OBJECT
COMPUTING

WORKSHOP

September 2021

Grails Deep Dive

Presented by Zachary Klein, Senior Software Engineer

OCI Engineering Training

- 24+ years experience
- Over 50,000 trained
- 150 current courses
- More than 40 instructors on staff
- All training delivered by practitioners and SME's in their respective fields
- Customized to fit your specific needs
- Flexible training delivery
- Training assessments

OCI
12140 Woodcrest Exec. Dr., Ste. 300
Saint Louis, MO 63141 USA

Limits of Liability and Disclaimer of Warranty

While every precaution has been taken in preparing this material, including research, development and testing, OCI assumes no responsibility for errors or omissions. No liability is assumed by OCI for any damages resulting from the use of this information.

Zachary Klein

- Senior Software Engineer at OCI
- JVM and Full Stack developer since 2010
- OSS contributor
- 2GM Team Member (Groovy, Grails, Micronaut)
- Email: kleinz@objectcomputing.com
- Twitter: @ZacharyAKlein

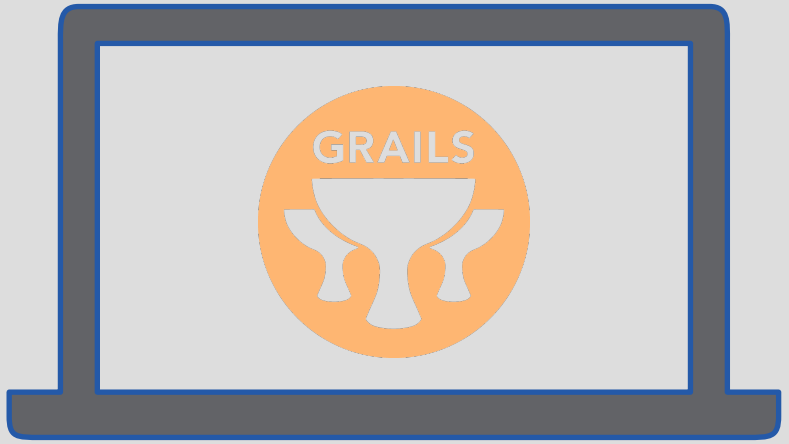
OCI | WE ARE SOFTWARE ENGINEERS.

Agenda

- Part 1: **Core Grails**: Grails (controllers, services, DI, configuration), Groovy, & Gradle
- Part 2: **APIs & Data**: GSPs, JSON views, Testing & GORM
- Part 3: **Advanced Grails**: GORM Data Services, Plugins, Spring, Security
- Part 4: **Future Grails**: Grails Upgrades, Micronaut, Groovy 3, What's New in Grails 4 & 5?

Workshop Format

- Lecture / Live-coding Combination
- Incrementally build a real-world Grails application!
- All sample code will be available on Github.com
- **Show your work!**
- Ask questions via Chat and/or Q&A Panel
- Email: kleinz@objectcomputing.com

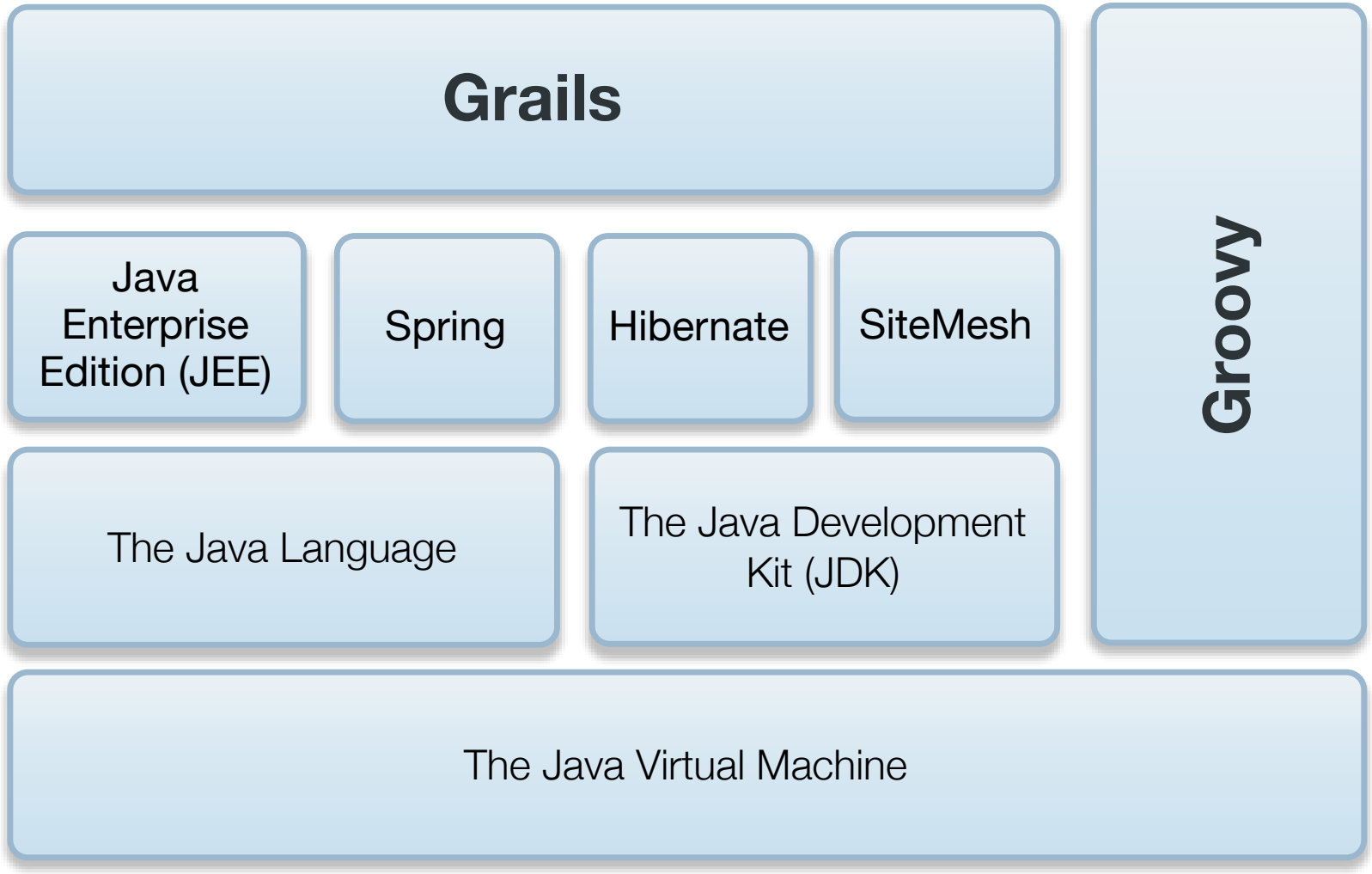


GRAILS®
DEEP DIVE

Introduction to Grails

- Full-stack web framework built on top of Spring Boot
- Embraces Convention-over-Configuration
- Designed to enhance developer-productivity
- Provides powerful Groovy-based persistence with GORM
- Leverages tried-and-true frameworks/libraries
- Extensible via plugins and configuration

