise used to offer a Server Reports feature with some of this information, this is no longer available in CFMX.)

Running Several Applications at Once

The advanced application management features described

above are both needed and enhanced by the fact that when a J2EE Web application is deployed on a J2EE server, that application is in many ways segregated from other Web applications deployed on the same server.

What this means is that unlike different applications being separated from each other only because they’re in different subdirectories, as happens on stand-alone versions of ColdFusion and BlueDragon, when you deploy CFML as a Web application on a J2EE server, it’s segregated from other Web applications on that J2EE server.

There are a couple of levels of segregation. If you’re familiar with the concept of “independent instances,” I’m not even talking about that in this section. I’ll discuss that concept later in this article.

SEPARATE APPLICATION MANAGEMENT

When CFML is deployed as a Web application on a J2EE server, that Web application can be stopped and started independently of any others. When an application is stopped, requests sent to the server are rejected, indicating that the application is unavailable. This could be useful during maintenance operations.

This is a level of granular control over applications that we don’t experience in either ColdFusion or BlueDragon stand-alone servers, where you can only stop and start the entire server, thus affecting all applications running there. (Note that this isn’t really stopping the application, but instead just stopping requests to it.)

This capability is extended with even more power when running as multiple independent instances, which is again a different and valuable feature explained later.

Some J2EE servers also offer a means to “reload” the Web application, which can be useful when configuration changes have been made to underlying XML and Java files.

SEPARATE ADMINISTRATION CONSOLES AND CONFIGURATION

Extending on the fact that each Web application is deployed separately from one another, an additional benefit of deploying CFML on a J2EE server is that each application can have its own administration console. I’m referring here to ColdFusion’s or BlueDragon’s console.

This is a powerful benefit that means you could give the owners of a single application the ability to both administer their application environment and set up various configuration aspects of their CFML environment. Imagine enabling debugging for one application but not for another, or enabling a datasource in one and not another, or turning on trusted cache for one but not another.

Still another benefit comes in the area of security: you can configure the Web application (using the Web application’s web.xml file and/or features unique to each J2EE server) to configure different Web applications to have different security settings. These could influence how Web application requests are authenticated and authorized (if you choose to use that) as well as controlling what directories can be accessed by code within the Web application.

These are clear benefits for hosting environments, but even an intranet with multiple unrelated applications, or a compa-

ny offering a service as an ASP (application service provider), could benefit.

Other features you might prefer to control in this granular manner include virtual directory mappings, custom tag paths, CFX tag registration, etc. (as well as things like whitespace compression, global error handlers, application timeouts, and client variable storage, though these can also be controlled at an application level via `application.cfm`.)

I will point out that one challenge with running multiple applications, each having their own configuration settings and CF/BD administration console, is that there's no current mechanism in either ColdFusion or BlueDragon to propagate changes in one application so that they're applied/synchronized with other CFML-based Web applications on the J2EE server. For now, this is something you need to manage yourself. (On the other hand, many J2EE servers will automatically deploy a given Web application across all servers in a cluster so that if the first one is configured correctly before deployment, its configuration information will also be deployed with the app across all the servers.)

RUNNING MULTIPLE VERSIONS OF APPLICATION, CFML ENGINE ON ONE SERVER

Continuing the idea enabled by the previous feature, since each application is effectively segregated from all others, and since the configuration info is contained within the Web application along with your CFML code, it's also possible then for each Web application to be created using different versions of the CFML engine.

I'm going to discuss only BlueDragon here, since I'm not as clear about how CFMX on J2EE works in this regard. In the case of BlueDragon, under the covers the configuration of the Web app as deployed on the J2EE server tells the J2EE server that requests for CFML pages should be handed to the BlueDragon engine. The guts of that engine – how to run CFML – are included within the Web application itself (the directory or WAR file, for those who understand such things).

So you could deploy some CFML within a Web application running one version of BlueDragon (say, BlueDragon 3.0) and then another running the CFML as BlueDragon 6.1 (our latest release). You could use this to test different betas or upgrades within a release as well.

ent from those I've described so far, though it's easy to confuse them. I'm referring to the concept of running multiple independent instances on the J2EE server, a phrase you might have heard quite a bit about if you've been following CFMX closely.

