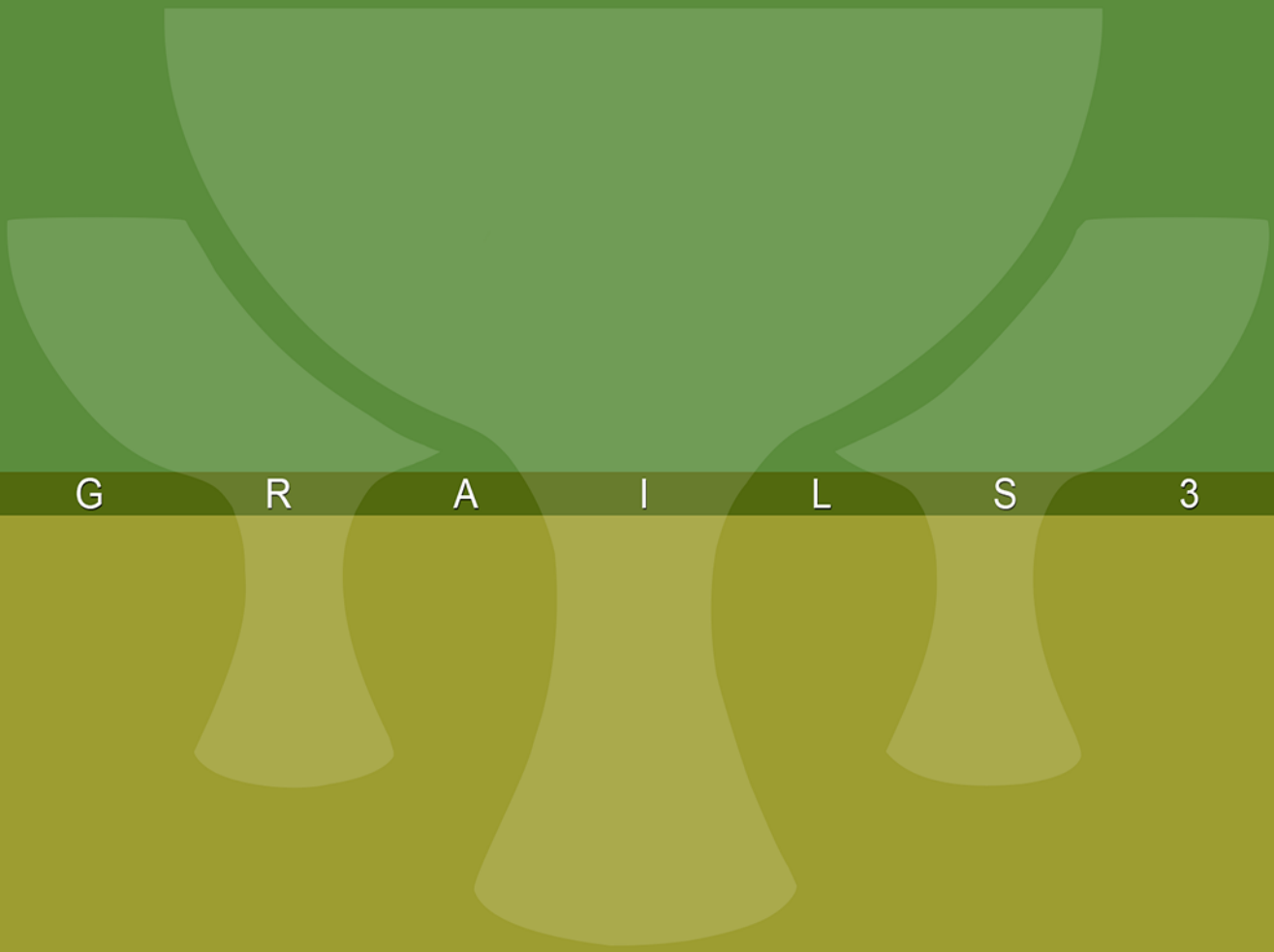
GRAILS 3

STEP BY STEP



G R A I L S 3

Greenfield applications made right with Grails 3

Grails 3 - Step By Step

Greenfield applications made right with Grails 3

Cristian Olaru

Contents

How this book is organized

We try to describe here how a complete greenfield application can be implemented with Grails 3 in a fast way using profiles and plugins - and we do this in the sample application that accompanies this book. You can find news about this book on its peresentation site located here: www.grailsthreebook.com[1]. This book is not a replacement of the Grails 3 Reference Documentation[2] which is a complete description of Grails 3[3], made by the creators of the framework.