The Grails Stack



Why Groovy And Grails?

- All the power of frameworks from other platforms like Ruby on Rails, Django and TurboGears
- Designed by JVM Developers for JVM Developers
- Built from the ground up on proven JVM technologies:
  - Spring Boot
  - Hibernate
  - Micronaut
  - Reactor
  - Spock
  - and more…

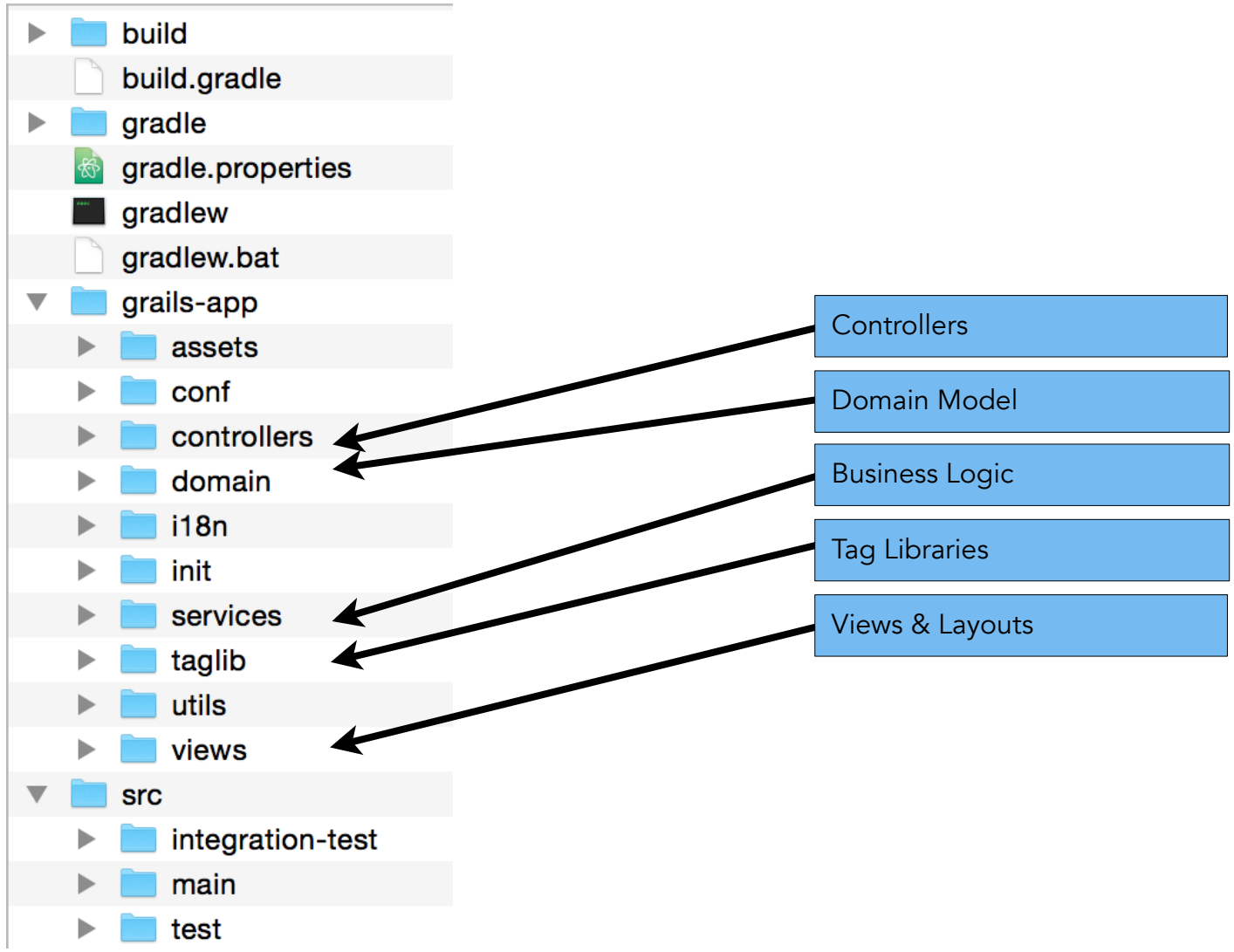
More than Just a Web Framework

- Grails delivers more than your regular web framework
  - "Full Stack!"
- Grails aims to ease development of every tier and features
  - An integrated Groovy build system
  - An incredibly simple ORM layer built on Hibernate
  - An amazing Groovy-based view technology called GSP
  - Domain specific languages for validation, web flows, querying and URL re-writing
  - An extensible plug-in system build on Spring

Getting Started

- SDKMAN
  - http://sdkman.io
- Manual Setup
  - Grails can be downloaded from http://grails.org
  - Simple installation:
    - Download and extract archive
    - Set GRAILS_HOME environment variable to location
    - Add $GRAILS_HOME/bin to PATH variable
- Run `grails create-app` to get started

