Ant4CF Documentation

Table of Contents

Introduction

Welcome to the Ant4CF documentation. This documentation is indented to help you integrate Ant4CF into your ColdFusion application's build and deployment process.

This documentation is specific to the Ant4CF project. This documentation assumes that you have some experience with Ant, or at least are capable of reading the Ant documentation.

If you are new to Ant, you will want to go to http://ant.apache.org/ and familiarize yourself with the project.

What is Ant4CF?

Traditional ways of deploying ColdFusion applications are a pain. Many applications need to have several configuration changes made, code needs to be pushed to one or more servers, and the servers have to be configured correctly to run the application.

This problem is not unique to ColdFusion. In fact all languages have similar deployment issues. Luckily, there are tools that can be used to automate the build and deployment process. Ant4CF is one of these tools for ColdFusion.

Ant4CF is a collection of tasks for Apache Ant that provide tasks specific to deploying ColdFusion applications. By combining this tool with other Ant tasks you can automate most build and deployment processes. And, by automating your build and deployment processes you can help insure your applications are always built and deployed correctly.

Read on to learn how to use Ant 4CF.

Installation

There are several ways to install Ant4CF, but all of them follow the same pattern. At it's simplest, you need to copy the Ant4CF folder out of the Ant4CF zip file and put it someplace accessible over the Internet. The simplest place to put this is directly under your CFIDE directory.

Experience with Ant4CF has shown that having a domain specifically for accessing Ant4CF is very beneficial. For example, you may want to setup a domain deploy.yourdomain.com that you can use to access the Ant4CF files.

If you are deploying your application to multiple servers or CF instances you will need to come up with a strategy to access the Ant4CF folder on each one of your servers and CF instances. There are several ways to do this. You may choose to either insure that the CFIDE folder is accessible under your application's root domain on each server and instance. Or, you may setup a domain specific to each server such as deploy1.yourdomain.com and deploy2.yourdomain.com.

Ant4CF depends on the Ant executable. Because of this, Ant must be installed on each server that Ant4CF is used to deploy to. You can install this wherever you want. **Once it's installed be sure that you can run Ant from the command line.**

In your Ant4CF installations you will need to edit the config.properties file ant change the pathToAnt value to the correct path to Ant on this server. This needs to relative to the server's root. For example, on *nix OSes this might be "/usr/bin/ant". On windows you will need to replace back slashes (\) with front slashes (/). As such, the correct path on windows might be "c:/program files/ant/bin/ant.bat", depending on where Ant in installed.

Please note that on *nix OSes the Ant executable file is simply "ant". On windows this is "ant.bat". Your config.properties file needs to specify the full path to the executable.

Also, if the path to the Ant executable has spaces you will need to surround the path in quotations marks.

When actually using Ant4CF you will need to copy the ant4cf-java-2.0-SNAPSHOT-jar-with-dependencies.jar into your Ant4CF lib directory. By default this directory would be "antlib" in the same directory as your Ant4CF build files. See the Basic Usage section for more information on this.

You will need to insure your server has at least Java 1.5.0_17 installed and running. To be clear, when you run Ant from the command line the JRE used must be at least 1.5.0_17. You can check this by checking which version of Java is used by default when you run the "java –version" command from the command line. This may be different from the JRE that ColdFusion uses.

Also, please note that as of the time of writing this document, Ant 1.8 is not yet supported and will report errors when Ant4CF build scripts are run. As of now, only Ant 1.7.* is supported, though it's our intent to enable compatibility with Ant 1.8.

Basic Usage

Ant4CF provides two custom Ant tasks that aid deploying ColdFusion applications. These tasks are:

remoteant – The remoteant task is used to execute a target within a local Ant build xml file on a remote server that has Ant4CF installed. This task can also automatically manage dependencies on third party Ant tasks.

