
GRAILS


spring
source

Cloud Foundry Plugin - Reference Documentation

Authors: Burt Beckwith

Version: 1.0

Table of Contents

1 Introduction to the Cloud Foundry Plugin

The Cloud Foundry plugin makes it easy to deploy, update and monitor Grails applications in Cloud Foundry. It has the same feature set as the "vmc" commandline Ruby client that Rails and Node.js users will be using. There's also support in SpringSource Tool Suite (STS) via a plugin.
The benefit of using this plugin versus the vmc client or STS is that the plugin is very Grails-aware. It makes several assumptions that reduce the amount of typing you need to do, for example you don't need to type the application name since it defaults to your Grails application name (although you can configure the deployed name in Config.groovy, or override the application name with the `--appname` commandline argument).

Use the interactive script when you're running multiple Cloud Foundry scripts - it will significantly reduce the startup time for each command you run once the initial JVM has started.

1.1 History

History

- July 13, 2011
  - 1.0 release
- April 29, 2011
  - 1.0.0.M3 release
  - Fixed support for RabbitMQ
- April 21, 2011
  - 1.0.0.M2 release
  - Added cf-env, cf-env-add, cf-del, cf-frameworks, and cf-runtimes scripts
- April 12, 2011
  - 1.0.0.M1 release

2 Getting Started

The first step is to install the plugin:

```
grails install-plugin cloud-foundry
```

Add your Cloud Foundry credentials to `grails-app/conf/Config.groovy`, or an external config file referenced from `Config.groovy`, or in `$HOME/.grails/settings.groovy`:

```
grails.plugin.cloudfoundry.username = 'your.email@server.com'
grails.plugin.cloudfoundry.password = 's3kr3t'
```

If you want to deploy with a different application name than your Grails application name, specify it with the `grails.plugin.cloudfoundry.appname` property:

```
grails.plugin.cloudfoundry.appname = 'my-cool-app'
```

There are other configuration options, but next you'll want to provision some services and deploy your application. See section 4 Deploying applications for details on that.

3 Configuration

There are a few configuration options for the plugin. All are specified in Config.groovy, but you can put the values for `grails.plugin.cloudfoundry.username` and/or `grails.plugin.cloudfoundry.password` in `$HOME/.grails/settings.groovy` to avoid storing this information in source control.

Property	Default	Meaning
grails.plugin.cloudfoundry.username	none, must be specified	the userna
grails.plugin.cloudfoundry.password	none, must be specified	the passw
grails.plugin.cloudfoundry.target	`'api.cloudfoundry.com'`	the cloud
grails.plugin.cloudfoundry.appname	application name	the applic applicatio overridde attribute
grails.plugin.cloudfoundry.showStackTrace	`false`	if `true` s
grails.plugin.cloudfoundry.testStartWithGet	`true`	whether t to checkir positives
grails.plugin.cloudfoundry.testStartGetUrl	application url	the url to `grails.` is `true`
grails.plugin.cloudfoundry.datasource.disableTimeoutAutoconfiguration	`false`	disables a timeout cl

4 Deploying applications

Initial state

When you initially start you won't have any configured services or applications. Running the cf-services script should display something similar to this:

```
$ grails cf-services
============== System Services ==============
+----------+---------+------------------------------+
| Service  | Version | Description                  |
+----------+---------+------------------------------+
| mongodb  | 1.8     | MongoDB NoSQL store          |
| redis    | 2.2     | Redis key-value store service |
| mysql    | 5.1     | MySQL database service       |
+----------+---------+------------------------------+
============ Provisioned Services ============
None provisioned yet.
```

All of the available services are listed first, and then your configured service instances, if any.
If you run the cf-apps script you should see that there are no applications:

```
No applications available.
```

And if you run the cf-info script you should see that there are no resources in use:

```
VMware's Cloud Application Platform
For support visit support@cloudfoundry.com
Target:   http://api.cloudfoundry.com (v0.999)
User:     your.email@yourserver.com
Usage:    Memory   (0B of 2.0G total)
          Services (0 of 16 total)
          Apps     (0 of 20 total)
```

Services

Before deploying your application, you'll probably need to configure at least one service, e.g. a MySQL, MongoDB, or Redis database. Use the cf-create-service script for that, e.g.

```
$ grails cf-create-service mysql
Service 'mysql-2f5fb76' provisioned.
```

or

```
$ grails cf-create-service redis
Service 'redis-364841f' provisioned.
```