Grails Project Structure

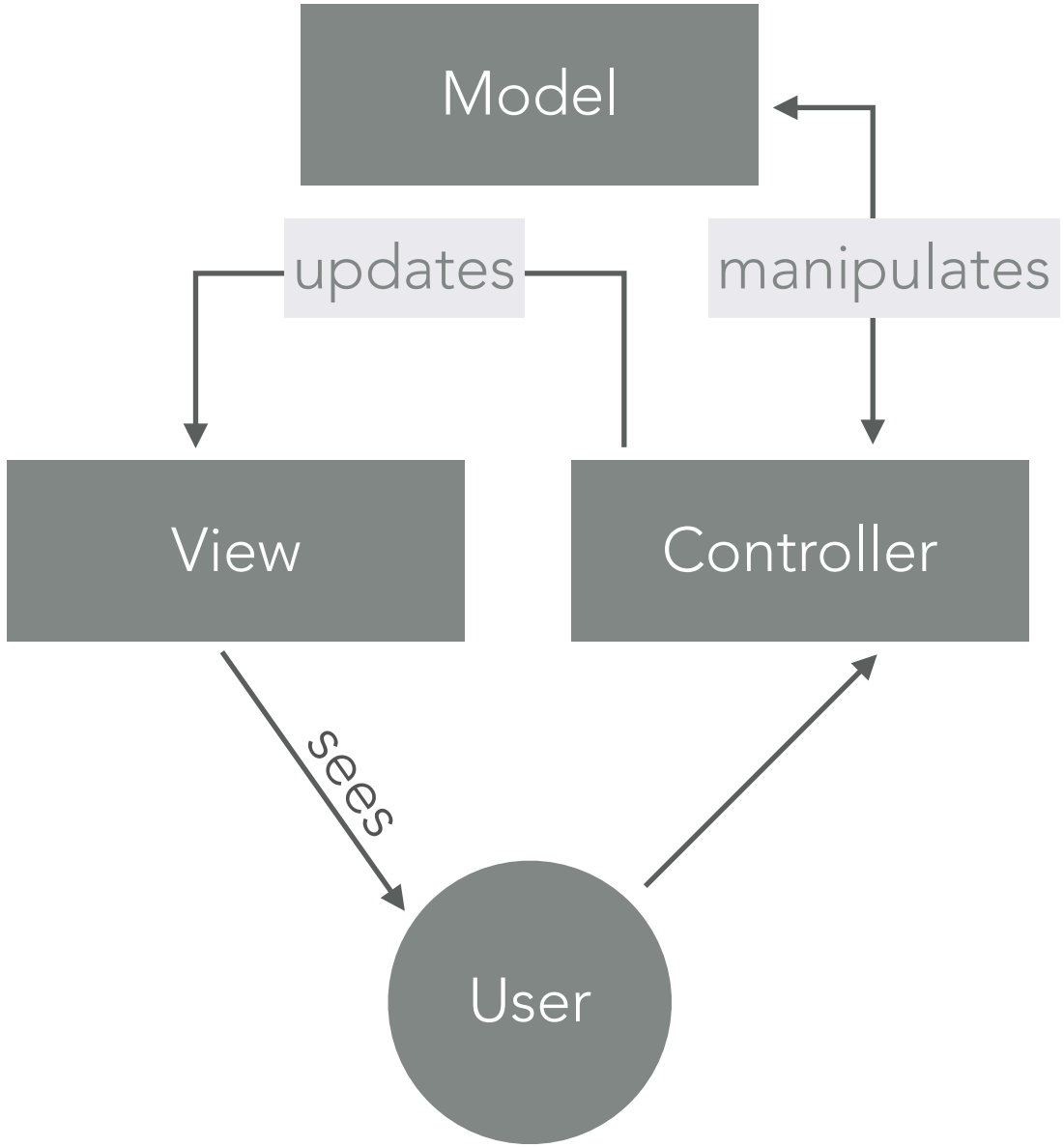


Sensible Defaults

- An in-memory H2 database
- Embedded Tomcat servlet container
- The ability to generate a deployable artifact (JAR or WAR file) out of the box
- A built-in interactive console and shell
- Fully customizable Gradle project

The MVC Pattern

- Grails implements the **Model / View / Controller** pattern
- **Model:** Represented by Grails domain classes, these represent the problem domain
- **View:** Typically defined in Groovy Server Pages (GSP) or JSON Views (RESTful APIs)
- **Controllers:** Serve incoming requests, interact with the model (usually via services) and render views



Spock

- Testing framework for JVM applications
- Built on top of JUnit
- Promotes expressive, behavior-focused testing
- Uses a Groovy-based DSL to provide expressive testing conventions
- Included in Grails for Unit and Integration tests
- http://spockframework.org

Summary

- Incredibly simple to get started with Grails!
- Database, web server and build environment included out of the box - Java web development doesn't get much easier!
- Built upon trusted frameworks and technology (Spring, Hibernate, Logback, Micronaut, etc)
- The Grails project infrastructure provides sensible defaults that remain the same from project to project

Workshop: Install & Run Grails!

Introduction to Groovy

- **Dynamic Language** for the JVM
- **Style-Agnostic**: OOP, Functional, Scripts, DSLs
- **Static** *and* **Dynamic** Type-checking
- **Compiles** to Java Bytecode
- Development **sponsored** by Object Computing, Inc
- Top-level Project of the **Apache Software Foundation**

A multi-faceted language for the Java platform

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 to your application powerful features, including scripting capabilities, **Domain-Specific Language** authoring, runtime and compile-time **meta-programming** and **functional** programming.

Flat learning curve

Concise, readable and expressive syntax, easy to learn for Java developers

Smooth Java integration

Seamlessly and transparently integrates and interoperates with Java and any third-party libraries

Vibrant and rich ecosystem

Web development, reactive applications, concurrency / asynchronous / parallelism library, test frameworks, build tools, code analysis, GUI building

Powerful features

Closures, builders, runtime & compile-time meta-programming, functional programming, type inference, and static compilation

Domain-Specific Languages

Flexible & malleable syntax, advanced integration & customization mechanisms, to integrate readable business rules in your applications

Scripting and testing glue

Great for writing concise and maintainable tests, and for all your build and automation tasks

Why Use Groovy?