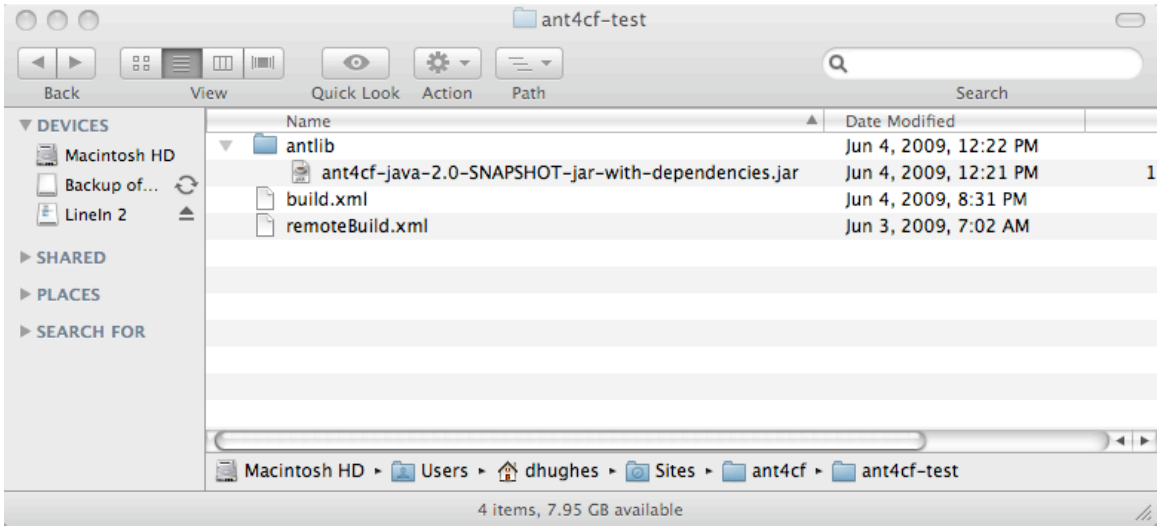
service – This is used to execute CFML code from a remotely-exposed server which is exposed by Ant4CF. You can easily write and use your own services, but Ant4CF

comes with proxies for the complete Admin API for ColdFusion 8.0.1 and scripts to generate these services for other versions of ColdFusion.

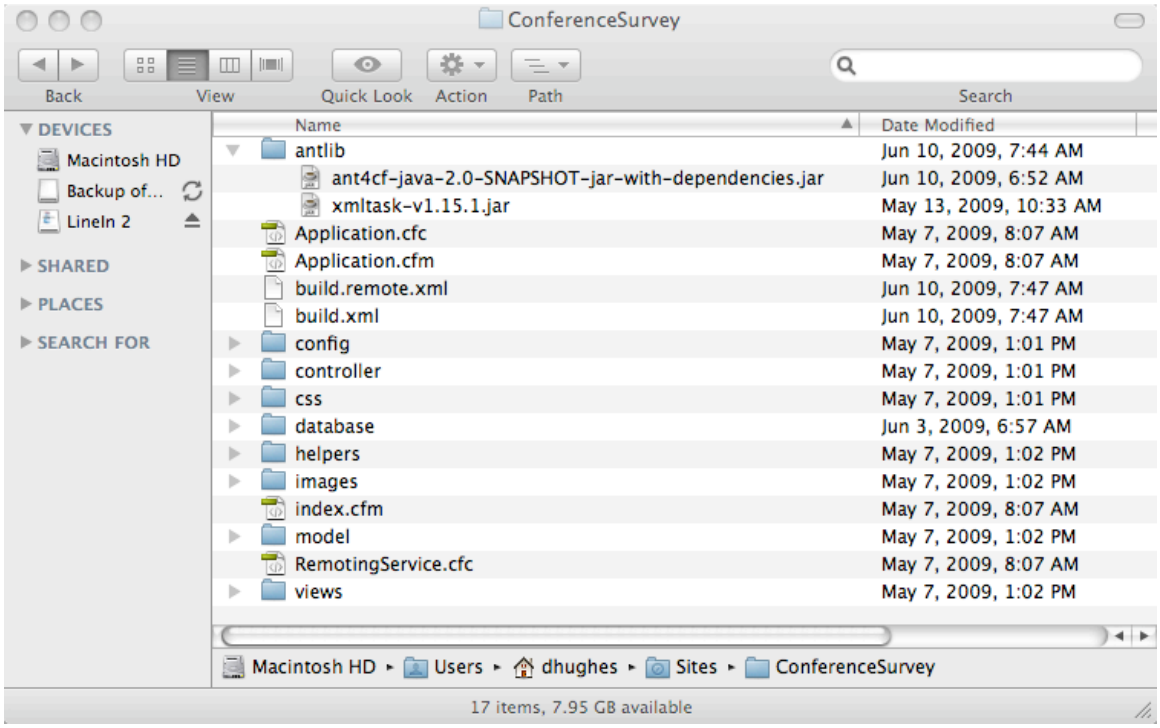
The most typical way to use Ant4CF is to create two build files. One would be a local build file (for example: build.xml) that is run locally and uses the remoteant task to execute a second build file (for example: remoteBuild.xml). Either the remoteBuild.xml file or the build.xml file can use the service task to execute build-specific CML code on your server.