By default a name is generated for you, but you can choose the name, and also bind it to an existing application. You can also bind a service to an application when you deploy the application.
Re-running the cf-services script will display the new services:

```
============== System Services ==============
+----------+---------+------------------------------+
| Service  | Version | Description                  |
+----------+---------+------------------------------+
| mongodb  | 1.8     | MongoDB NoSQL store          |
| redis    | 2.2     | Redis key-value store service |
| mysql    | 5.1     | MySQL database service       |
+----------+---------+------------------------------+
========== Provisioned Services ============
+-----------------+---------+
| Name            | Service |
+-----------------+---------+
| mysql-2f5fb76   | mysql   |
| redis-364841f   | redis   |
+-----------------+---------+
```

Use the cf-delete-service script to delete a service:

```
$ grails cf-delete-service redis-364841f
Successfully removed service: redis-364841f
```

Applications

The cf-push script creates an application. It deploys a war file and optionally binds one or more services, and has options to configure the allocated memory and associated URLs. By default it also starts the application but you can deploy it without starting if you want to perform configuration steps after creating the application.
The default memory allocation is 512MB but this can be overridden when running the script or changed later.
You can use an existing war file but if one isn't specified then a war is built for you.

Most scripts are not environment-specific but cf-push builds a war file, so be sure to specify the environment.

A typical deploy command would be

```
$ grails prod cf-push --services=mysql-2f5fb76
```

or if you're running the interactive script:

```
prod cf-push --services=mysql-2f5fb76
```

and you should see output similar to

```
Environment set to production
Building war file
     [gspc] Compiling 13 GSP files for package …
…
Building WAR file …
     [copy] Copying 633 files to …
…
[propertyfile] Updating property file: …
…
      [jar] Building jar: .../target/cf-temp-1299173015209.war
…
Done creating WAR ...target/cf-temp-1299173015209.war
Application Deployed URL: 'myappname.cloudfoundry.com'?
Uploading Application Information.
Trying to start Application: 'myappname'.
..................
Application 'myappname' started.
```

Running cf-apps should display something like this:

```
+-------------+----+---------+---------------------------+---------------+
| Application | #  | Health  | URLs                      | Services      |
+-------------+----+---------+---------------------------+---------------+
| myappname   | 1  | RUNNING | myappname.cloudfoundry.com | mysql-2f5fb76 |
+-------------+----+---------+---------------------------+---------------+
```

You might also omit the service bindings and deploy without starting:

```
grails prod cf-push --no-start
```

or use a pre-built war:

```
grails prod cf-push --warfile=target/myappname.war
```

Note that the war file name has no bearing on the deployed url - the war is deployed as the default ('ROOT') context in Tomcat.

Automatic binding and provisioning
You can also let the plugin discover required services for you. The plugin will determine that you probably need a MySQL database if you have the Hibernate plugin, a MongoDB instance if you have the Mongo plugin, and a Redis instance if you have that plugin.
If you specify a service of the correct type (with the `--services` attribute) then it'll be used, otherwise if you need one and have one provisioned it will be suggested to you, e.g.

```
Would you like to bind the 'mysql-2f5fb76' service? ([y], n)
```

and if you need one and don't have one provisioned, the plugin will offer to provision and bind one for you, e.g.

```
Would you like to create and bind a redis service? ([y], n)
```

5 Updating applications and services

Updating applications

Once an application has been created, there are several options for making changes to it:

- binding and unbinding services
- mapping and unmapping URLs
- increasing or decreasing the number of running instances
- increasing or decreasing the amount of memory allocated

Services

Use the cf-bind-service script to associate an existing service instance with your application:

```
grails cf-bind-service mongodb-eaa5601
```

or if you're running the interactive script:

```
cf-bind-service mongodb-eaa5601
```

```
Creating new service binding to 'mongodb-eaa5601' for 'myappname'.
Application 'myappname' updated
Service 'mongodb-eaa5601' added
Application 'myappname' stopped.
Trying to start Application: 'myappname'.
.
Application 'myappname' started.
```

If you run cf-apps you'll see that the service is registered:

```
+-------------+----+---------+----------------------------+---------------------------------+
| Application | #  | Health  | URLs                       | Services                        |
+-------------+----+---------+----------------------------+---------------------------------+
| myappname   | 1  | RUNNING | myappname.cloudfoundry.com | mysql-2f5fb76, mongodb-eaa5601 |
+-------------+----+---------+----------------------------+---------------------------------+
```

Use the cf-unbind-service script to remove an existing service instance from your application:

```
$ grails cf-unbind-service mongodb-eaa5601
Removing service binding 'mongodb-eaa5601' from 'myappname'.
Application 'myappname' updated
Service 'serviceName' removed
Application 'myappname' stopped.
Trying to start Application: 'myappname'.
.
Application 'myappname' started.
```