- **Power**
  - Advanced programming & productivity-enhancing features
- **Versatility**
  - Can be used for full programs to scripts; OOP or functional style; testing; build scripts; “glue code”
- **Simplicity**
  - Flat learning curve, expressive & readable code

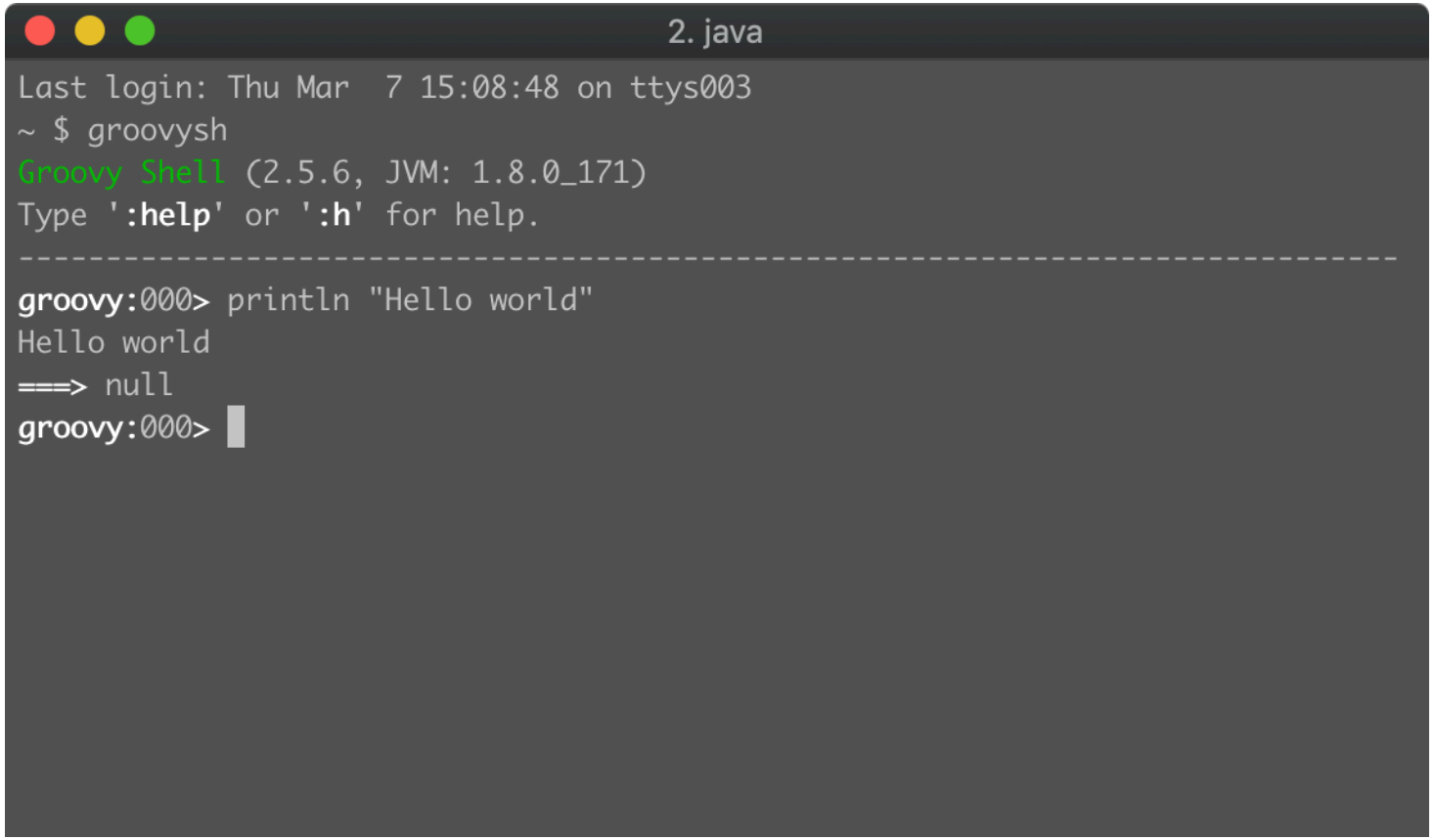
Comparing Java vs Groovy


```java
// PrintIndependenceDay.java

import java.util.Calendar;
import java.util.Date;

public class PrintIndependenceDay {

    public static void main(String[] args) {
        Calendar calendar = Calendar.getInstance();
        calendar.clear();
        calendar.set(Calendar.MONTH, Calendar.JULY);
        calendar.set(Calendar.DATE, 4);
        calendar.set(Calendar.YEAR, 1776);

        Date time = calendar.getTime();

        System.out.println(time);
    }
}
```

```groovy
// PrintIndependenceDay.groovy

def calendar = Calendar.instance
calendar.with {
    clear()
    set MONTH, JULY
    set DATE, 4
    set YEAR, 1776
    println time
}
```