Wherever these two ant build files are created you will also need to create an Ant4CF lib directory. By default this directory is called "antlib". The antlib directory is specific to Ant4CF and is where you will place the JAR files for all third party tasks including Ant4CF's jar file, ant4cf-java-2.0-SNAPSHOT-jar-with-dependencies.jar.

For example, the following screenshot shows a build.xml file, a remoteBuild.xml file and has, in the same directory, an antlib directory which holds the ant4cf-java-2.0-SNAPSHOT-jar-with-dependencies.jar



Theses files and folder would often be in the root directory of your application. For example, here's a screenshot from a small application:



You might also want to notice that there's an additional Ant task in this example for manipulating XML.

It's important to note that the build files do not need to be web accessible and could be put anywhere. The example above is just that, an example. Also, the antlib directory can be given any other name you want. See the remoteant task documentation for more information.

Here is a sample build.xml file that runs a remoteAnt.xml file:

```
<project name="Local Build Script" default="Build">

	<target name="Build">

		<input message="Please enter CFAdmin password:"
		addproperty="cfAdminPassword" />

		<input message="Please enter database user password:"
		addproperty="dsnPassword" />

		<input message="Please enter SVN username:"
		addproperty="svnUsername" />

		<input message="Please enter SVN password:"
		addproperty="svnPassword" />
```

```
        <taskdef name="configure"
        classname="com.alagad.ant4cf.Configure"
        classpath="antlib/ant4cf-java-2.0-SNAPSHOT-jar-with-
        dependencies.jar" />

        <taskdef name="remoteant"
        classname="com.alagad.ant4cf.RemoteAnt"
        classpath="antlib/ant4cf-java-2.0-SNAPSHOT-jar-with-
        dependencies.jar" />

        <configure ant4cfUrl="http://yourapp.com/CFIDE/ant4cf"
        adminPassword="${cfAdminPassword}" />

        <remoteant antfile="remoteBuild.xml"
        properties="cfAdminPassword,dsnPassword,svnUsername,svnPass
        word" timeout="120" />

    </target>

</project>
```

This file shows what might be in a basic build.xml file.

The first four input tags are used to gather input from the user who runs this script. The input task is a core tag of Ant and creates properties (really, variables) you can access later using the ${xyz} syntax.

Next we use the core taskdef task to define two Ant4CF tasks: configure and remoteant. This is required or the configure and remoteant tasks will not work. Note that the classpath needs to point the jar file under the antlib directory (or whatever directory you choose to use).

The configure task is used to tell Ant4CF where the Ant4CF folder is available and to set some basic configuration settings. The example above tells Ant4CF that the Ant4CF folder is available at http://yourapp.com/CFIDE/ant4cf. You will also need to provide at least the adminPassword attribute as well, which specifies the ColdFusion administrative password. If your server requires a username as well you can use the adminUserId argument.