There have been articles both from Macromedia and in *CFDJ* (the latter listed at the end of this article) describing this with respect to CFMX. The thing is, there's nothing special about CFMX that enables it. It's something that you can leverage with BlueDragon deployment on J2EE as well. In a future article I'll discuss deployment of CFML on J2EE as independent instances using BlueDragon.

With this feature, some J2EE servers offer the means to create separate virtual servers (sometimes called "instances"). In stand-alone implementations of CF and BlueDragon, all applications run under a single server instance. With a J2EE server offering this feature, each virtual server is independent of the others in terms of the underlying JVM that controls the server.

APPLICATION ISOLATION

The biggest advantage to each being controlled by its own JVM is that if the application(s) in one virtual server/instance fails or slows to a crawl, the application(s) running in another virtual server/instance should be able to continue to run unaffected.

Also, each JVM can be configured independently, allowing the developer to change JVM settings such as the amount of memory enabled, the version of the JVM, the classpath for the JVM, etc.

This is different from the application segregation features I described earlier, in which each deployed application can be controlled separately and has its own admin configuration. If they're not running in separate virtual servers/instances, then you get the benefits of application configuration segregation but not of process isolation.

By the same token, even if you don't (or can't) choose to enable independent instances, there are still those benefits that come from the segregation of J2EE Web applications, even in a single virtual server/instance.

PERFORMANCE/THROUGHPUT IMPROVEMENT

One thing that is indeed unique to using multiple independent instances is a possible performance/throughput improvement by using more than one instance on a single

**"It's perhaps a dirty secret (in the opinion of J2EE folks),
but you can do pretty much anything in CFML that you
can do in JSP or in a servlet"**

You could even extend this to creating different Web apps for different stages of development, perhaps one Web app for the development stage and another for QA testing, etc. It's really powerful to be able to run multiple versions of the application or different CFML engine versions on a single server.

Leveraging Multiple Independent Instances

The final aspect of application segregation is one that's differ-

server. This may seem illogical; if you run more than one instance, you're clearly spreading out the available resources of the machine. How can that improve performance?

The secret is in the fact that while each JVM has less memory to work with (since it's sharing a fixed amount of available memory), it also is therefore responsible for managing less memory when it comes time to do garbage collection, a JVM feature used to manage unused memory when processes leave

objects orphaned. (It's an inherent aspect of how Java-based applications work – and while we're programming in CFML, under the covers, everything is really happening in Java.)

ADDITIONAL CLUSTERING BENEFITS

One other cool thing about multiple independent instances is that by enabling them on a single machine, you can in effect “cluster” these multiple instances and then leverage the features discussed earlier in the Application Management Features section. These include clustering, load balancing, failover, starting/stopping applications independently, session replication, and session persistence.

And just as having segregated application configuration among multiple Web applications (even in a single instance) offers advantages, the same is obviously true of Web apps deployed on independent instances.

Of course, another benefit of the instances being separate from each other – and perhaps the primary reason for creating such virtual servers – is that you can generally configure Web servers to support multihoming, in which a single Web server supports multiple IP addresses or domain names, such as www.mycompany.com and services.anothercompany.com, each running out of the Web root of the different instances.

Reducing Licensing Costs by Reducing Machine Quantity

There's another major benefit that can be derived from

the combined features described in the previous two sections on supporting multiple applications. Whether deploying them on a single instance or on multiple instances, there's a real chance that you could save money using these features.

Consider that each offers the means to create multiple versions of your applications, and possibly even to cluster them, etc. You can clearly gain benefits that would otherwise require you to have separate physical machines (and each would of course need its own license for ColdFusion or BlueDragon).

Whether for clustering, or for running different versions for testing/QA, etc., each of those separate Web applications (or instances) runs under a single license. Both products are licensed on a per CPU basis when deployed on a J2EE server, not per instance or application. You could actually save money by deploying on J2EE.

Integrating CFML with J2EE

Another major benefit of deploying CFML on J2EE is that you can integrate your CFML with other J2EE components such as JSPs and servlets, EJBs, JMS queues, and more.

