Table of contents

Adds support for Dropwizard

Dropwizard Plugin - Reference Documentation

Authors: Burt Beckwith

Version: 0.1

Table of Contents

1 Introduction to the Dropwizard Plugin

The Dropwizard plugin integrates Dropwizard with Grails to make it easy to create REST services usi while still using Grails features.

Dropwizard is very opinionated, so several framework and library choices are fixed and cannot be changec

- you must use the Jetty server, Tomcat isn't an option
- Jackson
- Jersey
- Logback (this plugin depends on the Grails logback plugin to provide this)

In addition, the plugin enables the use of Freemarker and Mustache templates for response rendering, alth can continue to use Dropwizard's JSON support, or other text-based syntaxes such as XML

Note that since Dropwizard requires Jetty and the current version of Jetty in the Grails Jetty plugin is too server plugin (i.e. "tomcat") from `BuildConfig.groovy`. Use the plugin's `run-dropwizard` script

1.1 History

History

- March 5, 2013
  - initial 0.1 release

2 Getting Started

The first step is to add a dependency for the plugin in `BuildConfig.groovy`:

```
plugins {
   …
   compile ':dropwizard:0.1'
}
```

Next, create a YAML file that defines the Dropwizard configuration as described in the section on co
application classes (resources, health checks, etc.) as described in the general usage section.

3 Configuration

Much of the Dropwizard configuration is done in a YAML file. This includes the server and admin H database settings, etc. This can still be done when using this plugin, but it's best to leverage what is alrea practical. This can include defining database connectivity in `DataSource.groovy`, but can also e NoSQL plugin for persistence, and other plugins and Grails features. For example, Dropwizard has Liquibase, but since these are trivial to use in Grails it makes more sense to use the Grails integrations.

Application classes

In a typical Dropwizard application, you define a service class that extends `com.yammer.dropwiza` `main` method, and this is the entry point to the application. It configures other helper classes including health checks, tasks, etc. Using the plugin, much of this is done for you but you can customize most option

There are three ways to register application classes. The easiest is to use the typical Grails convention- and create Groovy classes in `grails-app/dropwizard`. These can include resource classes, which m in "DropwizardResource", health check classes ending in "HealthCheck", and task classes ending in "Dro must all have default no-arg constructors and must be written in Groovy.

For more flexibility, you can register classes as Spring beans in `grails-app/conf/spring/r` enable the appropriate "autoRegister" property as described below to have the plugin find them and regi you. For even more flexibility, you can define a Closure in `Config.` `grails.plugin.dropwizard.initializer` key that can do additional customization.

YAML

See the Dropwizard documentation for the supported syntax for the configuration file, but you can use thi (remove the # characters to uncomment lines):

```
http:
   port: 8080

#   adminPort: 8081

#   2-1000000
#   maxThreads: 1024

#   1-1000000
#   minThreads: 8

#   rootPath: /*

#   one of blocking, legacy, legacy+ssl, nonblocking, nonblocking+ssl
#   connectorType: blocking

#   maxIdleTime: 200s

#   1-128
#   acceptorThreads: 1

#   -Thread.NORM_PRIORITY-Thread.NORM_PRIORITY)
#   acceptorThreadPriorityOffset: 0

#   min -1
#   acceptQueueSize: -1

#   min 1
#   maxBufferCount: 1024
```

```
#   requestBufferSize: 16KB
#   requestHeaderBufferSize: 6KB
#   responseBufferSize: 32KB
#   responseHeaderBufferSize: 6KB

#   reuseAddress: true
#   soLingerTime: null
#   lowResourcesConnectionThreshold: 0
#   lowResourcesMaxIdleTime: 0s
#   shutdownGracePeriod: 2s
#   useServerHeader: false
#   useDateHeader: true
#   useForwardedHeaders: true
#   useDirectBuffers: true
#   bindHost: null
#   adminUsername: null
#   adminPassword: null

#   requestLog:
#      console:
#         enabled: true
#         threshold: Level.ALL
#         timeZone: UTC
#         logFormat: null
#      file:
#         enabled: false
#         threshold: Level.ALL
#         currentLogFilename: null
#         archive: true
#         archivedLogFilenamePattern: null
#         # 1-50
#         archivedFileCount: 5
#         timeZone: UTC
#         logFormat: null
#
#      syslog
#         enabled: false
#         threshold: Level.ALL
#         host: localhost
#         # auth, authpriv, daemon, cron, ftp, lpr, kern, mail, news, syslog, use
local1, local2, local3, local4, local5, local6, local7
#         facility: local0
#         timeZone: UTC
#         logFormat: null

#      timeZone: UTC

#   gzip:
#      enabled: true
#      minimumEntitySize: 256B
#      bufferSize: 8KB
#      excludedUserAgents: ImmutableSet<String>
#      compressedMimeTypes: ImmutableSet<String>
```

```
#   ssl:
#      keyStore: /path/to/file
#      keyStorePassword: null
#      keyManagerPassword: null
#      keyStoreType: JKS
#      trustStore: /path/to/file
#      trustStorePassword
#      trustStoreType: "JKS"
#      needClientAuth: true/false
#      wantClientAuth: true/false
#      certAlias: null
#      allowRenegotiate: true/false
#      crlPath: /path/to/file
#      crldpEnabled: true/false
#      ocspEnabled: true/false
#      maxCertPathLength:
#      ocspResponderUrl:
#      jceProvider:
#      validatePeers
#      supportedProtocols:
#         - SSLv3
#         - TLSv1
#         - TLSv1.1
#         - TLSv1.2

#   contextParameters:
#      ImmutableMap<String, String>
```