The remoteant task is used to execute the remoteBuild.xml file on the server specified in the configure task. By default this task will execute the default target within the specified remoteBuild.xml file. A timeout is provided so that long running build scripts will not timeout and fail.

The remoteant task works by uploading the remoteBuild.xml file to the server specified in the configure task. It will also upload all of the JAR files in the antlib directory to the server if they're not already on the server. Ant4CF will take into

account the exact version of the JAR file and cache it on the server so that the file only needs to be uploaded once. Once the remoteBuild.xml file and all of the dependencies are uploaded to the server Ant4CF will execute the build script on the server and any configured action will be taken. The properties specified in the remoteant task will also be passed into the remote target.

The following is a remoteBuild.xml file:

```
<project name="Remote Build" default="BuildRemote">

	<target name="BuildRemote">
		<!-- any  tasks loaded just use classes from the antlib
		directory (even ant4cf) -->
		<taskdef name="configure"
		classname="com.alagad.ant4cf.Configure"
		classpath="antlib/ant4cf-java-2.0-SNAPSHOT-jar-with-
		dependencies.jar" />

		<taskdef name="service"
		classname="com.alagad.ant4cf.Service"
		classpath="antlib/ant4cf-java-2.0-SNAPSHOT-jar-with-
		dependencies.jar" />

		<taskdef name="xmltask"
		classname="com.oopsconsultancy.xmltask.ant.XmlTask"
		classpath="antlib/xmltask-v1.15.1.jar" />

		<property name="webroot" value="C:\webs\yourapp.com\www" />

		<configure ant4cfUrl="http://yourapp.com/CFIDE/ant4cf"
		adminPassword="${cfAdminPassword}" />

		<!-- get the app -->
		<echo message="Export App out of SVN" />
		<exec executable="svn">
			<arg line="export
			http://svn.yourdomain.com/YourApp/trunk ${webroot} --
			username ${svnUsername} --password ${svnPassword} --
			force" />
		</exec>

		<!-- update the configuration -->
		<echo message="Update the configuration with the correct
		admin password" />
		<xmltask source="${webroot}/config/ColdSpring.xml"
		dest="${webroot}/config/ColdSpring.xml">
			<replace
			path="//property[@name='reload']/value/text()"
```

```
      withText="false"/>
      <replace
      path="//property[@name='debug']/value/text()"
      withText="false"/>
</xmltask>

<!-- get model-glue -->
<echo message="Export Model-Glue 3 out of SVN" />
<exec executable="svn">
      <arg line="export http://svn.model-
      glue.com/trunk/ModelGlue ${webroot}\ModelGlue --
      force" />
</exec>

<!-- get coldspring -->
<echo message="Get ColdSpring from CVS" />
<cvspass
cvsroot=":pserver:anoncvs@cvs.coldspringframework.org:/cold
spring/" password="anoncvs" />
<cvs
cvsRoot=":pserver:anoncvs@cvs.coldspringframework.org:/cold
spring" package="coldspring" dest="${webroot}" />

<!-- create the dsn -->
<echo message="create the dsn" />
<!-- create the DSN if necessary -->
<service component="adminapi.801.datasourceProxy"
method="setMSSQL" property="result">
      <argument name="name" value="yourapp" />
      <argument name="host" value="10.248.79.47" />
      <argument name="database" value="yourdatabase" />
      <argument name="username" value="dbusername" />
      <argument name="password" value="${dsnPassword}" />

      <argument name="selectmethod" value="direct" />
</service>
<echo message="result: ${result}" />

<!-- clear the trusted cache -->
<echo message="clear the trusted cache" />
<service component="adminapi.801.runtimeProxy"
method="clearTrustedCache" property="result" />
<echo message="result: ${result}" />

<!-- turn on trusted cache -->
<echo message="turn on trusted cache" />
<service component="adminapi.801.runtimeProxy"
method="setCacheProperty" property="result">
```

```
                <argument name="propertyName" value="TrustedCache" />
                <argument name="propertyValue" value="true" />
            </service>
            <echo message="result: ${result}" />

            <!-- reload the application -->
            <echo message="reload the application" />
            <get src=http://survey.alagad.com?init=true
            dest="init.txt"/>
            <delete file="init.txt" />
        </target>

</project>
```

This remoteBuild.xml file is where the magic really happens. This file is run remotely on the target server by the remoteant task. Let's review this file and see what we're doing. Keep in mind that you can do pretty much anything in this file that you can with Ant. Ant4CF attempts to handle all of the dependencies required to run the script.

