From Server "A" to Server "B" - Details Matter When Re-hosting

Posted At : November 16, 2007 1:14 PM | Posted By : Mark Kruger
Related Categories: Hosting and Networking

You probably know that **CF Webtools** hosts a fair number of sites in our own burgeoning data center. We are not a *commodity* host (i.e. Godaddy or HostMySite). Instead, we host a large group of **Farcry** sites, several dedicated servers, and a large group of very complicated Coldfusion sites with special requirements (data feeds, point to point encryption, data aggregation and third party secure services etc.). Our hosting has grown substantially in the last year and has become an excellent revenue center for us.

One of the type of projects we find ourselves doing with some regularity is a "site re-host". Usually a company has an application that clearly requires more help and attention than can be gained using a commodity host and self service control panels. Furthermore, such sites have often "evolved" from widgety little intranet type sites with B2B tools, special custom ecommerce applications or homegrown CMS capabilities into monsters of maintenance with hundreds of pages (many of them titled stuff like "order_bak.cfm" or "index.old"). Incidently ***never*** leave a file like "index.old" on your web site. If the web server is in default config mode it can serve that file up to your user without running it through the Coldfusion engine. That exposes your code and makes you easier to attack.

In any case, re-hosting a site seems like a simple enterprise. If your site consists of a database and codebase then it *can* be simple - but the devil is in the details (and in the cat as my Dad use to say). Here is a rundown that you might find useful.

Preliminary Tasks

Before you begin you need to make sure the site owner agrees to *no code changes* until the re-host is complete. If code changes are critical while the project is underway you will need to re-synch the code "on the fly" during deployment (or make some other arrangement). Usually that's a bad idea and results in errors and customer service issues. In any case, the following steps should be taken *prior* to moving to a production server.

- Download the code onto a test Coldfusion server. Make sure your test server mirrors the file paths of the final production server if possible. Otherwise you will end up changing code twice. If an SSL cert is being used get an export of it (if possible) from the site host.
- Download the database backup and restore it to the right platform. We don't handle Access sites so for us it is either MySQL or MS SQL.
- Configure the web server to serve the site. Pay attention to default documents and virtual folders.
- Search the code for calls to COM objects, Java classes (that are not native to your Coldfusion install) and CFX tags. Each of these items will need to be discovered and either installed or re-written. For example, we have seen a lot of code using the C++ CFX Image tag. We usually re-tool this code to use image.cfc. Sometimes COMs and Jar files cost license fees. Make sure you communicate this possibility to the customer as a part of the estimate.
- Configure the CF server with the proper data sources, mappings, verity collections, web services etc.

- Search for CFMAIL tags to determine if there are special mail settings that need changing (like specifying or not specifying a server as the case may be).
- Search for file paths and operations like *CFFILE* and *CFDirectory*. Each will need to be modified to reflect the new environment. NOTE: If the test environment does not mirror the live server this task will need to be done again when you move to production. If the code is well written chances are you will only need to change a few application or request variables (or a config file or something). Be prepared however - that is *often* not the case.
- Test the code. Make sure that it does what it is supposed to do.
- As you test, clean up the file system removing those ".bak" and "_bak" files (and all those files with dates in the name :)
- If everything checks out to this point you are ready to move the code to production (don't forget about the file paths). Move it to production, set up the data source and re-test.
- At this point you need to install your SSL certificate if one being used. This can be tricky on windows and requires the use of the certificate MMC. Verisign has some excellent instructions on how it is accomplished. In Apache it's a breeze.

All right! You have your code on the production server! What's next?

Domain and Email Tasks

Time to setup the domain records and email. If the user is using a third party DNS then this step will be part of the deployment script. Otherwise, set up domain records as needed on your DNS server. When the site owner switches the registrar delegate to your DNS server your site will "begin" to be live on your servers (more about this later).