Config.groovy

There are a few configuration options for the plugin that are defined in Config.groovy:

Property	Default	Meaning
grails.plugin.dropwizard. banner	none; look for a file named `banner.txt` in the classpath	the string to display as the startup banner
grails.plugin.dropwizard. dropwizardContext	"dropwizard"	the string to display as the startup banner
grails.plugin.dropwizard. autoRegisterResources	`false`	whether to discover all resource classes regis those with a `javax.ws.rs.Path` annotation
grails.plugin.dropwizard. autoRegisterHealthChecks	`false`	whether to discover all health check classes reg those that extend `com.yammer.metrics.c`
grails.plugin.dropwizard. autoRegisterManaged	`false`	whether to discover all managed classes regis those that im `com.yammer.dropwizard.lifecycle.`
grails.plugin.dropwizard. autoRegisterLifeCycle	`false`	whether to discover all lifecycle classes regis those that im `org.eclipse.jetty.util.component`
grails.plugin.dropwizard. autoRegisterAnnotatedProviders	`false`	whether to discover all provider classes regis those with a `javax.ws.rs.ext.Provider`
grails.plugin.dropwizard. autoRegisterInjectableProviders	`false`	whether to discover all injectable provider c beans (i.e. those that `com.sun.jersey.spi.inject.Inject`
grails.plugin.dropwizard. autoRegisterTasks	`false`	whether to discover all task classes registered that extend `com.yammer.dropwizard.ta`
grails.plugin.dropwizard. yamlPath	`"classpath: dropwizard.yml"`	the location of the YAML configuration file; syntax
grails.plugin.dropwizard. assets	none	a Map of asset paths to register as `AssetsB` resource paths (relative to the classpath) and th names to serve as
grails.plugin.dropwizard. serviceClassName	`"grails.plugin. dropwizard. GrailsService"`	the name of the service class that configures t constructor that takes a Spring `Applicati` pointing at the YAML configuration file
grails.plugin.dropwizard. initializer	none	optional closure that can do additional configura `com.yammer.dropwizard.config.B` `com.yammer.dropwizard.config.Er` `grails.plugin.dropwizard.config.` , and @org.codehaus.groovy.grails.commons.Gr

4 General Usage

Usage

Start the application with

```
$ grails run-dropwizard
```

Use CTRL-C to stop.

URLs

As a hybrid application, the Grails URLs and Dropwizard URLs use different context paths. For example be configured under http://www.servername.com:8080/app/ and the Dropwiz http://www.servername.com:8080/dw/. It would be more convenient to have one http://www.servername.com:8080/app/ and http://www.servername.com:8080/app/dw/ but given the appr determine the resource handlers, this isn't possible.

The context for your Grails controllers is determined the same way as when not using this plugin, i.e. name but can be overridden. Specify the Dropwizard context with the `grails.plugin.dropwizar` attribute in `Config.groovy` - it defaults to "dropwizard".

Admin URIs

Dropwizard has an admin servlet with some convenient URIs to monitor your application. By default thi include:

- http://server:8081/dropwizard/metrics
  - displays extensive runtime and usage information; append ?pretty=true to pretty-print the JSON
- http://server:8081/dropwizard/healthcheck
  - runs all health checks and displays their statuses
- http://server:8081/dropwizard/threads
  - displays a thread dump
- http://server:8081/dropwizard/ping
  - responds with "pong" as a simple test that something is there

Utility methods

Dropwizard displays all known endpoints at startup, but this information is also available at runti `dropwizardService` bean and call

```
def dropwizardService
...
def endpoints = dropwizardService.findEndpoints()
```

This will be a `List` of `grails.plugin.dropwizard.util.EndpointData`

You can also retrieve information from Dropwizard via the `dropwizard` Spring bean. This is simply a `configuration` (the `com.yammer.dropwizard.config.Configuration`), `e` `com.yammer.dropwizard.config.Environment`), and `servic` `grails.plugin.dropwizard.GrailsService`)

Domain Object serialization

You can call GORM and use domain classes from your Dropwizard REST resources. Note however tha domain classes using Dropwizard's automatic marshalling since there's no way to exclude non-serializabl classes (DTOs) built from domain class instances (they can be written in Groovy) instead.

Reloading

Reloading doesn't currently work. I'm looking into supporting runtime reloading in development mode, a the fly.