This script starts out by defining a few tasks. The first two tasks are Ant4CF tasks: configure and service. We've already discussed configure. The service task is used to execute CFML code in the form of a remotely accessible service. We'll discuss this more shortly.

The third task defined is a third party custom task for working with XML. To use this task you would need to have the xmltask-v1.15.1.jar file in your antlib directory. When remoteant is run this file would be synced with the server and then used by the server.

The core property tag is used to create a variable that holds the path to the webroot where the application will be built.

Next, the configure task is used just like it was in the build.xml file. Because Ant4CF is a collection of Ant tasks and Ant runs under Java, outside the context of ColdFusion all communication from Ant to ColdFusion happens over HTTP. For this reason we do still need to use the configure task to tell the Ant4CF tasks what serer to talk to and the password to the server.

Following the configure task, the script exports the application out of Subverion. This uses core ant tasks to execute the svn command.

Then the build script uses the custom xmltask to update the application's configuration for a production environment. In this case this is a Model-Glue application and we're setting the application no to reload and to not show debugging output.

If this were any other application you could make whatever configuration changes you required by editing files, replacing text or anything else.

After configuration changes this script downloads the latest and greatest Model-Glue and ColdSpring source files and place them in the application's webroot. (We avoid mappings at Alagad due to conflicts between versions of dependencies like Model-Glue.) This script is just for an example. You may not always want to do checkout the latest code from SVN. You'll need to use your own judgment in your own scripts.

The last section of the remoteBuild.xml file uses the service task and the adminapi service to make configuration changes to the ColdFusion server itself. In the Ant4CF folder there is a services directory. Any CFC under this directory can be accessed in dot notated format. So, for example, look at the folowing this service task:

```
<service component="adminapi.801.datasourceProxy" method="setMSSQL"
property="result">
      <argument name="name" value="yourapp" />
      <argument name="host" value="10.248.79.47" />
      <argument name="database" value="yourdatabase" />
      <argument name="username" value="dbusername" />
      <argument name="password" value="${dsnPassword}" />
      <argument name="selectmethod" value="direct" />
</service>
```

This is very similar to the cfinvoke tag in ColdFusion. The component argument specifies which Ant4CF service to use, in this case, "adminapi.801.datasourceProxy". The method argument specifies the method "setMSSQL" will be run. The result, if any, will be returned into the result property. The various arguments to this function are provided using the argument tag which is a child tag to the service task and does not need to be loaded using a taskdef task.

The example above creates (or updates) a MSSQL datasource using the arguments provided.

The remoteBuild.xml script wraps up by enabling then clearing the trusted cache via the service task and reloading the application using the core get task.

To run this script you would, on your local machine, browse to the directory which contains your build.xml file and run this command (assuming Ant is on your path): ant.

