Configuring your CommandBox servers on first start (https://www.ortussolutions.com/blog/configuring-your-commandbox-servers-on-first-start)


Brad Wood

Dec 13, 2016

More and more people are starting to use CommandBox for their local development, especially among teams who want to quickly and easily start up the same environment on each of their machines. That has led to the most common question now for CommandBox users which is:

> How do I automatically configure my ColdFusion/Lucee settings on my server?

This is a fair question, and for the most part I've treated configuring the settings in your CF engine to be outside the realm of what CommandBox tries to solve. However the reality is, the CF engines often times fall short and people need a better way. Here's a guide for your current options when it comes to configuring the settings on your CF engine. These are listed in no particular order.

Update: This post is pretty old and much of the information here is superseded by a new project called CFConfig (https://cfconfig.ortusbooks.com/introduction/getting-started-guide). Please check it out here:

https://cfconfig.ortusbooks.com/introduction/getting-started-guide (https://cfconfig.ortusbooks.com/introduction/getting-started-guide)

The Manual way

Each CommandBox server is a "full" CF server with access to the web-based administrator to add mappings, datasources, mail servers, and any other setting you want. Right click on the tray icon for your server and choose the option to open the server admin. For Lucee engines, you'll have an option to open the web admin as well. Of course, using the web interface is a manual process and not something you can easily deploy to your entire team of developers without them each needing to take those manual steps.

Application.cfc

This is my favorite and recommended approach. It's automatic, keeps your configuration with your codebase, and requires no manual process. The biggest issue with configuring your app server's settings in Application.cfc is that Adobe ColdFusion doesn't provide nearly the same amount of configuration as Lucee, and older versions of Adobe CF don't allow any at all. Lucee isn't 100% either. For instance, mail servers and cache connections just got added there in recent releases. Here's a quick run down:

- **Adobe CF 9** - CF mappings only

- **Adobe CF 10** - CF mappings only
- **Adobe CF 11** - Adds support for data sources (link (http://blogs.coldfusion.com/post.cfm/application-datasources-in-coldfusion))
- **Adobe CF 2016** - A few small additions (http://cfdocs.org/application-cfc)
- **Lucee 4.x** - Many non-compilation server settings (See setting export page in admin)
- **Lucee 5.x** - *Most* non-compilation server settings (See setting export page in admin)

So unless you're on Lucee 5, you're going to be limited. I'm not aware of a comprehensive overview of Adobe settings available other than this (http://cfdocs.org/application-cfc), but here is a Google doc (https://docs.google.com/spreadsheets/d/10s-nn_FsoSD_RiLwjYZICacCoC386SjkEGT3pOfBJVU/edit?usp=sharing) that shows an overview of all the possible Lucee 5 configurations. Check the **Application.cfc** column.

In the Lucee admin, you can click the little question mark under any setting and it will show you what to copy paste into your **Application.cfc** to set the equivalent setting in your code. There's also an option to export all your Lucee config at once.

For an example of an Application-specific datasource in Adobe CF11+, click here (https://github.com/foundeo/cfml-security-training/blob/master/wwwroot/Application.cfc#L15-L22).

Environment variables/Java properties

This is a Lucee 5-only option, not supported on Adobe CF or Lucee 4.x servers at all. A select number of Lucee settings can be specified as environment variables or Java system properties prior to starting the server. These are ideal for compilation settings which can't be specified in code. Here is a Google doc (https://docs.google.com/spreadsheets/d/10s-nn_FsoSD_RiLwjYZICacCoC386SjkEGT3pOfBJVU/edit?usp=sharing) that shows an overview of all the possible Lucee configurations. Check the **EnvVar/SysProp** column. Here are a few big ones:

- **lucee.full.null.support** (boolean)
- **lucee.web.dir** (path)
- **lucee.base.dir** (path)
- **lucee.enable.dialect** (boolean)

To use these on a CommandBox server, you can set them as environment variables in your operating system, or package them as Java system properties in your server's **server.json** like so:

```
{
  "jvm":{
    "args":"-Dlucee.enable.dialect=true"
  }
}
```

Basically, prefix the property name with "**-D**". This is the standard Java way to set system properties via JVM args. When the server starts, those Java system properties will be defined and picked up by the engine.