If you run cf-apps you'll see that the service is no longer registered:

```
+-------------+----+---------+----------------------------+---------------+
| Application | #  | Health  | URLs                       | Service       |
+-------------+----+---------+----------------------------+---------------+
| myappname   | 1  | RUNNING | myappname.cloudfoundry.com | mysql-2f5fb76 |
+-------------+----+---------+----------------------------+---------------+
```

URLs
Use the cf-map script to associate an additional URL with your application:

```
$ grails cf-map myotherappname.cloudfoundry.com
Succesfully mapped url
```

If you run cf-apps you'll see that the extra URL is mapped:

```
+-------------+----+---------+-----------------------------------------------------------------+
| Application | #  | Health  | URLs                                                            |
+-------------+----+---------+-----------------------------------------------------------------+
| myappname   | 1  | RUNNING | myappname.cloudfoundry.com, myotherappname.cloudfoundry.com |
+-------------+----+---------+-----------------------------------------------------------------+
```

Use the cf-unmap script to remove a URL from your application:

```
$ grails cf-unmap myotherappname.cloudfoundry.com
Succesfully unmapped url
```

If you run cf-apps you'll see that the URL is no longer mapped:

```
+-------------+----+---------+---------------------------+---------------+
| Application | #  | Health  | URLs                      | Service       |
+-------------+----+---------+---------------------------+---------------+
| myappname   | 1  | RUNNING | myappname.cloudfoundry.com | mysql-2f5fb76 |
+-------------+----+---------+---------------------------+---------------+
```

Instances
Use the cf-update-instances script to increase or decrease the number of running instances of your application:

```
$ grails cf-update-instances 3
Scaled 'myappname' up to 3 instance(s).
```

It will take a short amount of time to start the new instances, so if you run cf-apps you'll see that the extra instances are allocated but not all are running yet:

```
+-------------+----+---------+---------------------------+---------------+
| Application | #  | Health  | URLs                      | Service       |
+-------------+----+---------+---------------------------+---------------+
| myappname   | 3  | 33%     | myappname.cloudfoundry.com | mysql-2f5fb76 |
+-------------+----+---------+---------------------------+---------------+
```

Wait a while longer and all should be running:

```
+-------------+----+---------+---------------------------+---------------+
| Application | #  | Health  | URLs                      | Service       |
+-------------+----+---------+---------------------------+---------------+
| myappname   | 3  | RUNNING | myappname.cloudfoundry.com | mysql-2f5fb76 |
+-------------+----+---------+---------------------------+---------------+
```

If you have a test page that displays the environment variables VCAP_APP_HOST and VCAP_APP_PORT you'll

see that you're automatically load-balanced randomly between the instances when you refresh your browser:

```
VCAP_APP_HOST     10.114.110.207
VCAP_APP_PORT     48684
```

```
VCAP_APP_HOST     10.114.110.208
VCAP_APP_PORT     47793
```

```
VCAP_APP_HOST     10.114.110.209
VCAP_APP_PORT     58232
```

Use the cf-show-instances script to display information about the instances:

```
$ grails cf-show-instances
+-------+---------+--------------------+
| Index | State   | Start Time         |
+-------+---------+--------------------+
| 0     | RUNNING | 04/12/2011 11:33AM |
+-------+---------+--------------------+
| 1     | RUNNING | 04/12/2011 11:33AM |
+-------+---------+--------------------+
| 2     | RUNNING | 04/12/2011 11:33AM |
+-------+---------+--------------------+
```

Use the cf-stats script to display runtime information about the instances:

```
$ grails cf-stats
+----------+-------------+----------------+--------------+--------------+
| Instance | CPU (Cores) | Memory (limit) | Disk (limit) | Uptime       |
+----------+-------------+----------------+--------------+--------------+
| 0        | 9.5% (4)    | 372.7M (512M)  | 52.6M (2G)   | 0d:0h:1m:12s |
| 1        | 3.3% (4)    | 375.7M (512M)  | 52.6M (2G)   | 0d:0h:1m:53s |
| 2        | 3.7% (4)    | 383.2M (512M)  | 52.6M (2G)   | 0d:0h:1m:55s |
+----------+-------------+----------------+--------------+--------------+
```

You can also scale the number down:

```
$ grails cf-update-instances 1
Scaled 'myappname' down to 1 instance(s).
```

Memory
Use the cf-update-memory script to increase or decrease the allocated memory for your application:

```
$ grails cf-update-memory 1G
Updated memory reservation to '1.0G'.
Application 'myappname' stopped.
Trying to start Application: 'myappname'.
.
Application 'myappname' started.
```

Note that the memory must be one of 64M, 128M, 256M, 512M, 1G, or 2G.