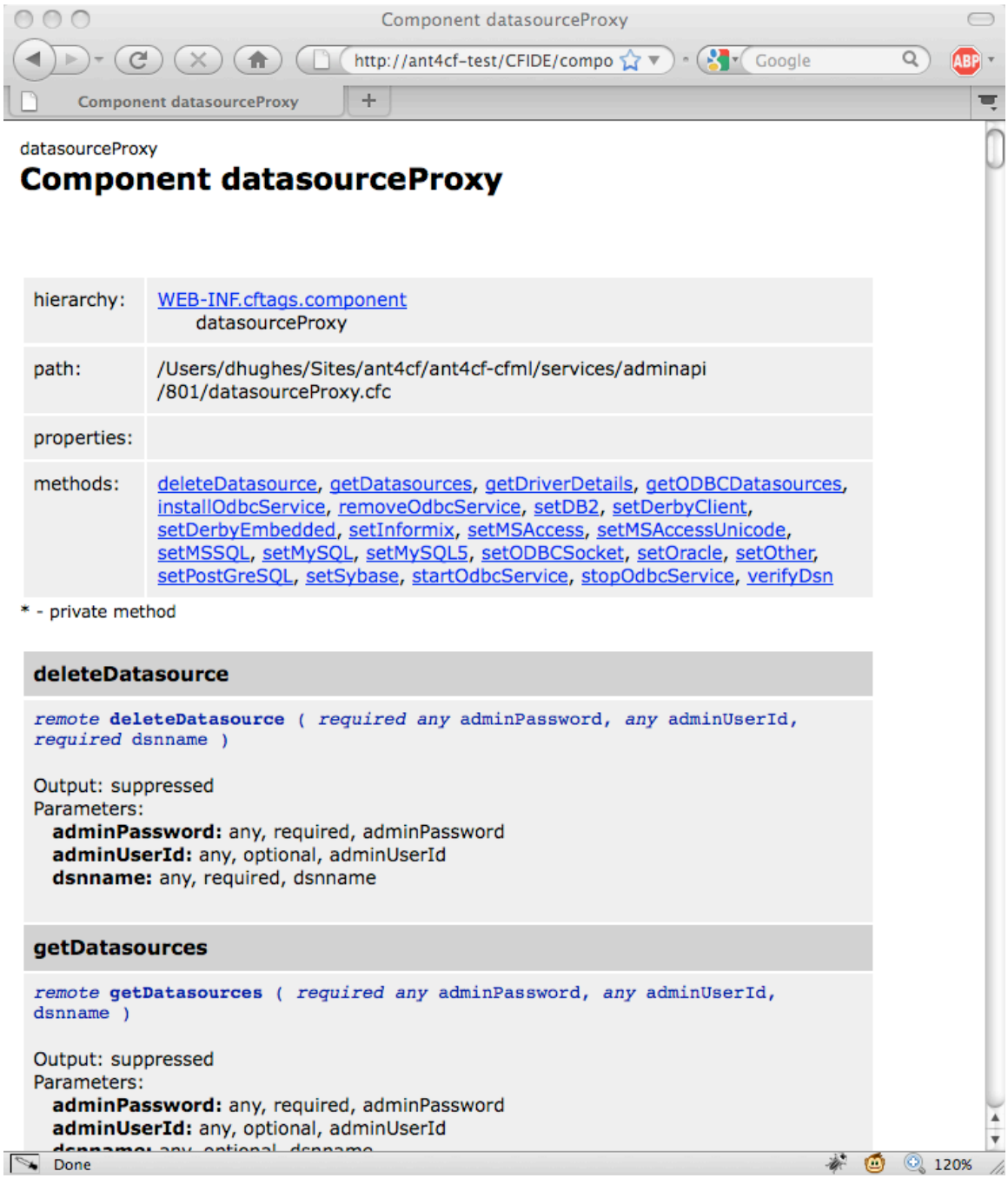
This will collect any input your requested, run the script and show generated output.

Using Services

You may be wondering how to know what is available via the adminapi service or what to do in the case that you're using a version of CF different than 8.0.1 or how to create your own service. This is all very simple. We'll address these things one at a time.

Browsing Services

Because Ant4CF's services are exposed via HTTP you can browse services by setting the services directory under Ant4CF to enable directory browsing in your Web Server. This will let you step through the directory structure and easily browse directly to a CFC. For example, you could browse to the adminapi/801/datasourceProxy.cfc file. This will prompt you to enter your CFIDE password and open generated documentation for this component.



This documentation will tell you each method and all the arguments and their explanation for the admin API proxy. This will also work for any custom services you create or download.

Generating Admin API Services

By default Ant4CF comes with a proxy for the ColdFusion 8.0.1 Admin API. If you are running an older or newer version of ColdFusion then the 8.0.1 Admin API proxy may not be what you need.