The source code used in this book is part of this sample application which is a free project hosted on GitHub on this location https://github.com/colaru/mvp-application[4] and is exposed online here http://application.eu-central-1.elasticbeanstalk.com[5]. If a source code fragment is included in this book, then it is taken from this project. We use BDD or other specification by example techniques for describing the sample application specification. The application code is tested in a TDD style using automated unit, integration and functional tests.

The sample application is based on a multi project Gradle build so the application can be automatically built using Gradle - the build tool used by Grails 3. The sample application is available online and the deployment is done automatically to AWS cloud in a Continuous Deployment style using Jenkins.

We use links to various external resources in this book (you will see them in the page's footers), because this book is intended to be in electronic format from the beginning. The reader can use these links for consulting external references in an easy way.

The book has two parts.

First part - Grails 3 Essentials

An introduction to application development with Grails 3; we describe the main application that will be implemented in the second part of the book

Chapter 1 is an introduction to the framework. You will find here some history of the framework and a lot of links to the Grails resources; it is presenting the classical three layered architecture for a web application provided by Grails 3

Chapter 2 describes how to start your work with Grails 3 framework and how to install all the tools for a free working environment - free as much as possible because you have to pay some money for the best tools on the market

[1] https://grailsthreebook.com
[2] http://docs.grails.org/latest
[3] https://grails.org
[4] https://github.com/colaru/mvp-application
[5] http://application.eu-central-1.elasticbeanstalk.com

Chapter 3 presents the project we want to implement as an example; it is presenting also the way we choose to work on implementing the sample application, based on BDD, TDD and CI, CD; it shows how the main application is split in parts based on Grails 3 profiles

Second part - Practical example with Grails 3

A practical implementation of a greenfield application with Grails 3; the application is composed from multiple parts corresponding to various Grails 3 profiles

Chapter 4 describes the implementation of the application admin portal that will be used in Intranet by the site administrators, based on a classical Grails 3 Web Profile

Chapter 5 describes the implementation of the application site exposed to the Internet to the customers, that is based on Grails 3 Angular Profile

Chapter 6 describes a REST web API exposed with Grails 3 Rest Profile and consumed by a mobile hybrid application created with Ionic (can be published in Google Play, Apple Store and Windows Store)

Chapter 7 describes a Microservice developed with Grails 3 Micro Profile

Introduction to Grails 3

Grails is a full stack Web framework, covering all the aspects of a modern web application development. Its plugins system is unique in the Java world - you can divide your application into complete functional modules, containing presentation and business logic. Because it runs in JVM, it has access to the entire Java ecosystem. Because is written in Groovy, it has access to all Groovy libraries and tools.

The main site for Grails 3 is grails.org[6]. Here is the definition of Grails on the official site:

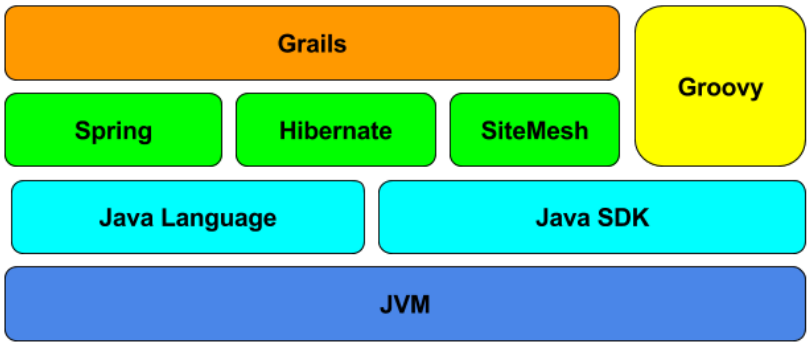
> Grails is a powerful web framework for the Java platform, aimed at multiplying developers' productivity, thanks to Convention-over-Configuration, sensible defaults, and opinionated APIs. It integrates smoothly with the JVM, allowing you to be immediately productive, whilst providing powerful features, including integrated ORM, Domain-Specific Languages, runtime and compile-time meta-programming, and Asynchronous programming.

If you have developed multiple applications over time, you have recognized a set of principles and good practices that can be reused. You don't want to start from scratch each time you build your new application. Grails has good technical principles that make it a good starter for your new applications. With Grails, you have an established architecture from the beginning, and you can implement new features in an easy way, using scaffolding and the plugins ecosystem. And this is more valuable when you have to get to the market first, before your competitors.

Some history

Grails 3 is based on a stack of other technologies Java[7], Spring[8], Groovy[9], Hibernate[10], Sitemesh[11]. We try here to show the history of these technologies and how they are used in Grails 3.

