Grails

Cyrille Le Clerc
2013/06/03 09:51

Table of Contents

Contents

Grails is a self-contained project environment with its own tools for performing project tasks like creating models, controllers and running applications. This self contained design does not use a standard WAR-file layout, so the easiest way to work with Grails is to create and develop using the normal grails tools, and then export the grails application as a WAR file that can be deployed to CloudBees using the bees app:deploy command.

Initial setup

First, make sure you have installed Grails.GVM is a convenient tool to install Grails:

```
curl -s get.gvmtool.net | bash
gvm install grails
```

Creating your Grails app

This will create a standard grails directory structure where you can use your standard grails commands.

```
grails create-app my-app
cd my-app
```

Install Grails Wrapper, it will be particularly useful for the Continuous Integration setup on Jenkins.

```
grails wrapper
```

Integrate the project with your IDE ("`--intellij--`" or "`--eclipse--`")

```
grails integrate-with --intellij
```

Running your Grails app locally

To run your grails app, you need to use the standard grails run-app command.

```
grails run-app
```

Deploying your Grails app on CloudBees

Creating MySQL and Tomcat instances

Deploying an application often requires to associate it with a database. Here is how to create a MySQL database.

- Create the "`my-db`" database
  ```
  bees db:create my-db
  ```
- Create the "my-app" Tomcat container
  ```
  bees app:create -a my-app jvmPermSize=128
  ```

- Bind the MySQL database to the Tomcat container to create the JNDI DataSource
  bees app:bind -a my-app -db my-db -as mydb
  - Where "mydb" is the name of the database binding. The JNDI name is "`java:comp/env/jdbc/mydb`"

Setting the Grails Environment of the Tomcat server

The Grails system property "`grails.env`" is the standard Grails mechanism to manage environment-specific configuration parameters such as database configuration (see Grails Environments).

You can define this "`grails.env`" System Property with the CloudBees SDK.

Associate your application CloudBees Tomcat instance "`my-app`" with the "production" environment:

```
bees config:set -a my-app -P grails.env=production
```

Using a JNDI datasource in Grails

Grails is by default configured to use an In Memory H2 database. You have to replace this H2 Datasource declaration for the Grails environment associated with this CloudBees Tomcat instance:

Overwrite the "`production`" environment DataSource to use the "`mydb`" JNDI DataSource ("`java:comp/env/jdbc/mydb`").

```
environments {
  ...
  production {
    dataSource {
      dialect = 'org.hibernate.dialect.MySQL5InnoDBDialect'
      pooled = false
      dbCreate = 'validate' // use 'update', 'validate', 'create' or 'create-drop'
      jndiName = 'java:comp/env/jdbc/mydb'
    }
  }
}
```

Where "`mydb`" is the name of the database binding.

More details at Grails Reference / Plugins / DataSource.

Add MySql driver dependency in "`grails-app/conf/BuildConfig.groovy`" file

```
dependencies {
  runtime 'mysql:mysql-connector-java:5.1.24'
}
```

Configuring Logging

To have Grails log statements appear in the CloudBees log file (and be accessible via the RUN console UI), a console appender needs to be defined.

Logging used by Grails is defined in the "`grails-app/conf/Config.groovy`" file.

```
// log4j configuration
log4j = {
  // Example of defining the default console appender
  //
   appenders {
      console name:'stdout', layout:pattern(conversionPattern: '%d [%t] %-5p %c - %m%n'), threshold:
org.apache.log4j.Level.DEBUG
   }//
   error  'org.codehaus.groovy.grails.web.servlet',  //  controllers
        'org.codehaus.groovy.grails.web.pages', //  GSP
        'org.codehaus.groovy.grails.web.sitemesh', //  layouts
        'org.codehaus.groovy.grails.web.mapping.filter', // URL mapping
        'org.codehaus.groovy.grails.web.mapping', // URL mapping
        'org.codehaus.groovy.grails.commons', // core / classloading
        'org.codehaus.groovy.grails.plugins', // plugins
        'org.codehaus.groovy.grails.orm.hibernate', // hibernate integration
        'org.springframework',
```

```
        'org.hibernate',
        'net.sf.ehcache.hibernate'
    warn   'org.mortbay.log'
    debug  'grails.app'
    root {
      error 'stdout'
      info 'stdout'
      warn 'stdout'
      debug 'stdout'
      additivity = true
    }
}
```

Packaging and deploying your Grails Application on CloudBees platform

Use the standard Grails war command to package your application as a standard WAR file.

grails war

Deploy the generated WAR file to CloudBees RUN@cloud

bees app:deploy -a my-app target/my-app-0.1.war

Continuous Integration with Grails and Jenkins on CloudBees DEV@cloud platform

Please see this detailed guide Continuous Integration with Grails and Jenkins.

Grails FAQ

Configuring the JVM PermGen space

To deploy your application with more PermGen space see JVM PermGen Space

Configuring Grails Environment ("`grail.env`")

The Grails system property 'grails.env' is the standard Grails mechanism to manage environment-specific configuration parameters such as database configuration (see Grails Environments).

Verifying active Grails environment ("`grail.env`")

You can verify the active Grails Environment of a deployed application looking for the value of the "`grails.env`" system property.
To do so, you need to list the System properties of your application with the CloudBees SDK.

```
$ bees config:list -a my-application
Application Parameters:
  grails.env=production
```

- "`-a my-application`": name of your application
- In this sample, the "`grails.env`" is set to "`production`"

Changing active Grails environment ("`grail.env`")

You can set the active Grails Environment of a deployed application setting the value of the "`grails.env`" system property with the CloudBees SDK.

```
$ bees config:set -a my-application -P grails.env=production
  Application config parameters for demo/my-application: saved
  Application Parameters:
    grails.env=production
$ bees app:restart -a my-application
```

- "`-a my-application`": name of your application

- In this sample, the "`grails.env`" is configured to "`production`"
- Don't forget to restart your application!