Start, stop, restart

Use the cf-stop script to stop your application:

```
$ grails cf-stop
Application 'myappname' stopped.
```

Use the cf-start script to start your application:

```
$ grails cf-start
Trying to start Application: 'myappname'.
.
Application 'myappname' started.
```

Use the cf-restart script to stop and then start your application:

```
$ grails cf-restart
Application 'myappname' stopped.
Trying to start Application: 'myappname'.
.
Application 'myappname' started.
```

Updating
Use the cf-update script to redeploy an application after making code changes:

```
$ grails prod cf-update
Uploading Application.
Building war file
     [gspc] Compiling 13 GSP files for package …
…
Building WAR file …
     [copy] Copying 635 files to …
…
[propertyfile] Updating property file: …
…
      [jar] Building jar: .../target/cf-temp-1299178697861.war
…
Done creating WAR ...target/cf-temp-1299178697861.war
Application 'myappname' stopped.
Trying to start Application: 'myappname'.
.
Application 'myappname' started.
```

Most scripts are not environment-specific but cf-update builds a war file, so be sure to specify the environment.

Like the cf-push script you can either let the script create a war file for you, or specify an existing one.

Deleting applications
Use the cf-delete-app script to delete an application:

```
$ grails cf-delete-app
Application 'myappname' deleted.
```

Use the cf-delete-all-apps script to delete all applications:

```
$ grails cf-delete-all-apps
Application 'myappname1' deleted.
Application 'myappname2' deleted.
Application 'myappname3' deleted.
```

6 Monitoring

Crashes
If a deployment or update fails, the script will display the application logs but you can view crash information and logs at any time.
Use the cf-crashes script to check if there have been recent crashes:

```
$ grails cf-crashes
+-------------+--------------------------------------+--------------------+
| Name        | Instance ID                          | Crashed Time       |
+-------------+--------------------------------------+--------------------+
| myappname-1 | 9f3e61c9-7c64-4e57-ab3e-8d7e892dd722 | 04/11/2011 02:43PM |
| myappname-2 | 17b6470c-6444-49f4-8981-d3bd7d7796dc | 04/11/2011 02:43PM |
| myappname-3 | d1b99f31-203c-48e3-aa17-17ca26178a49 | 04/11/2011 02:44PM |
+----------+-------------+---------------+-------------+--------------+
```

Use the cf-crashlogs script to view log files for crashed instances:

```
$ grails cf-crashlogs
==== logs/stdout.log ====
2011-03-03 14:50:23,904 [main] ERROR context.ContextLoader  - Context initialization failed
org.springframework.beans.factory.access.BootstrapException: Error executing bootstraps; nes
        at org.codehaus.groovy.grails.web.context.GrailsContextLoader.createWebApplicationCo
…
Caused by: java.lang.NullPointerException
        at $Proxy13.save(Unknown Source)
…
Stopping Tomcat because the context stopped.
```

Logs
Use the cf-logs script to view log files:

```
$ grails cf-logs
==== logs/stderr.log ====
Mar 3, 2011 2:51:32 PM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-50478
…
INFO: Server startup in 9169 ms

==== logs/stdout.log ====
```

By default all logs are displayed, but you can limit it to one or more:

```
$ grails cf-logs --stderr
==== logs/stderr.log ====
Mar 3, 2011 2:51:32 PM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-50478
…
INFO: Server startup in 9169 ms
```

And you can also write a log to a file instead of the console:

```
$ grails cf-logs target/myapp.stderr.log --stderr
Wrote logs/stderr.log to target/myapp.stderr.log
```

Viewing files
Use the cf-list-files script to view directory listings:

```
$ grails cf-list-files /
logs/                                          -
tomcat/                                        -
```

```
$ grails cf-list-files tomcat
bin/                                           -
conf/                                          -
lib/                                           -
logs/                                          -
temp/                                          -
webapps/                                       -
work/                                          -
```

```
$ grails cf-list-files tomcat/webapps
ROOT/                                          -
```

```
$ grails cf-list-files tomcat/webapps/ROOT
META-INF/                                      -
WEB-INF/                                       -
css/                                           -
images/                                        -
js/                                            -
plugins/
```

Use the cf-get-file script to view a file:

```
$ grails cf-get-file tomcat/webapps/ROOT/css/main.css
html * {
    margin: 0;
    /*padding: 0; SELECT NOT DISPLAYED CORRECTLY IN FIREFOX */
}
...
```