[6] https://grails.org/index.html
[7] https://www.oracle.com/java
[8] https://spring.io
[9] http://groovy-lang.org
[10] http://hibernate.org
[11] http://wiki.sitemesh.org



Technologies inside Grails

Java history

First it was Java, and was created by *Sun Microsystems*[12] and its father was *James Gosling*[13]. Java came with the idea of JVM - *Java Virtual Machine* that lets you *Write once, Run everywhere*. Everywhere means on any operating system. You write the code compiled to a byte code runnable by a virtual machine. And the virtual machine specification has an implementation for virtually any operating system.

Because Sun was bought by Oracle, now Java is in Oracle's portfolio. Java is used by millions of developers (Oracle claims 10 million) and is the number one TIOBE[14] index of most used languages in the world.

Why was Java such a strong language from the beginning? Some clues:

- It was totally object-oriented from the beginning and let people think *everything is an object* (this is the name of a chapter from one of the best Java books Thinking in Java - TIJ[15]) written by Bruce Eckel[16].
- It eliminates the *pointers* and memory deallocation, letting this be handled by the JVM and the garbage collector; the GC is constantly improved by the Oracle team. G1 GC[17] is the most advanced GC these days.
- It has multithreading inside from the beginning.
- It has exceptions handling inside.
- It has modularity inside, represented by packages and the possibility of packaging libraries in *.jar* files that can be reused between projects; there are a lot of libraries on the market, and as proof, you can search a Maven global repo[18]

[12] https://en.wikipedia.org/wiki/Sun_Microsystems
[13] https://en.wikipedia.org/wiki/James_Gosling
[14] http://www.tiobe.com/tiobe_index
[15] http://mindview.net/Books/TIJ4
[16] https://twitter.com/bruceeckel
[17] http://www.oracle.com/technetwork/tutorials/tutorials-1876574.html
[18] https://mvnrepository.com/

Java EE history

After this was the *Java EE* (known in the past as J2EE or JEE) – an attemptto make it easy to create enterprise applications (compared to other 2 profiles *Java SE* for desktop applications and *Java ME* for mobile applications). The language and Java EE is specified by a committee named JCP[19] *Java Community Process*, which is responsible for creating *JSRs*, specifications implemented by various vendors. A reference implementation for any specification is proof the specification can be implemented.

For example, the last specification for the Java Servlet is JSR 315: JavaTM Servlet 3.0 Specification[20] and the reference implementation for this standard is Oracle GlassFish 3.x (RI)[21]. Grails embeds by default inside, in DEV mode, a Tomcat[22] server and you can see what version of Java Servlet is implemented on each Tomcat version[23].

Here are the Java EE Full Platform Compatible Implementations[24] on different Java EE versions. The current version is Java EE 7, and here is the official tutorial[25]

Spring history

But the *Java EE* standards were too *de jure* for the Java community, compared to a lot of frameworks that emerged in the Java open source space, considered more *the facto standards* like: Struts for Web, Hibernate for persistence, etc. And then Spring Framework was born as a response to this *committee style* and *specification centric* style of driving the Java future - Rod Johnson[26] and others put the foundation, and Juergen Hoeller[27] is now responsible for the framework as part of the Pivotal portfolio. The framework introduced Dependency Injection and reaffirmed the power of POJOs (Plain Old Java Objects) - in contrast to EJBs (Enterprise Java Beans) which were at version 2 when Spring emerged - things have improved in EJB 3.

In time, these two technologies are evolving in parallel. For example, Java EE adopted the DI in its CDI, some implementations of JPA wrap Hibernate. Spring embraces annotations for configurations, instead of XMLs, as in Java EE. And Spring Framework has integration with some JSRs and is based on some technologies provided by Java EE.

The main documentation for Spring Framework is the reference documentation[28].

Groovy History

Java is an imperative style programming language and, compared with functional programming languages, it suffers from accidental complexity as Venkat Subramaniam[29] says.

[19] https://www.jcp.org/en/home/index
[20] https://jcp.org/en/jsr/detail?id=315
[21] https://glassfish.java.net
[22] http://tomcat.apache.org
[23] http://tomcat.apache.org/whichversion.html
[24] http://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html
[25] https://docs.oracle.com/javaee/7/tutorial
[26] https://twitter.com/springrod
[27] https://spring.io/team/jhoeller
[28] http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle
[29] https://twitter.com/venkat_s

> Imperative code is packed with accidental complexity.

Groovy is a functional programing language that is bringing functional style to Java programming. It introduces the notion of *closure* which is the idea of function as a main citizen of your code. It is also a scripting language because it has the characteristic of incorporating a lot of semantics in a short syntax.