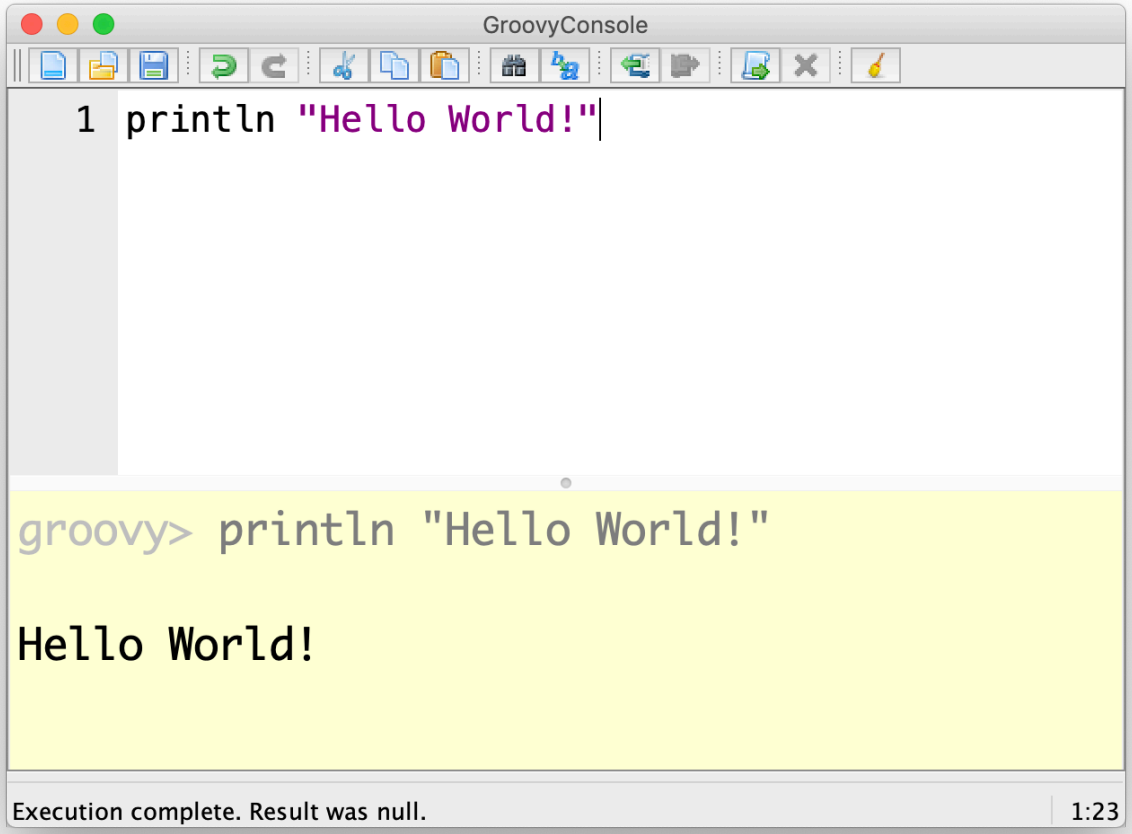
Playing with Groovy

- **groovysh** - Groovy Shell



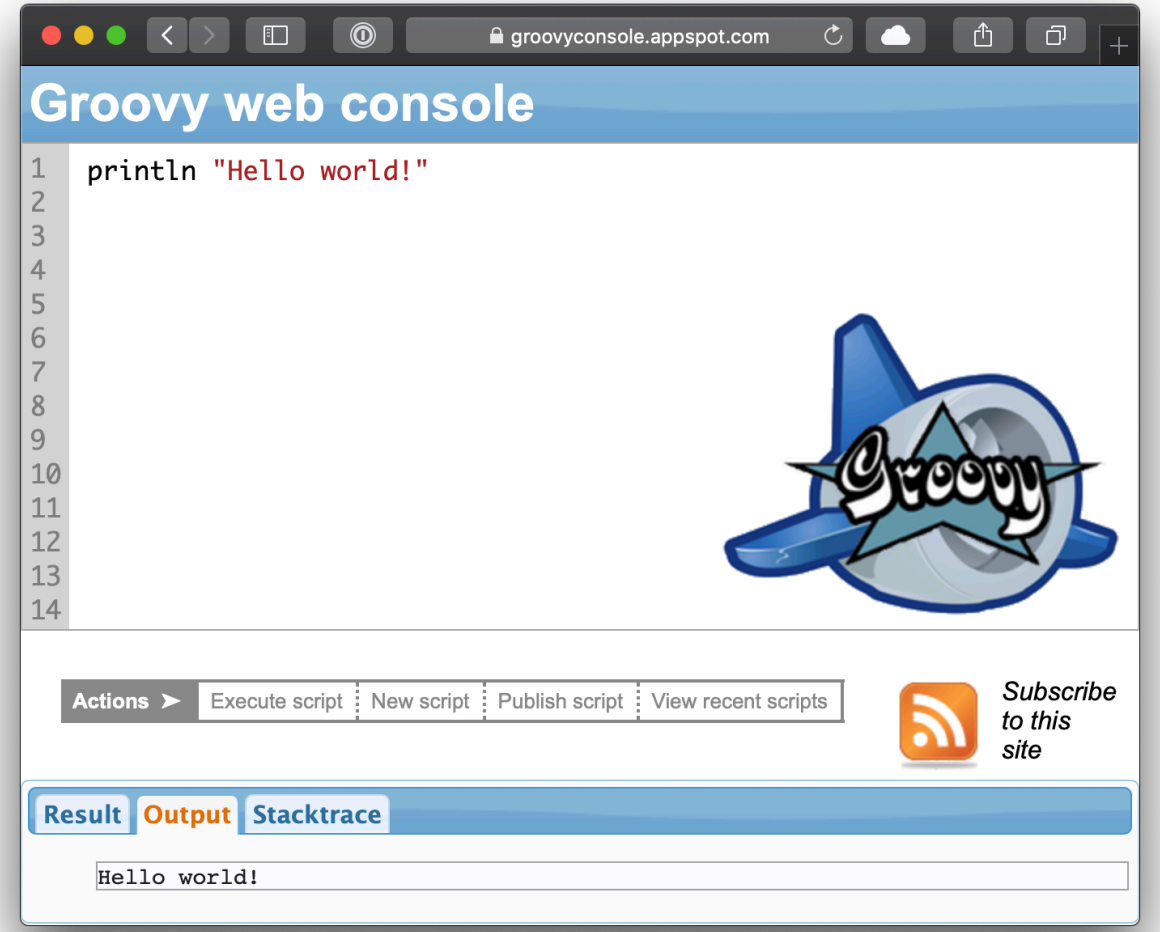
Playing with Groovy

- **groovyConsole** - Groovy Console



Playing with Groovy

- **Groovy Web Console** - groovyconsole.appspot.com



Workshop: Intro to Groovy

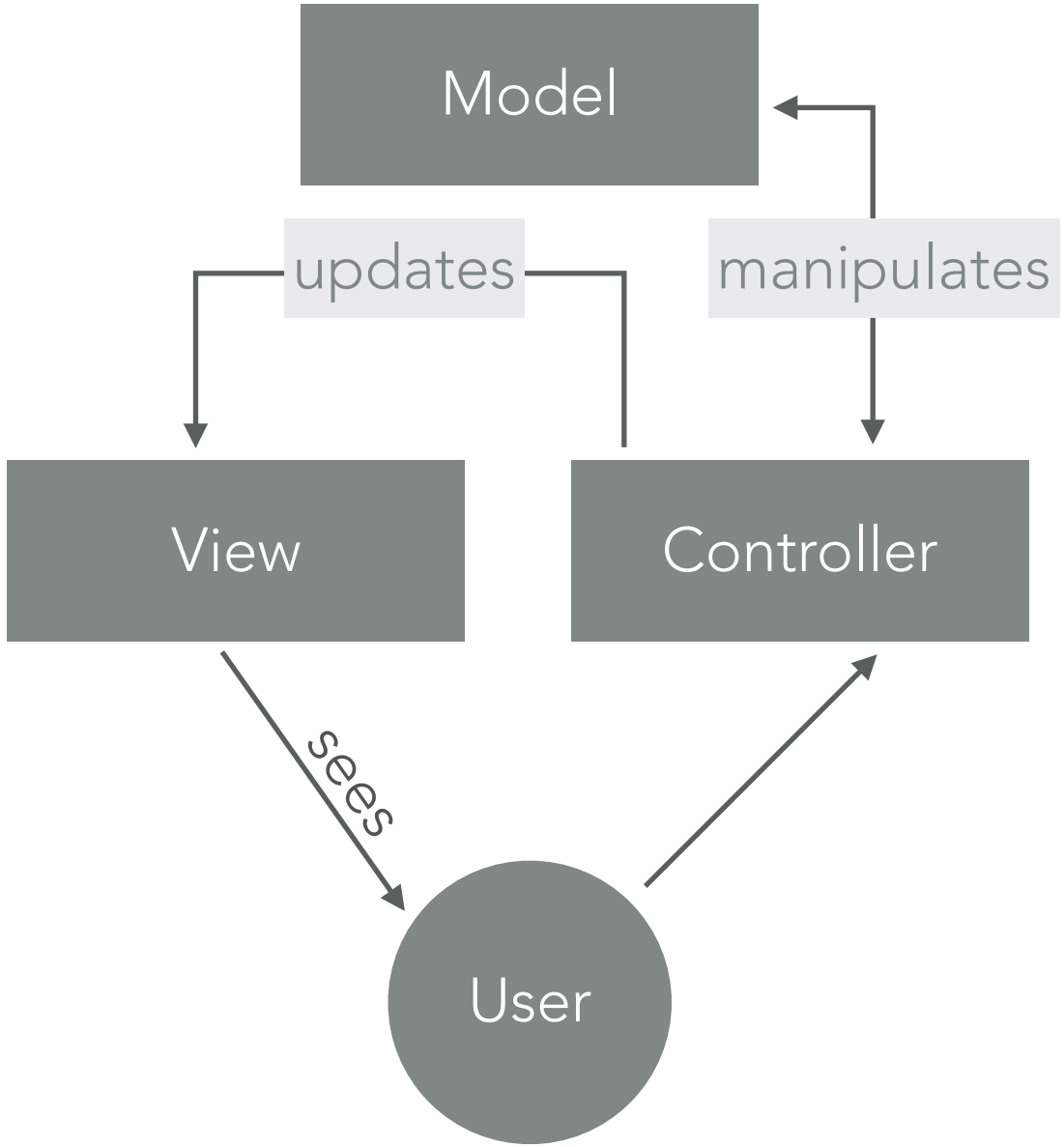
Introduction to Gradle

- Grails projects are built using Gradle
- Full-featured build tool and task runner
- First-class support for Groovy projects, including a dedicated plugin
- Uses a Groovy DSL for build scripting (a Kotlin DSL is also available)
- https://gradle.org/

Workshop: Intro to Gradle

Controllers: Easy MVC

- Comprehensive MVC Framework Built On Spring
- Data Binding (aka Command Objects)
- Convention over Configuration
- URL Mappings (routes)
- Groovy Server Pages (GSPs)
- Custom Tag Libraries
- Cross Origins Resource Sourcing (CORS)
- I18n Internationalization
- etc...



MVC Terms

- Model - a map returned by an action to be rendered by a view
- View - a Groovy Server Pages (GSP) view located in the `grails-app/views` directory
- Controller - a class ending with the convention "Controller" located in the `grails-app/controllers` directory
  - A controller in turn has many **actions,** which are public methods defined in the controller

Respond, Redirect, Render, Return


```
class BookController {
    // write some text
    def hello() { render "hello!" }

    // return a model to the view called "list"
    def list() {
       [ books : Book.list() ]
    }

    // perform redirection
    def index() { redirect(action:list) }

    // output some XML
    def bookXml() {
        render(contentType:"text/xml") {
            book(title: Book.get(params.id)?.title)
        }
    }
}
```

Render simple text based responses

Return a model to be rendered by the view

Redirect control to another action

Render XML or JSON based responses

"respond" will be covered later

Respond & Content-Negotiation


```
def list() {
    def people = Person.list()

    withFormat {
