Table of contents

This allows deployment of your grails applicaiton WAR file directly to an application server. This plugin uses the cargo libraries to deploy WAR files and manage servlet containers. If you wish to check to see if your servlet container is supported, see: http://cargo.codehaus.org/

Deploy Plugin - Reference Documentation

Authors: Aaron J. Zirbes

Version: 0.1

Table of Contents

1 Introduction

There are many automated deployment solutions out there: chef, puppet, ant and gradle, among others. This plugin aims to let you deploy your grails application from within the Grails environment by leveraging the cargo libraries.

To find out if your specific set of servlet containers are supported, please visit the cargo page and look under **Container Support**.

1.1 History

- Initial public release 0.1
  - June 22th, 2012

2 Installation

To install this plugin in your application, add the following to your BuildConfig.groovy file under the `plugins` setction

```
grails.project.dependency.resolution = {
…
    plugins {
    …
        provided ":deploy:0.1"
    …
    }
…
}
```

3 Configuration

This plugin requires you to configure a destination container. The default container used by the 'deployer' command is configured in your BuildConfig.groovy, or alternatively in your ~/.grails/settings.groovy file.

> ⚠ It is suggested that you keep at least the username and password in your ~/.grails/settings.groovy file so that it is not tied to your project source code. This is a safer location, expecially if your source code is shared with others, or is an open source project.

Defaults

The defaults settings should be enough to get most people up and running. These are the settings that are used when the `deployer` command is called without any arguments.

> ⚠ The following settings should be prefaced by: `grails.plugins.deploy.defaults.` For example the `containerId` setting would become `grails.plugins.deploy.defaults.containerId`

Option	Default Value	Description
containerId	'tomcat7x'	The container you are deploying to, such as Tomcat, Glassfish, JBoss, WebLogic, etc..
containerType	'remote'	The type of container connection. 'remote' should be fine for most scenarios.
deployerType	'remote'	The type of deployer used to deploy the WAR file. 'remote' should be fine for most scenarios.
configurationType	'runtime'	The type of container configuration used. 'runtime' should be fine for most scenarios.
propertySet	`[protocol:'http', hostname:'localhost', 'servlet.port': 8080 ]`	a map containing all the properties to set for the configuration of the servlet container deployment. See the section below regarding the options available for the `propertySet` configuration option.

containerId

This must match the containerId of your servlet container from the list provided on the cargo site. The default containerId is 'tomcat7x'

containerType

One of 'remote', 'installed' or 'embedded'. See the class definition for details.

deployerType

One of 'remote', 'installed' or 'embedded'. See the class definition for details.

configurationType

One of 'runtime', 'existing' or 'standalone'. See the class definition for details.

The use of anything other than 'runtime' is not recommended unless you know what you are doing.

propertySet

This is a map containing the cargo configuration properties that are used to connect to the container and deploy or control the WAR file or application.

Some common settings would look like

```
grails.plugins.deploy.defaults.propertySet = [
            'remote.username': 'autodeployer',
            'remote.password': 'UGheoqu0Washu2aa0Li1',
            protocol: 'http',
            hostname: 'localhost',
            'servlet.port': 8080 ]
```

For a complete list of properties, see the cargo containt values page. An alternate listing is available on the properties page

Destinations

To manage configurations for more than one destination, you can set up a map containing destiniation specific settings. This is useful for setting up a different configuration for testing vs. staging vs. production. You can also set up configurations for multiple servers if you deploy your app to multiple production servers either for load balancing, or for geographical reasons.

⚠ You only need to set the options that are **different** than those set in your `grails.plugins.deploy.defaults` setting. This will keep the size of your `destinations` settings down.