The official Groovy site is groovy-lang.org[30]. Here is the definition of Groovy on the official site:

> Apache Groovy is a powerful, optionally typed and dynamic language, with static-typing and static compilation capabilities, for the Java platform aimed at improving developer productivity thanks to a concise, familiar and easy to learn syntax. It integrates smoothly with any Java program, and immediately delivers powerful features to your application, including scripting capabilities, Domain-Specific Language authoring, runtime and compile-time meta-programming and functional programming.

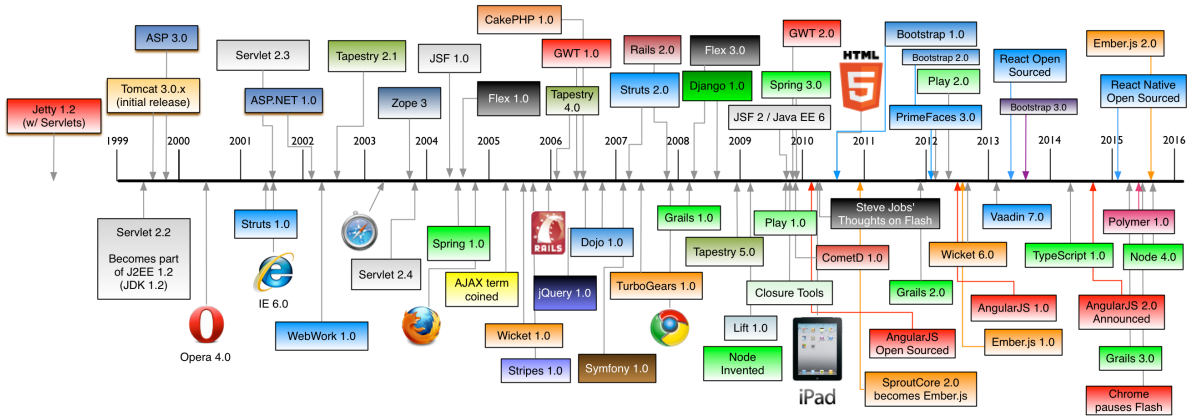
Some characteristics of Groovy

- it was a way to get functional programming into JVM; it introduced *closure* to the Java world long before the Lambda Expressions in Java 8[31]
- it is compiled to Java bytecode that is running in JVM (there is a *groovyc* compiler that is compiling Groovy code to Java bytecode in the same way the *javac* compiler is doing with Java code)
- Groovy scriptlet can be embedded in Java code and the resulted mixture is still working after compilation
- is enriching the Java classes with other convenient methods resulting in a new API named GDK[32]

Grails History

This is a timeline created by Matt Raible[33] where he tries to describe the history of Web Frameworks[34].

[30] http://www.groovy-lang.org
[31] http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Lambda-QuickStart/index.html
[32] http://www.groovy-lang.org/gdk.html
[33] https://twitter.com/mraible
[34] https://raw.githubusercontent.com/mraible/history-of-web-frameworks-timeline/master/history-of-web-frameworks-timeline.png



A history of Web frameworks

As you can see in this diagram the Grails framework was born in 2008 with its first GA release. Grails 2 was released in 2011. And Grails 3 was first released in 2015.

What makes Grails a good choice from the multitude of frameworks for building Web applications

- Grails is a full stack web framework in the sense that you can create an entire application using it. It covers the web presentation, business domain and database persistence aspects of your application. Not alone, but integrating the right technologies for this: Java, Spring, Hibernate. And all of these are linked by Groovy language with is used for configuring your application - the DSL for configurations are Groovy based (now in Grails 3 you will have the option to use YAML also)
- conventions over configurations
  - for creating a service, it is enough to place a groovy file containing a class without state in */service* folder without annotating it with *@Service* or mark it in a configuration file like in Spring or annotate with *@Stateless* for a EJB 3 stateless session bean
  - for creating an *entity*, it is enough to place a POJO class in a */domain* folder and no *@Entity* annotation like in JPA or Hibernate or declaring it in configuration files
- scaffolding for generating a seed application or a variety of elements starting from domain objects; it can be static (generating the corresponding files in filesystem) or dynamic (generating just proxies in memory)
- environments inside from the beginning like TEST, DEV, PROD (or a custom one) letting you take different actions in runtime based on the given environment
- testing inside the framework (unit - in isolation using mocking, integration, functional) using JUnit[35], Spock[36] and Geb[37]
- static resources served in an optimized way (*asset pipeline* plugin replacing *resources* plugin)
- asynchronous features based on plugins in first versions and directly inside in version 3 using Reactor[38]

[35] http://junit.org
[36] http://spockframework.org
[37] http://www.gebish.org
[38] https://projectreactor.io

- dynamic reload of classes and resources in runtime based on spring-loaded[39] technology and *on the fly reloading* in Tomcat (dev mode); no server restarts are needed in development or products like JRebel
- and many more…

Also, the history of the companies that stay before the framework is interesting. First it was supported by G2One Inc a company founded by the Groovy and Grails project leads, Guillaume Laforge and Graeme Rocher. It was acquired by SpringSource (first Interface 21) which was on background of Spring Framework. SpringSource was acquired by VMWare and transferred to the Pivotal portfolio. In 2015 at SpringOne conference in Washington the end of support from Pivotal to Grails and Groovy was announced and Grails is now supported by OCI and Groovy was adopted by Apache Foundation.

