Introducing CFANT

Deploying apps can be a pain. It's not so much the copying of files from one place to another, but all the related tasks you have to remember to do!

As an example, we tend to have local development, staging, and production servers. Each of these servers tend have different configuration settings. On staging I might have a datasource called myAppDev. In staging it might be known as myAppStaging and in production it could be known as something completely different.

Then there are the issues of mappings, custom tag paths, event gateways, email servers, reloading applications, clearing template caches, etc, etc. Historically, I've avoided using a number of advanced features of ColdFusion because of the configuration required on the server and the requirement that I *remember* tasks that need doing on each deployment.

I'll wager that the vast majority of websites and applications are quite simple and can easily be deployed via a copy and paste process or FTP. However, the larger and more complex an application and hosting environment become, the more arduous this process gets. At this point you begin to lust for automation but there are some things that are just hard to automate. In these cases you need to automate as much as you can and have a script for any other steps you need to take manually.

Well, at Alagad, we've been pretty stuck at the last mile. For some of our apps we have an ANT build script which essentially calls a remote cfm file that runs another ANT file. This remote ANT file would export the application out of Subversion and to the correct location depending on if we're deploying to staging or production. Finally, it might make some configuration changes to the application such as setting Model-Glue and Reactor to production modes.

However, many of our applications run in production with the trusted cache enabled. Some of our applications require mappings or event gateways. Even if we manually set some of these up in the ColdFusion server, we still have to *remember* all this stuff when if we ever move to a new server or if we cluster our applications. Beyond that, there are still some things you have to do manually on each deploy such as clearing the template cache.

This problem started me thinking about the ColdFusion Admin API. As of version 7, ColdFusion shipped with a collection of CFCs which provided complete access to all the capabilities of the ColdFusion Administration interface. Why couldn't we create a set of reusable scripts which would use the Admin API to do things we would have to manually do while deploying an application? This weekend I decided to give it at shot and the result is a new Alagad offering known as CFANT.

CFANT provides a default mechanism for deploying applications remotely and also includes ANT tasks for everything you can do via the Admin API.

Deploying Applications Remotely

The CFANT project should be installed under your application's webroot in a directory named "cfant." This directory should be web accessible. If you're squeamish about this cfant has some properties you can tweak to change the the directory's name and location and you can always make your build.xml file not web-accessible.

Under this folder there is a build.xml file. This file is heavily laden with comments to help you out, but the file itself is really just a starting point. If you look at the file you'll see that the default comes with three targets: BuildLocal, BuildStaging and BuildProduction.

Let's start by looking at the BuildLocal target is what you would run locally to kick off the remote build processes. BuildLocal is also the default target. The BuildLocal target defines a few inputs. You can specify whatever inputs your script requires, but I recommend keeping the TARGET input at least as this specifies the name of the remote target you're executing. The rest of of the inputs would be inputs required for the remote target to run. Due to ColdFusion's case insensitivity, I recommend you uppercase any values that will be passed to a remote script.

All of the inputs and any properties you may specify need to be passed to the remote build runner. The remote build runner is a simple CFM file in the /cfant/build directory that is executed via HTTP by the local build target. The remote build runner runs ANT on the remote system, specifies the same build.xml file that you've been editing, and runs the task specified. Any arguments that come in through the url as passed as arguments into ANT.

Two things are implicit in this process:

First, you must have ANT installed on your server and it must be accessible via the path. Or, in other words, you must be able to run ANT from the command line without specifying the full path to the ANT executable.

Secondly, you must have already manually pushed CFANT to your remote server. The remote build runner won't run if it and the build XML and various other files it depends on are not there.

At this point your remote build script runs and does whatever you told it too. For a typical Alagad build script, we'd probably export code from Subversion, change some XML configuration, and reload the application via a URL.

But, I still need to use some of the features of the ColdFusion Admin API which is why I also included...

ColdFusion Administrator ANT Tasks

CFANT includes 320 ColdFusion administration ANT tasks. Each of these tasks map to a function on the ColdFusion Administration API. Each of these tasks are provided as a class under the bin/com/alagad/cfant folder and must be loaded using the <taskdef> ant tag.

To use any of these tasks you'll first need to use the <cflogin> ANT task. (No, this is not the same things as the <cflogin> coldfusion tag. The name is confusing, I know. There may be others that are similarly confusing.) Here's some sample XML:

```
<taskdef name="cflogin" classname="com.alagad.cfant.cflogin"
classpath="${thisDir}/bin" />
<cflogin adminPassword="${CFADMINPASSWORD}" />
```

Note that I use taskdef to load the cflogin task and then use the cflogin task. At this point I can use any of the other tasks.

Unfortunately, cfant doesn't come with any real documentation. Instead, you'll want to use the documentation for the Admin API. And, also unfortunately, that documentation is next to non-existent.

To work around this, I recommend browsing directly to an Admin API CFC specified in the documentation. For example, go to http://yoursite.com/CFIDE/adminapi/administrator.cfc. ColdFusion will prompt you for your administrator password and show you automatically generated documentation.

Any public method on any of the Admin API CFCs has a corresponding tag. For example, the login method on the administrator.cfc has the cflogin task.

Each of the arguments on the Admin API method has a corresponding attribute on the task. For example, the login method accepts four arguments: adminPassword (required), adminUserId (optional), salt (don't use), rdsPasswordAllowed (optional). The task has corresponding attributes which you use in the exact same manner. You'll see in the example above that the cflogin task was provided the one required argument of adminPassword.

If an admin function returns a value you can use a property attribute to specify an ANT property to place the results into. Here's an example using the cfverifyDsn task:

```
<taskdef name="cfverifyDsn" classname="com.alagad.cfant.cfverifyDsn"
classpath="${thisDir}/bin" />
<cfverifyDsn dsn="VirtualPM" property="dsn" />
<echo message="VirtualPM DSN is Valid: ${dsn}" />
```

Again, the <taskdef> tag loads the cfverifyDsn task which maps to the verifyDsn function in the admin API. This method accepts an argument named dsn which is the name of the datasource we're verifying. The method returns a boolean value and you can use the property attribute on the CFANT task to specify a property name. In the example above we then output the results of this using the ANT echo task.

Any function which returns a value (and the ColdFusion team has property indicated in their ColdFusion code returns a value) will have a property attribute. If the function returns a complex value this is dumped to text and the dump is returned.

How do the CFANT tasks work?

I'm glad you asked! Essentially, I wrote a generator.cfm which gets the metadata for each of the Admin API CFCs. Based on that, I generated a set of remote proxy CFCs that make the public methods on the Admin API remotely accessible. I then generated a simple Java class for each of the methods on the Admin API CFCs. This class ultimately extends the ANT task object and calls remote methods on the remote proxies. Returned values are parsed for errors.

Overall, because this is generated code and a new project, the vast majority of CFANT is completely untested. If you find a problem or have a suggestion please register with the project's Trac site and file a ticket. The Trac site is located here: http://trac.alagad.com/cfAnt.

You will be able to download CFANT from the Alagad.com products section as well as from Subversion at: http://svn.alagad.com/cfAnt/trunk/.