        html { [personList: people] }
        xml { render people as XML }
        json { render people as JSON }
    }
}

def show(Person personInstance) {
    respond personInstance
}
```

Preferred

Customizing The Default Renderers


```
// grails-app/conf/spring/resources.groovy
import grails.rest.render.json.JsonCollectionRenderer
import demo.Person

beans = {
    peopleJsonRenderer(JsonCollectionRenderer, Person) {
        excludes = ['age']
    }
}
```

```
curl -i http://localhost:8080/restconfdemo/people.json

[{“class":"demo.Person","id":1,"firstName":"Jake","lastName":"Brown"},
 {“class":"demo.Person","id":2,"firstName":"Zack","lastName":"Brown"},
 {"class":"demo.Person","id":3,"firstName":"Jeff","lastName":"Brown"}]
```

Mapping URLs To Controllers


```
class UrlMappings {
    static mappings = {
      "/$controller/$action?/$id?"{
          constraints {
             // apply constraints here
          }
      }
      "500"(view:'/error')
    }
}
```

Mapping URLs To Controllers


```
class UrlMappings {

    static mappings = {
        "/$controller/$action?/$id?(.$format)?"{
            constraints {
                // apply constraints here
            }
        }

        "/people"(controller: 'person', action: 'index')

        "/youngerThan/$maxAge"(controller: 'person', action: 'search') {
            constraints {
                maxAge range: 5..125
            }
        }

        // ...
    }
}
```

Controller Action Allowed Methods


```
def update(Person personInstance) {
    if(request.method != 'PUT') {
        response.sendError 405
    } else {
      // ...
    }
}
```

Don't Need To Do This If Using

```
static allowedMethods = [save: "POST", update: "PUT", delete: "DELETE"]
```

Data Binding & Command Objects


```
// You write this...
def someControllerAction(Widget w) {
    // ...
}

// The compiler generates this (PSEUDO Code)...
def someControllerAction() {
    Widget w = new Widget()
    bindData(w, params)

    // DI code omitted here...

    w.validate()
    someControllerAction(w)
}
```