G2One Inc[40] -> SpringSource[41] -> VMware[42] -> Pivotal[43] -> OCI[44]

We present here a list of the main contributors to the framework and the plugins ecosystem (not all are working currently on this):

- Graeme Roche[45] - the father of framework with more than 10 years of development
- Jeff Scott Brown[46] - Cofounder, implementer of Rest API and Grails books writer
- Burt Beckwith[47] - Security guy (Spring security and other security plugins)
- Marc Palmer[48] - Resources plugins (replaced by asset pipeline suite of plugins)
- Alvaro Sanchez[49] - Rest security plugin

Here is the contributors list for Grails core Git repository[50] Here is the reference documentation[51]

Grails application architecture

A classical layered architecture

If we are talking about the architecture of a Grails 3 application, we should express this in architectural design patterns. And the right place to find architectural design patterns is this book Patterns Of Enterprise Application Architecture[52] written by Martin Fowler[53]. His catalog

[39] https://github.com/spring-projects/spring-loaded
[40] https://www.crunchbase.com/organization/g2one#/entity
[41] https://www.crunchbase.com/organization/springsource#/entity
[42] https://www.crunchbase.com/organization/vmware#/entity
[43] https://www.crunchbase.com/organization/pivotal#/entity
[44] https://www.ociweb.com/products/grails/
[45] https://twitter.com/graemerocher
[46] https://twitter.com/jeffscottbrown
[47] https://twitter.com/burtbeckwith
[48] https://twitter.com/marcpalmerdev
[49] https://twitter.com/alvaro_sanchez
[50] https://github.com/grails/grails-core/graphs/contributors
[51] http://docs.grails.org/latest/guide/single.html
[52] http://martinfowler.com/books/eaa.html
[53] http://martinfowler.com

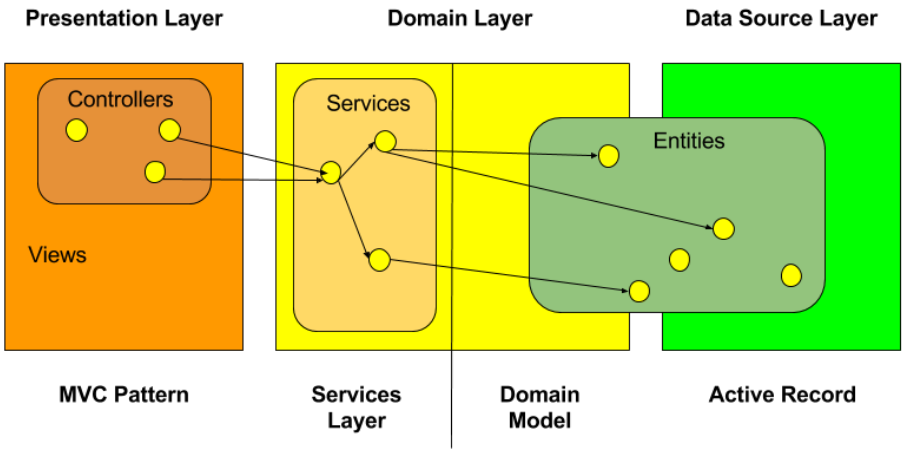
of design patterns[54] is available online. We will use a set of patterns from this book but other patterns are used inside the Grails integrated frameworks.

First of all, a Grails application is a classical three layered architecture composed from presentation, domain and data source layers.

Grails layered architecture

In a Grails application, the layers are represented by:

- Presentation Layer - is a classical Model View Controller - MVC[55] represented by Spring MVC framework[56]. The Spring MVC framework is wrapped by Grails which has some specific elements. In Grails, the *C* are the Grails *controllers*, the *V* are the Grails *GSP views* (rendered from *layouts*, *templates* and *tag libraries*), and the *M* are the domain *entities* (the entities are part of the domain layer but are reused in this layer). Also, the Spring MVC framework is based on Servlet API, JSPs, tag libraries and is responsible for interaction with the user via HTTP 1 (or 2)
- Domain Layer - Is split between a Services Layer[57] which are Spring singletons beans and a Domain Model[58] of *entities* which are the mappers to the database
- Data Source Layer - represented by GORM framework and domain entities which are classical Active Record[59]. Gorm offers access to Sql databases and is based on Hibernate which is based also on Java JDBC API and uses SQL for queries.