And you can also write the contents to a file instead of the console:

```
$ grails cf-get-file tomcat/webapps/ROOT/css/main.css target/main.css
Wrote tomcat/webapps/ROOT/css/main.css to target/main.css
```

7 Service Configuration

When you provision a service and bind it to an application, an environment variable `VCAP_SERVICES` is set for your server instance(s). This is a JSON string and you can use it to configure your DataSource, MongoDB, and Redis connections and will look something like this (formatted for readability):

```
{"redis-2.2":
   [{"name":"redis-1d8e28a",
     "label":"redis-2.2",
     "plan":"free",
     "credentials":{
         "node_id":"redis_node_3",
         "hostname":"172.30.48.42",
         "port":5004,
         "password":"1463d9d0-4e35-4f2e-be2f-01dc5536f183",
         "name":"redis-1a69a915-6522-496c-93d5-1271d2b3118e"}
     }],
 "mongodb-1.8":
   [{"name":"mongodb-3854dbe",
     "label":"mongodb-1.8",
     "plan":"free",
     "credentials":{
         "hostname":"172.30.48.63",
         "port":25003,
         "username":"b6877670-da98-4124-85ca-84357f042797",
         "password":"f53e6a4b-f4b8-497d-ac81-43cb22cf1e88",
         "name":"mongodb-9dda2cfb-9672-4d58-8786-98c3abcb21ec",
         "db":"db"}
   }],
 "mysql-5.1":
   [{"name":"mysql-497b12e",
     "label":"mysql-5.1",
     "plan":"free",
     "credentials":{
         "node_id":"mysql_node_8",
         "hostname":"172.30.48.27",
         "port":3306,
         "password":"p3rO5K5lRZaEU",
         "name":"d887d4f5664f64dde86f3ce42c6333962",
         "user":"umuPIJ8IzSKVA"}
     }]
}
```

Fortunately the plugin manages this for you, so in general you don't need to be aware of this. It will update your DataSource configuration if you have a MySQL service provisioned and bound, and likewise your MongoDB and Redis connection information if you have those plugins installed and have those services provisioned and bound. If you're not using the MongoDB or Redis plugin then you'll need to access the information yourself - just call `System.getenv('VCAP_SERVICES')` and parse the JSON.

DataSource

In addition to replacing the username, password, and JDBC url for your DataSource, the plugin will configure MySQL connection testing, using the following settings:

```
removeAbandoned = true
removeAbandonedTimeout = 300 // 5 minutes
testOnBorrow = true
validationQuery = '/* ping */ SELECT 1'
```

This will only be done if the DataSource is the standard `org.apache.commons.dbcp.BasicDataSource` that Grails configures since we assume that if you have customized the DataSource, you will configure connection testing yourself. You can also tell the plugin to not make these changes with

```
grails.plugin.cloudfoundry.datasource.disableTimeoutAutoconfiguration = true
```

in `Config.groovy`.
You should specify the `Dialect` to ensure that the correct table type is used. InnoDB tables are preferred since MyISAM tables do not support transactions, so it's a good idea to use the `MySQLInnoDBDialect`. In addition you can customize the JDBC url (for example to use UTF-8). The best way to configure these is to populate the `production` settings (or whichever environment you're using for deployment) with these values, for example:

```
production {
   dataSource {
      dialect = org.hibernate.dialect.MySQLInnoDBDialect
      driverClassName = 'com.mysql.jdbc.Driver'
      username = 'n/a'
      password = 'n/a'
      url = 'jdbc:mysql://localhost/db?useUnicode=true&characterEncoding=utf8'
      dbCreate = 'update'
   }
}
```

`username` and `password` will be replaced but can't be blank or missing, so they're set to dummy values. In addition the part of the URL to the left of the ? character will be replaced, so that just has to be valid syntax but doesn't need to be a real database. The part to the right of the ? (`"useUnicode=true&characterEncoding=utf8"`) will be appended to the auto-generated URL.

Searchable
Although there's no explicit support for Compass in Cloud Foundry, the plugin does detect that the Searchable plugin is installed and will configure it to write its Lucene index to a writeable directory on your server.

8 Troubleshooting

You can enable logging of the JSON responses from client calls by setting the `grails.plugin.cloudfoundry.Scripts` category to the `debug` level in your log4j configuration, e.g.

```
debug 'grails.plugin.cloudfoundry.Scripts'
```

You can also log all client method calls with

```
trace 'grails.plugin.cloudfoundry.ClientWrapper'
```

Some exceptions will always show a stack trace, but others are somewhat expected and just the error messages are displayed, but you can print all stacktraces by adding this to `Config.groovy`

```
grails.plugin.cloudfoundry.showStackTrace = true
```