An example configuration for a few different destinations would look like

```
grails.plugins.deploy.destinations = [
    // staging deploy destination
    staging: [
        propertySet: [
            protocol: 'https',
            hostname: 'testing.example.org',
            'servlet.port': 8443 ] ],
    // production deploy destination
    production: [
        propertySet: [
            protocol: 'https',
            hostname: 'www.example.org',
            'servlet.port': 443 ] ],
    // local test server destination
    testing: [
        containerId: 'tomcat6x',
        propertySet: [
            'remote.username': 'deployer',
            'hostname': '192.168.6.142',
            'servlet.port': 80 ] ] ]
```

Groups

The `grails.plugins.deploy.groups` setting lets you set up different groups you can use to deploy your application WAR file to. The following example sets up an east coast and a west coast group. These group names will then be passed to the --destination-group command line parameter.

```
grails.plugins.deploy.groups = [
    westCoast: [ 'seattle', 'sandiego' ],
    eastCoast: [ 'maryland', 'newyork'] ]
```

Each of the destinations above would also need to have a corresponding entry in the `grails.plugins.deploy.destinations` configuration option. For example:

```
grails.plugins.deploy.destinations = [
    …
    seattle: [
        propertySet: [ hostname: 'app-seattle.example.org' ]
   …
   ]
```

4 PropertySet options

The following propertySet options were pulled from the page: http://cargo.codehaus.org/maven-site/cargo-core/apidocs/constant-values.html

⚠ If when using properties, please exclude the 'cargo.' prefix from the property name as this is automatically added.

- datasource.datasource
- datasource.driver
- datasource.id
- datasource.jndi
- datasource.password
- datasource.properties
- datasource.transactionsupport
- datasource.type
- datasource.url
- datasource.username
- geronimo.log.console
- geronimo.log.file
- geronimo.servlet.containerId
- geronimo.users
- glassfish.adminPort
- glassfish.domain.debug
- glassfish.domain.jmxPort
- glassfish.domain.name
- glassfish.http.ssl.port
- glassfish.java.debugger.port
- glassfish.jms.port
- glassfish.orb.listener.port
- glassfish.orb.mutualauth.port
- glassfish.orb.ssl.port
- glassfish.osgi.shell.telnet.port
- glassfish.portbase
- hostname

- java.home
- jboss.classloading.webservice.port
- jboss.clustered
- jboss.configuration
- jboss.ejb3.remoting.port
- jboss.invoker.pool.port
- jboss.jmx.port
- jboss.jrmp.invoker.port
- jboss.jrmp.port
- jboss.management.port
- jboss.naming.port
- jboss.osgi.http.port
- jboss.password
- jboss.profile
- jboss.remotedeploy.hostname
- jboss.remotedeploy.port
- jboss.remotedeploy.timeout
- jboss.remoting.transport.port
- jboss.transaction.recoveryManager.port
- jboss.transaction.statusManager.port
- jboss.user
- jetty.createContextXml
- jetty.deployer.url
- jetty.servlet.default.useFileMappedBuffer
- jetty.session.path
- jonas.cluster.name
- jonas.configurator.
- jonas.deployable.identifier
- jonas.domain.name
- jonas.jms.port
- jonas.jndi.initial.context.factory
- jonas.jndi.mejb.path
- jonas.mejb.jaas.entry

- jonas.mejb.jaas.file
- jonas.server.name
- jonas.services.list
- jonas.undeploy.ignoreVersion
- jrun.server.name
- jrun4x.classpath
- jrun4x.home
- jsr88.deploytoolclasspath
- jsr88.deploytooljar
- jsr88.password
- jsr88.user
- jvmargs
- logging
- oc4j.admin.pwd
- oc4j.auto-deploy.dir
- process.spawn
- protocol
- remote.password
- remote.uri
- remote.username
- resin.socketwait.port
- resource.class
- resource.name
- resource.parameters
- resource.resource
- resource.type
- rmi.port
- runtime.args
- servlet.port
- servlet.uriencoding
- servlet.users
- ssh.host
- ssh.keyfile

- ssh.password
- ssh.remotebase
- ssh.username
- standalone.ignoreNonExistingProperties
- tomcat.ajp.port
- tomcat.connector.emptySessionPath
- tomcat.context.reloadable
- tomcat.copywars
- tomcat.httpSecure
- tomcat.webappsDirectory
- weblogic.administrator.password
- weblogic.administrator.user
- weblogic.bea.home
- weblogic.configuration.version
- weblogic.deployable.folder
- weblogic.domain.version
- weblogic.logging
- weblogic.server

5 Usage

This plugin provides the script deployer to be used from the command line, or from within the Grails interative mode. Please see the command line usage for details.

For more advanced use, you can use the org.zirbes.grails.deploy.DeployerService to deploy WAR files from within your application, or from within your own Grails scripts.

Please see the groovy docs for more information on the ConfigBuilder and DeployerService classes.