Grails three layered architecture

[54] http://martinfowler.com/eaaCatalog
[55] http://martinfowler.com/eaaCatalog/modelViewController.html
[56] http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html
[57] http://martinfowler.com/eaaCatalog/serviceLayer.html
[58] http://martinfowler.com/eaaCatalog/domainModel.html
[59] http://martinfowler.com/eaaCatalog/activeRecord.html

Because it is a technology stack, we have to deal with a lot of knowledge and abstraction.

Layer	**Presentation Layer**	**Domain Layer**	**Data Source Layer**
Patterns	MVC	Service Layer & Domain Model	Active Record
Frameworks and libraries	Spring MVC	Spring Beans, DI, Transactions	GORM (Hibernate and NoSql)
Java technologies	Servlets, JSP, tag libraries	POJO	JDBC
Basic technologies	HTML and HTTP	-	SQL

Some observations

- The domain/persistence entities are used in the entire Grails application from presentation to the persistence with different roles; they are primary persistence objects having state and persistence methods (CRUD operations but also finders generated dynamically by the Grails framework) but you can choose to enrich them with other functionality because they are part of the domain layer
- Dependency injection[60] (which is coming from Spring) is used for injecting services in other services and in controllers
- The services layer acts as a façade or as an API to the layer in front of it; typically, on this layer we will offer the ACID transactional support
- Controllers from the MVC is responsible just for managing the user interaction flow and will not contain any logic; the place for logic is in the services (and eventually in the domain entities)

New technologies inside Grails 3

Grails 3 is a rewrite of Grails 2 and now is based on Spring Boot and not just on Spring framework. The old build system (Gant scripts) are replaced with Gradle with is already used by Spring Boot. This migration has a downside - all the plugins have to be migrated to the new Grails version. The good thing is that the most important plugins were already beien migrated by the Grails community. And a set of plugins are not needed anymore because they can be replaced with Gradle plugins and Spring starters.

Spring Boot

You can access the reference documentation for SpringBoot[61].

Some advantages from having Spring Boot as your Grails 3 heart

[60] http://www.martinfowler.com/articles/injection.html

[61] http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle

- All Spring Boot starters[62] (starter is the SpringBoot plugin system) are available for your application
- In SpringBoot[63] you can have both Java and Groovy source files
- Some new technologies like websockets and reactive (Reactor project) are available now to your Grails application via Spring Boot

Gradle

In Grails 3, the old Gant[64] scripts were replaced with Gradle build tool. You can access the Gradle main site[65]. The Gradle user guide[66] is also available. Gradle is the third generation build tool for Java after Ant[67] and Maven[68] and has inside all the accumulated experience from its predecessors. As an example of its importance, Gradle was chosen by Google as the de facto build tool for Android projects.

We try to enumerate here just a part of the advantages that come with Gradle

- All Gradle plugins[69] are now available for you and can now be used in a variety of aspects of building
- The main scripting language in Grails is (still) Groovy, the same scripting language used in Grails. In this way, it will be simple for you to improve your build system whenever you like.
- Gradle is now offering the possibility to create *multi projects builds* for Grails 3 and this feature was impossible in the past

Gorm

Starting with Grails 3 the database persistence part of the framework named Gorm[70] was extracted completely outside the framework by the Grails team and was redesigned to be a wrapper over both Sql and NoSql databases. Also, starting with Gorm 4, this persistence framework can be used outside of Grails applications (in other Spring Boot applications for example).

In the new Gorm with Hibernate integration (which was the default persistence mechanism in Grails 1 and 2), you can still have support to the main SQL databases on market: MySql[71], Oracle[72], Postgresql[73], MsSql Server[74], etc. Also the framework is now offering support for other

[62] https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters
[63] https://projects.spring.io/spring-boot
[64] https://gant.github.io/
[65] https://gradle.org
[66] https://docs.gradle.org/current/userguide/userguide.html
[67] http://ant.apache.org
[68] https://maven.apache.org
[69] https://plugins.gradle.org
[70] http://gorm.grails.org/latest
[71] https://www.mysql.com
[72] https://www.oracle.com/database/index.html
[73] https://www.postgresql.org
[74] https://www.microsoft.com/en-us/sql-server

persistence technologies like the NoSql databases: MongoDB[75], Neo4j[76], Cassandra[77], Redis[78]. And the trend of Gorm is to use non-blocking drivers and to offer as reactive an approach as possible.