Data Binding And Domain Class Command Objects


```
def someControllerAction(Person per) {
    // ...
}
```

```
// PSEUDO Code...
def someControllerAction() {
    Person p
    if(params.id) {
        p = Person.get(params.id)
    } else if(request.method == 'POST') {
       p = new Person()
    }
    def doBinding = true
    if(params.id) {
        if(!(request.method in ['PATCH', 'POST', 'PUT'])) {
            doBinding = false
        }
    }
    if(doBinding) {
        bindData(p, params)
    }
    // DI code omitted here...
    p.validate()
    someControllerAction(p)
}
```

Namespaced Controllers


```
// grails-app/controllers/demo/first/DemoController.groovy
package demo.first

class DemoController {

    static namespace = ‘alpha'

    // …
}
```

```
// grails-app/controllers/demo/second/DemoController.groovy
package demo.second

class DemoController {

    static namespace = ‘beta'

    // …
}
```

Workshop: Controllers

JSON Views

- Extensible View Library For Grails
  - JSON Views
  - Markup Views

- Support for Hypermedia (HAL JSON, JSON API)
- Installed by default when using the **rest-api** profile

JSON Views


```
//grails-app/views/greeting/hello.gson
json.message {
    hello 'world'
}

{"message": {"hello":"world"}}
```

JSON Views


```
import demo.Person

model {
    Person person
}

json g.render(person)
```

Workshop: JSON Views

Testing in Grails

- Grails includes first-class support for testing
- Types of tests in Grails:
  - **Unit tests** - testing individual classes (including controllers, services, domain classes, taglibs, etc)
  - **Integration tests** - test running Grails application
  - **Functional tests** - browser-based interaction testing (built on Geb: https://gebish.org)

Spock

- Testing framework for JVM applications
- Built on top of JUnit
- Promotes expressive, behavior-focused testing
- Uses a Groovy-based DSL to provide expressive testing conventions
- Included in Grails for Unit and Integration tests
- http://spockframework.org

Grails Testing Framework

- Support for writing concise expressive tests for Grails artifacts
- Based on Groovy Traits
- Allows for customization of Grails Application Context, Spring configuration, and mocking of DI within unit tests
- First-class support for Grails controllers, domain classes, services, and taglibs
- https://testing.grails.org/

GORM - ORM Reinvented

- Grails provides a “convention-over-configuration” Hibernate configuration strategy
- Uses the convention in the class names, property names and property types to provide mapping
- All this information is already there in the JVM!
- New strategy is called **GORM**

Standard Domain Class


```
1 package demo
2
3 class Person {
4     String firstName
5     String lastName
6     Integer age
7 }
```

`person` table

id (int)	first_name (varchar)	last_name (varchar)	age (int)
1	John	Doe	40
2	Sally	Due	35

hasMany Mapping

- `hasMany` is a map of one-to-many relationships. Example:

```
static hasMany = [comments:Comment]
```

- A property of type `java.util.Set` is automatically created

```
bookmark.comments?.each { println it.text }
```

belongsTo Mapping

- The `belongsTo` mapping is important in that it states:
  - "I belong to X and therefore when X is deleted so am I"

```
class Comment {
   ...
   static belongsTo = Bookmark
}
```

Lists, Sets & Maps

- GORM supports mappings of Lists and Maps using an index column

```
class Bookmark {
   List comments
   static hasMany =
      [comments:Comment]
}
```

```
class Bookmark {
   Map commentsByUser
   static hasMany =
     [commentsByUser:Comment]
}
```

- Default behavior is to assume a collection is a Set, but you can use SortedSet too!

Components

- Grails supports the concept of components where an associated class maps into the same table

```
class Bookmark {
   ...
   Note desc
   static embedded = ['desc']
}
class Note {
  Date lastModified
  String text
}
```

bookmark table

id	...	desc_text	desc_last_modifi
16	...	some Grails site	6th of May 2007
17	...	A great Groovy article!	6th of May 2007

Advanced Mapping

- GORM features a ORM mapping DSL for more advanced mappings
- With access to features like:
  - Optimistic locking control
  - Second level cache Config
  - Inheritance strategy (table-per-subclass?)
  - Id generation strategy (sequence, native ..)
  - Composite Ids
  - Eager/Lazy fetching
  - Table/Column name & type mappings

Mapping DSL Example


```
class Person {
    String firstName
    static hasMany = [children:Relative]
    static mapping = {
        table 'people'
        version false
        firstName column: 'f_name'
   }
}
```