Luckily, Ant4CF comes with a simple script you can use to generate a set of Proxy components. To use this script you simply need to browse to the "support/generate/generateAdminApiService.cfm" file under Ant4CF. For example: http://yourdomain.com/CFIDE/ant4cf/support/generate/generateAdminApiService.cfm. Assuming you're running on ColdFusion 7.0.0 this would generate the admin API proxy in the services/adminapi/700 folder.

Creating Your Own Services

You can also create your own services to do whatever you want. For example, perhaps your organization has a hardware load balancer. This load balancer may have an API that you can somehow use from ColdFusion. It would be relatively simple to create a CFC that provided methods to interact with this load balancer.

For the sake of this example, let's pretend you'd be creating a service that could take one server out of the load balancer cluster and add a server into the cluster. For this contrived example you might have a CFC named "loadbalancer.cfc" with these two functions:

removeServer(name) – removes a server from the cluster
addServer(name) – adds a server to the cluster.

You can place this CFC anywhere under the services directory that you want. For this example we'll say it was put under a folder named "loadbalancer".

To use this service to remove a server from the cluster you would write some XML which looks like this:

```
<service component="adminapi.loadbalancer.loadbalancer"
method="removeServer" property="result">
      <argument name="propertyName" value="Web1" />
</service>
```

To add the server back into the cluster you would write this XM:

```
<service component="adminapi.loadbalancer.loadbalancer"
method="addServer" property="result">
      <argument name="propertyName" value="Web1" />
</service>
```

Ultimately, services can be written to do just about anything you would want them to.

Ant4CF Tasks

configure Task

The configure task is used to configure the basic settings used by Ant4CF.

Attributes

Attribute	Description	Required
adminPassword	The password to the ColdFusion admin interface.	Yes
adminUserId	The username to the ColdFusion admin interface. This is only used if your administrator has enabled this.	No
ant4cfUrl	This is the URL where the Ant4CF tasks can find the Ant4CF installation on the remote server.	Yes
debug	Turns on debugging output.	No. Defaults to False.

Example

```
<configure ant4cfUrl="http://yourapp.com/CFIDE/ant4cf"
adminPassword="${cfAdminPassword}" />
```

remoteant Task

The remoteant task is used to execute a target within a local Ant build xml file on a remote server that has Ant4CF installed. This task also automatically manages dependencies on third party Ant tasks.

Attributes

Attribute	Description	Required
antfile	This is relative path to the Ant file to remotely execute	Yes
libdir	This is the relative path to a directory containing the Ant4CF jar file as well as any other JARs for third party tasks.	No. Defaults to “antlib”
properties	This is a comma separated list of properties to pass to the remotely executed Ant script.	No
target	This is optionally used to specify the target to run in the remote Ant file. This overrides whatever default may be set in this file.	No
timeout	This specifies the number of seconds an remote ant script can run before it times	No. Defaults

	out.	to 60 seconds.

Example

```
<remoteant antfile="remoteBuild.xml"
properties="cfAdminPassword,dsnPassword,svnUsername,svnPassword"
timeout="120" />
```

service Task

The service task used to run methods within CFCs that have been exposed under the Ant4CF services directory. For more information see Using Services. Arguments can be passed into the service using the child service task.

Attributes

Attribute	Description	Required
component	The path to the service component to execute. This component is assumed to be under the "services" directory under the Ant4CF installation directory.	Yes
method	This is the name of the method to execute in the service component	Yes
property	This is the property into which results (if any) from the service are written.	No

Argument Tag Attributes

Attribute	Description	Required
name	The name the argument on the service	Yes
value	The value for the argument on the service	Yes

Example

```
<service component="adminapi.801.datasourceProxy" method="setMSSQL"
property="result">
      <argument name="name" value="yourapp" />
      <argument name="host" value="10.248.79.47" />
      <argument name="database" value="yourdatabase" />
      <argument name="username" value="dbusername" />
      <argument name="password" value="${dsnPassword}" />
      <argument name="selectmethod" value="direct" />
</service>
```