LogBack

LogBack is the most modern logging framework successor of Log4j and it comes to Grails 3 because was chosen as the logging framework for Spring Boot. You can access the LogBack main site[79].

Important aspects of Grails

Plugins

Plugins were a main feature of Grails from the beginning of the framework. Here is the list of Grails 3 plugins[80]. Even the framework itself in his core is a bunch of essential internal plugins. In fact, Grails is an aggregator for its plugins. If you look in a web profile of a Grails application you can see the default Grails plugins that are declared for this profile (will be different plugins for different profiles)

```
dependencies {
    compile "org.springframework.boot:spring-boot-starter-logging"
    compile "org.springframework.boot:spring-boot-autoconfigure"
    compile "org.grails:grails-core"
    compile "org.springframework.boot:spring-boot-starter-actuator"
    compile "org.springframework.boot:spring-boot-starter-tomcat"
    compile "org.grails:grails-dependencies"
    compile "org.grails:grails-web-boot"
    compile "org.grails.plugins:cache"
    compile "org.grails.plugins:scaffolding"
    compile "org.grails.plugins:hibernate5"
    compile "org.hibernate:hibernate-core:5.1.2.Final"
    compile "org.hibernate:hibernate-ehcache:5.1.2.Final"
    console "org.grails:grails-console"
    profile "org.grails.profiles:web"
    runtime "com.bertramlabs.plugins:asset-pipeline-grails:2.11.6"
    runtime "com.h2database:h2"
    testCompile "org.grails:grails-plugin-testing"
    testCompile "org.grails.plugins:geb"
```

[75] https://www.mongodb.com
[76] https://neo4j.com
[77] http://cassandra.apache.org
[78] http://redis.io
[79] http://logback.qos.ch
[80] https://grails.org/plugins.html

```
20      testRuntime "org.seleniumhq.selenium:selenium-htmlunit-driver:2.47.1"
21      testRuntime "net.sourceforge.htmlunit:htmlunit:2.18"
22  }
```

Here we try to enumerate some plugins and let you understand how many technologies can be accessible in such a simple way

- internal plugins
  - Hibernate plugin[81] is the plugin with the GORM implementation for Hibernate 5 persistence framework[82]
  - Asset Pipeline[83] it is responsible for rendering in an optimal way (unification in one file, compression, minification, and cache-digests) for static resource in Web pages; it is a replacement for the old Resources plugin[84] from Grails <2.x and there are other plugins[85] responsible for other aspects of serving resources: compile of CSS from LESS and SASS, trans-piling of Javascript from CoffeeScript
- external plugins
  - Spring Security Core Plugin[86] is the plugin responsible for the security of the Grails applications; it is based on Spring Security[87] library and it has a lot of companion plugins for other aspects of security - for example Spring Security REST Plugin[88] for REST API security
  - Quartz plugin for Grails[89] is the plugin responsible with scheduling jobs based on a well known Java library Quartz[90]; you can use it with another plugin Quartz Monitor Grails Plugin[91] responsible for monitoring the list of jobs
  - Mail plugin[92] is the plugin that can be used to send emails via a configurable SMTP server
  - Elastic Search plugin[93] will let you implement search for your site; your data is indexed using Elastic Search[94] library
  - Audit Logging plugin[95] can be used for auditing your application activity

About plugin's benefits

- should be declared in configuration files and are resolved as dependencies in a Maven, Ivy, Gradle style

[81] http://plugins.grails.org/plugin/hibernate5
[82] http://hibernate.org
[83] http://plugins.grails.org/plugin/asset-pipeline-grails
[84] https://grails.org/plugin/resources
[85] http://plugins.grails.org/q/asset%20pipeline
[86] http://plugins.grails.org/plugin/spring-security-core
[87] http://projects.spring.io/spring-security
[88] http://plugins.grails.org/plugin/spring-security-rest
[89] http://plugins.grails.org/plugin/quartz
[90] http://www.quartz-scheduler.org
[91] http://plugins.grails.org/plugin/org.grails.plugins:quartz-monitor
[92] http://plugins.grails.org/plugin/mail
[93] http://plugins.grails.org/plugin/elasticsearch
[94] https://www.elastic.co
[95] http://plugins.grails.org/plugin/audit-logging

- expose Grails constructs like *services*, *controllers*, *views*, *entities*, et cetera
- come with configurations that must be added to main app configuration file
- easy to integrate Java, Groovy and even client side JavaScript libraries into Grails applications