Create a custom CF engine

This is probably the most work, but allows you to customize every nook and cranny of your CF engine. You can save your own custom WAR file and use it to start your servers. When you use the **cfengine** parameter to the **server start** command, you're probably used to pointing it to the ForgeBox slugs maintained by Ortus (https://www.forgebox.io/type/cf-engines). The **cfengine** parameter can be any valid CommandBox endpoint ID though including a local file/folder, Git repo, HTTP(s) URL, or custom ForgeBox entry. So let's say you create a custom ColdFusion 11 WAR that comes pre-loaded with all the settings your app needs and you place it on a shared network drive or web site for your fellow developers. Just put this in the server.json:

```
{
  "cfengine" : "/local/path/to/engine.zip"
  or...
  "cfengine" : "http://www.mysite.com/engine.zip"
}
```

As documented here (https://ortus.gitbooks.io/commandbox-documentation/content/embedded_server/multi-engine_support.html), the package zip file needs to contain the following two things:

1. **box.json**
2. **Engine.[zip|war]** (file name doesn't matter)

The easiest way to do this is to start your server with one of our Ortus default engines, log into the administrator and make all your changes, and stop the server and navigate to the server's home directory by running this:

```
CommandBox> server info property=serverHomeDirectory | open
```

Zip up the contents of the folder that opens and optionally rename the zip file to have a **.WAR** extension. The **WEB-INF** folder should be in the root of your zip file. Then package up that new archive in a new zip long with a box.json that minimally has a **version**, **type** (cf-engines), and **slug**.

Copy Configs on first start

This is probably the best one as it is very flexible and will work on any CF engine regardless of vendor or version. Please note, this requires you to have at least **CommandBox 3.4.1-snapshot** installed, which is in pre-release at the time of this blog post. You can grab it here (http://integration.stg.ortussolutions.com/artifacts/ortussolutions /commandbox/3.4.1-snapshot/). In this version of CommandBox, all CF engines have been standardized to expand their WARs to the same consistent directory structure. We've also enhanced the **onServerInstall** package script (https://ortus.gitbooks.io/commandbox-documentation/content/developing/interceptors /interceptor_based_cli_scripts.html) to have access to the server home folder like we did above through the **server info** command. **onServerInstall** will only fire the first time a server is started and the WAR gets installed. You'll need to **stop**, **server forget** and then **start** again for the event to fire again. If you want to overwrite the configs every time, use the **onServerStart** event.

Basically, we'll just copy the XML config files for the server after we've installed the WAR, but before the server actually boots up. The only drawbacks of this are that the config files differ per engine and per engine version. If you're starting several different CF engines

/versions in the same web root, you'll have some issues with the package script approach since it doesn't have access to the server name being started.

- For Adobe CF WARs, the xml config files are located in the WAR here: **/WEB-INF/cfusion/lib/neo.*.xml**
- For the Lucee **server** context, the xml config file is located in the WAR here: **/WEB-INF/lucee-server/context/lucee-server.xml**
- For the Lucee **web** context, the xml config file is located in the WAR here: /**WEB-INF/lucee-web/lucee-web.xml.cfm**

An Adobe CF **box.json** might look like so. This will copy my datasource XML file from my web root into the WAR the first time I start up the site.

```
{
  "name":"My app",
  "version":"1.0.0",
  ...
  "scripts":{
    onServerInstall: "cp neo-datasource.xml '`server info property=serverHomeDirectory`/W
  }
}
```

A Lucee Server **box.json** might look like so.

```
{
  "name":"My app",
  "version":"1.0.0",
  ...
  "scripts":{
    onServerInstall: "cp lucee-server.xml '`server info property=serverHomeDirectory`/WEB
  }
}
```

Pay attention to your quotes. My JSON uses double quotes, and the second parameter to the **cp** command is wrapped in single quotes, and contains a CommandBox expression (https://ortus.gitbooks.io/commandbox-documentation/content/usage/parameters/expressions.html) wrapped in back ticks. It's also worth noting the server-related package scripts run with their current working directory set to the web root of the server, so any relative paths will be relative to the web root. The easiest way to get the config files is to make your desired changes via the web-based admin, then stop the server, open the server home, and copy the file.

I would recommend keeping config files outside the web root so you don't deploy them on accident. In that case you'd reference them with something like "**../config.xml"**. Read more about package scripts here (https://ortus.gitbooks.io/commandbox-documentation/content/developing/interceptors/interceptor_based_cli_scripts.html).

Conclusion

Hopefully this guide has given you some ideas on how to better package up your servers. If you want to learn more about server.json in general, check out our docs here (https://ortus.gitbooks.io/commandbox-documentation/content/embedded_server/serverJSON/serverjson.html). Please use these examples a starting point and remember you can get even more funky by creating a CommandBox module (https://ortus.gitbooks.io/commandbox-documentation/content/developing/modules/developing_modules.html) that listens to the **onServerInstall** or **onServerStart** interception points. They will have access to much more data than the package scripts do.

www.ortussolutions.com (http://www.ortussolutions.com)