Don't forget about email. You will need to get a list of email addresses (if you are supporting the customers email) and set them up on your email server along with passwords. If you are using POP then you should assist the user with setting up the account in their email client. They will have both accounts set up (the old and the new) so make sure you have a unique way of hitting both servers. This is important because while the domain propagates it's possible for email to go to either email server. If email is important to the customer then you should set up 2 email accounts in the client - the old one can be dropped a few days after the DNS is switched over. Remember this is all *in preparation* for the switch over.

Write a deployment Script

This is a 1 page list of things you will do *and the order in which to do them*. These tasks are the "go live" tasks. When you get to the end of the list the site will be live. This is important in order to minimize downtime and *loss of data*. The list should include your plan for keeping the data in synch, adding maintenance messages to the old and new server, changing DNS etc. Remember, re-hosting a site will bring on unique Data synchronization problems.

Data Synchronization

Take an ecommerce site as an example. After you make your DNS changes it will take 12 to 48 hours for those changes to "propagate" throughout the Internet. It takes that long for all the thousands of DNS servers to "know" about your changes. During that time the potential exists for orders to be placed on both the *old* and the *new* servers. What should you do about that? There are a few of approaches.

1. Take Your Medicine

In this approach you simply take down the old server and put up a message like "down for maintenance. You take a hit on traffic for 12 to 48 hours but it is the simplest "brute force" method to get this done with no data problems. If the site is mission critical then this is not an option and you will need one of the other approaches.

2. The Redirect

With this approach your deployment script includes synching the database and then putting a redirect on the *old* site that forwards the user to the new site via the IP address. Note - you have to forward to the IP address because there is no way to "force" the browser to resolve the domain name to resolve as your new IP address. The browser is dependent on whatever DNS servers it can query. Consequently, you have to redirect to the raw IP address - i.e. <cflocation url="http://xxx.xxx.xxx.xxx"/>.

What's the problem with this approach? SSL will not work as expected. A certificate is bound to an IP *and to a domain*. Your cert expects "www.123.com" not 123.123.123.123. You can still use SSL and encrypt the page using the ip address like so - https://123.123.123.123. The page will indeed be safely encrypted and from a technical standpoint just as secure as if you used the domain. But the browser will throw warning messages because the domain doesn't match. This will naturally strike fear in the heart of the user and possibly cause a seizure - or worse they could call or email you.

Aside from the warning message, most sites that use SSL hard code the links into specific sections of the site that are supposed to be protected (like shopping cart checkout). The redirect method may result in the user showing up on the new site via IP address, but being redirected back to the old site via domain as soon as SSL is required. Again, using ecommerce as an example, the user would put things in his cart and then click "checkout" - whereas the browser would head back to the "old" site. But the "old" server would not know what to do because nothing would be in the cart on the "old" server.

3. Two Servers One Datasource

To make this work usually requires the cooperation of the "old" site host. Other than general goodwill there is little incentive for them to do this. But if you can find a disgruntled technician ready to "stick it to the man" you might be in luck. This method ensures maximum uptime with minimum disruption, but it does pose something of a security risk so beware. In this scenario you set up a new datasource on the *old* server that points to the *new* database server (are you with me so far). In order to do this you will have to temporarily fiddle with firewall rules. I recommend opening the port only to the IP address of the OLD server. If you have a security problem you will at least suspect that it originated on the old host network and not on the web at large (at least that is the idea).

During deployment you would take down the "old" site temporarily, synchronize the data from the "old" server to the "new" server, then re-point the code on "old" server to use the datasource on the "new" server. When you bring up the old server and change the DNS both the old and new servers will be live - but using the same datasource. This results in little downtime and keeps your data in synch. You can even do this in advance of the actual switch over if you have time constraints. Please note, although the same datasource is used, this does not mean that sessions are synchronized. So you

want to make sure that if the user arrives on Server A, that they stay there and are not re-directed to server B.

Summary

As you can see re-hosting a site or application is not as simple as moving code from server A to server B. There are many many details to navigate. Missing a small item (like a jar file for example) can result in lost sales, disruption of service or even a hung server. But if you pay attention to the details you can do it successfully with little downtime.