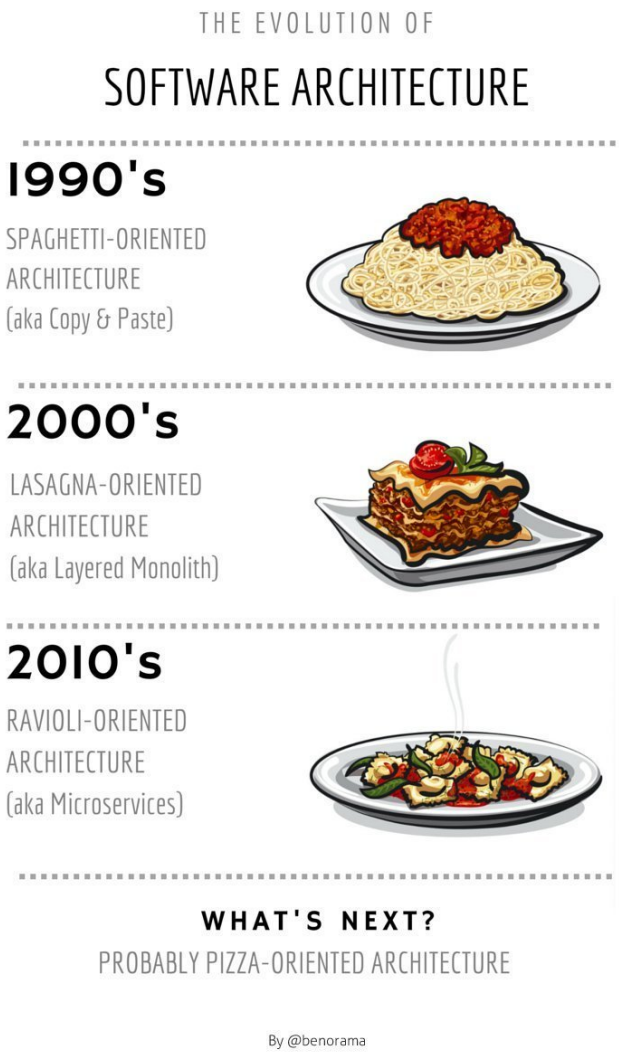
Profiles

Profiles were introduced in Grails 3 and now we can create different types of applications, not just the main type that was available till now, based on a server side HTML rendered interface using technologies like Gsp (based on a MVC framework on the server side). Now we can create a Grails 3 application that is just exposing a web service API without a classical server side rendered Web interface. Also, we can create a rich client directly from the server side - using scaffolding in the same way we have done till now for the classical type.

This is in accord with the global trend of IT industry these days which is trying to break the monolith application into small pieces, introducing the concept of Miroservices[96]. In these days, Java EE is creating a new Microprofile[97] for microservices in addition to their Full Profile and Web Profile.

You can see very clear the evolution of software architecture in the last decades in a tweet from Benoit Hediard[98]:

[96] http://martinfowler.com/articles/microservices.html
[97] http://microprofile.io
[98] https://twitter.com/benorama



Software architecture evolution

Another reason for this profiles movement is the need to expose just Web services and not a HTML based Web interface. And this is more valuable in the context of the mobile application and the *mobile first*[99] approach in developing applications. That's why a new set of server side rendering technologies were introduced in Grails 3 like Grails Views[100] that are rendering JSON and XML from domain objects.

Here you have the complete list of Grails 3 profiles[101]. Here is the Git official repository for Grails 3 profiles[102].

These are the profiles introduced in Grails 3

- angular - A profile for creating applications using AngularJS
- rest-api - Profile for REST API applications

[99] http://blogs.atlassian.com/2012/01/modern-principles-in-web-development
[100] http://views.grails.org/latest/
[101] https://github.com/grails-profiles
[102] https://github.com/grails/grails-profile-repository/tree/master/profiles

- base - The base profile extended by other profiles
- angular2 - A profile for creating Grails applications with Angular 2
- plugin - Profile for plugins designed to work across all profiles
- profile - A profile for creating new Grails profiles
- react - A profile for creating Grails applications with a React frontend
- rest-api-plugin - Profile for REST API plugins
- web - Profile for Web applications
- web-plugin - Profile for Plugins designed for Web applications
- webpack - A profile for creating applications with node-based frontends using webpack

Some details about the most important profiles that will be used in the application built in this book

- web - is a profile for creating classical Grails applications with a server side MVC framework in the same way it was in Grails 1 and 2; the presentation web interface is rendered on the server side
- rest-api - is a profile for creating applications that are exposing REST APIs; don't have a Web presentation and no MVC is needed
- angular(2) - is a profile for creating applications with a rich client based on AngularJS[103](1 or 2 versions) Javascript framework where MVC is located on client side (JavaScript) and the communication with the server side is via REST

[103] https://angularjs.org