This is powerful. If your company is moving to J2EE, you may well feel a pinch from trying to do things in CFML, even once you've convinced management that you're really running on J2EE. They may know that other applications or components have been built in pure J2EE.

Ad

No worries. It's perhaps a dirty secret (in the opinion of J2EE folks), but you can do pretty much anything in CFML that you can do in JSP or in a servlet. Want to call an EJB? You can. Want to include a header that's already been written in JSP? Go for it. Need to call a Java library method that does something you can't do in CFML? It will work. Are you curious to see if a CFML page can be viewed in a Struts application, or be rendered as a portlet? People have done it.

Your CFML running as a J2EE Web application can benefit from other aspects that apply to pure J2EE Web applications, such as integration with servlet filters and listeners. I already mentioned how you might also use J2EE Web application security authentication and authorization. Each of these is a feature that applies to any template requested in a J2EE Web application that's been configured to use the feature; they're not unique to JSPs and servlets.

Supporting Various Platforms

Finally, one aspect of deploying CFML on J2EE is something that may not seem to be an obvious "benefit" if you look at J2EE servers and wonder what you can do with them using CFML.

With stand-alone versions of BlueDragon and CFMX, the vendors (New Atlanta and Macromedia, respectively) have had to build separate installers to be executable on various operating systems.

With deployment on J2EE, however, you're not installing a stand-alone CFML server. You're just deploying the CFML in (or as) a J2EE Web application. That Web application can be created and implemented in a platform-independent way.

Again, I'll speak more for BlueDragon here, as I have more experience with it. The BlueDragon/J2EE edition is available as a zip file. When you unzip it, it's just a directory containing the documentation and a subdirectory that is itself the J2EE Web application. You can make a copy of that subdirectory, name it whatever you like, copy your CFML into it, and deploy it onto any J2EE server.

The point is that there is no platform-specific installation routine to be executed to create the skeletal J2EE Web application into which you place your CFML to be deployed onto a J2EE server. That's any J2EE server, on any platform. There's nothing OS-specific about the J2EE Web application, at least with BlueDragon/J2EE.

Bottom line: you can run your CFML on any platform for which there's an available J2EE server. I've been asked if BlueDragon would allow a CFML developer to run their application on such things as a mainframe or an iSeries/AS 400, and such operating environments as FreeBSD, OpenBSD, and even NetWare. My answer is the same: if there's a J2EE server for them (and there is), then you can deploy BlueDragon/J2EE there and run your CFML application. It's truly CFML everywhere!

Summary

I hope I've helped persuade you that running CFML on J2EE can not only be a great solution if your organization is moving in that direction, but even if you've never considered J2EE before.

If You're Facing a Mandated Move to J2EE

If you're facing an inevitable, inexorable push in your environment toward J2EE, then running your CFML on J2EE gives you several benefits:

- You can just park the CFML on the J2EE server and keep it running.
- You can start integrating with Java, if you'd like, at your own pace.
- You can start converting CFML apps to JSP/servlets – again, at your own pace.

In shops where a move to J2EE is being mandated from the top, this is a fight for the very survival of CFML. You've invested far too much in your CFML apps (and talent) to just throw them away.

You'll be arguing against architecture gurus who see CFML as a toy, if indeed they've even heard of it. Their experience may be tainted by the CF servers of old. You'll need to help them understand how all this works. I hope this article serves as a starting point for your discussions.

You'll also benefit from understanding more about how these J2EE architects and developers see things, and the environment in which they work. I will discuss these issues further in an upcoming article.

If You're Motivated by the Benefits of CFML on J2EE

Even if you weren't interested in deploying on J2EE before, I hope you can see now why it's so exciting. There are just so many benefits that you can't get by running your CFML on a stand-alone server:

- Application management
 - Clustering, load balancing, and failover
 - Session replication and persistence
 - J2EE datasources
 - Application tracking
- Running several applications at once
 - Separate application management
 - Separate administration consoles and config settings
 - Running multiple application versions/engines at once
- Leveraging multiple independent instances
 - Application isolation
 - Performance/throughput improvement
 - Additional clustering benefits
- Reducing licensing costs by reducing machine quantity
- Integrating CFML with J2EE
- Supporting various platforms

As you can tell, I'm excited about this prospect. Others have been too, and the resources that follow describe many of these features in more detail.

Resources

I'll be giving more details about working with CFML on J2EE in a future article, but if I've really done a good job of motivating you, you may want to get started right away.

Installing CFMX for J2EE Deployment

The ColdFusion manuals explain their approach to deploy-

ing CFML on J2EE. There are actually several resources, and to be honest, the approach has changed three times since CFMX was first released. (There was “phase 1” and “phase 2” in the CFMX timeframe, when the capability was called CFMX for J2EE, then another change in the CFMX 6.1 release when it was bundled into the CFMX Enterprise product as an additional installation option.)

The best place to start may be the sections of the Macromedia site devoted to the subject, including:

- *ColdFusion MX and the Java Platform:* www.macromedia.com/software/coldfusion/j2ee
- *Deploying ColdFusion MX 6.1 on J2EE Application Servers:* www.macromedia.com/go/cfmj2ee-cert
- *Installing the J2EE Configuration:* <http://livedocs.macromedia.com/coldfusion/6.1/htmldocs/installj.htm>

Installing BlueDragon/J2EE

- *BlueDragon Documentation:* www.newatlanta.com/products/bluedragon/self_help/docs/index.cfm

Both CFMX and BlueDragon are available for free trial and as free development editions.

Integrating CFML with Java/J2EE

- *Integrating J2EE and Java Elements in CFML Applications:* <http://livedocs.macromedia.com/coldfusion/6.1/htmldocs/java.htm>
- *ColdFusion MX/J2EE Hybrid Applications:* <http://sys-con.com/coldfusion/article.cfm?id=636>
- *Fun with Filters:* <http://www.sys-con.com/coldfusion/article.cfm?id=573>

Still another resource for learning more about J2EE integration with CFML is *Reality ColdFusion MX: J2EE Integration* (Macromedia Press) by Ben Forta, et al. Like other books in the “Reality” series, it’s got a different style and isn’t really geared toward teaching you how to do the integration. You’re more like a fly on the wall as the book goes through several different projects, but you’ll surely pick up more information along the way. (I’ll add that Amazon and other sites list me as a coauthor. Actually, I pulled out of the book over editorial differences. This series of articles is more the style of presentation I’d wanted to make to CFML developers.)

Multiple Independent Instances

- *When One ColdFusion Is Not Enough:* <http://sys-con.com/coldfusion/article.cfm?id=626>
- *Using Multiple Server Instances:* <http://livedocs.macromedia.com/coldfusion/6.1/htmldocs/clusteri.htm>
- *Advantages of using multiple instances for ColdFusion MX for J2EE:* www.macromedia.com/devnet/mx/coldfusion/j2ee/articles/multiple.html
- *Installing and Configuring ColdFusion MX 6.1 Multiple Instances with IIS and Apache Virtual Hosts:* www.macromedia.com/devnet/mx/coldfusion/articles/multi_instance.shtml

- *Introducing Multiple Server Instances in ColdFusion MX Enterprise 6.1:* www.macromedia.com/devnet/mx/coldfusion/articles/multiple_61.html

Making the Case to J2EE Folks/Management

Finally, I’ll point out a couple of other previous *CFDJ* articles written by my colleague (and boss), Vince Bonfanti. These offer still more discussion on the general benefits of running CFML on J2EE, but they’re more from the standpoint of convincing J2EE developers and management why CFML on J2EE is different from what they might expect:

- *It’s Not ColdFusion – It’s J2EE!:* <http://sys-con.com/coldfusion/article.cfm?id=588>
- *Making the Case for CFML:* <http://sys-con.com/coldfusion/article.cfm?id=618>

About the Author

Charlie Arehart is CTO of New Atlanta Communications, makers of BlueDragon. A Macromedia Certified Advanced ColdFusion developer and trainer, he continues to support the CFML community, contributing to several CF resources, and speaking frequently to user groups throughout the country.

charlie@newatlanta.com

Ad