Relationship Management

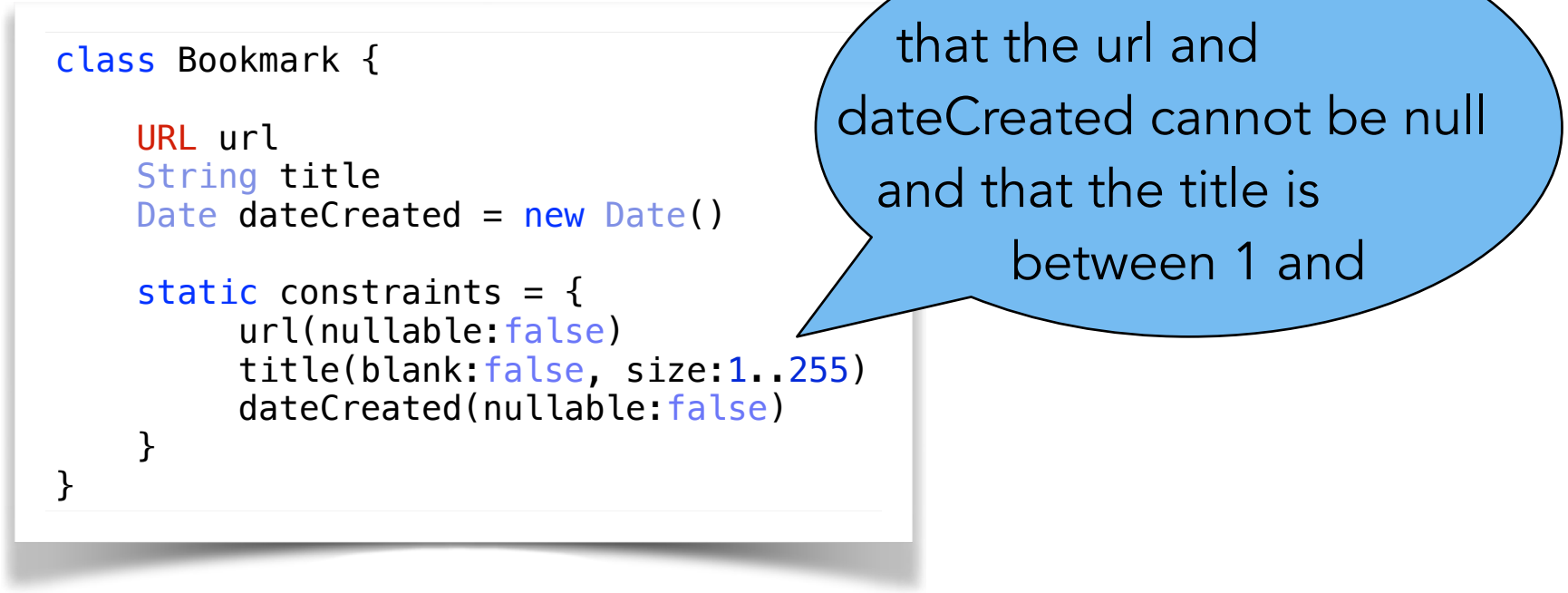
- GORM automatically adds handy relationship management methods

```
new Bookmark(url:"http://grails.org")
      .addToComments(text:"Great!", leftBy:"Joe")
      .addToComments(text:"Groovy!", leftBy:"Fred")
      .save()

// and later
bookmark.removeFromComments(myComment)
```

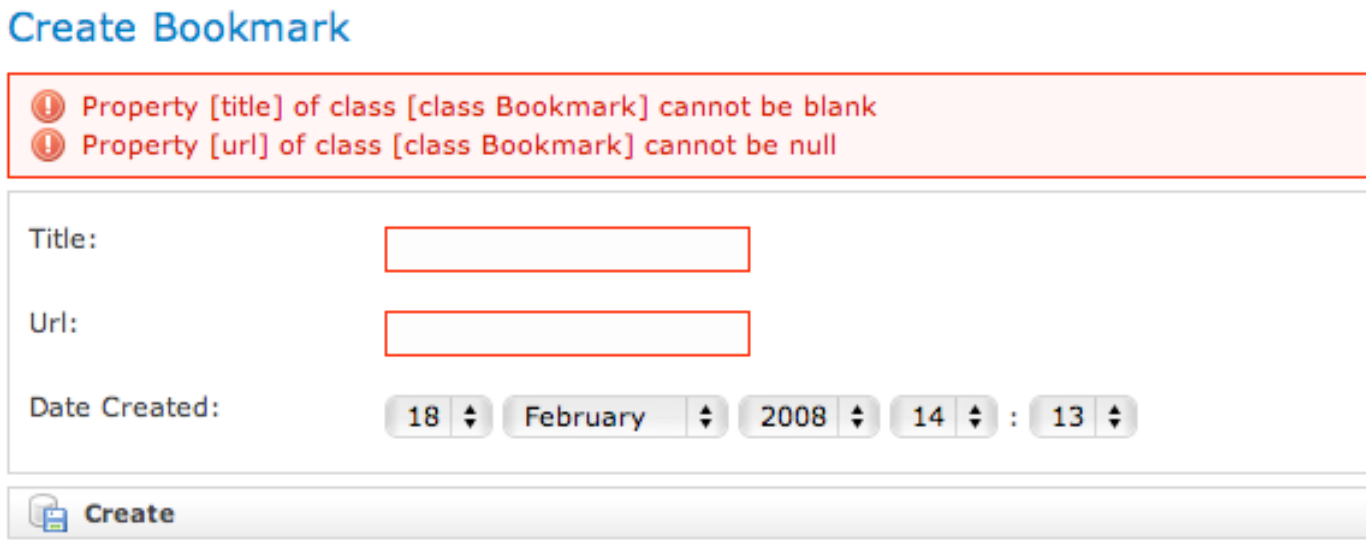
Domain Constraints

- Constraints are settings applied against properties of the domain
- They allow:
  - Additional declarative validation
  - Control of database schema generation
  - Control of scaffolded view generation (CRUD)
- Constraints are defined with a Groovy builder

Constraints in Action



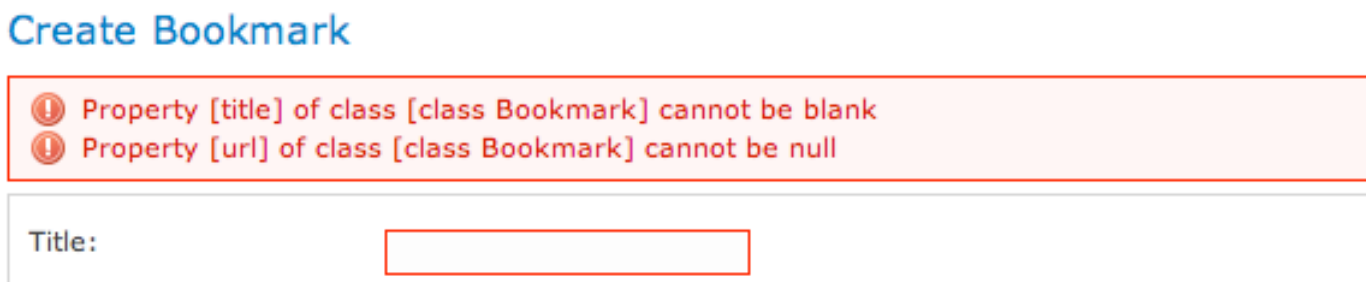
Automatic Validation



Message Codes

- If the error messages aren't to your liking change them in the `grails-app/i18n/message.properties` bundle

```
bookmark.title.blank=Title must be specified!
bookmark.title.length.tooshort=Your title is too short, it must be at
least 1 character!
bookmark.url.nullable=The URL must be specified, or is not a a valid URL
```



Programmatic Validation

Automatic Databinding

```
class BookmarkController {
    def save() {
        def b = new Bookmark(params)
        if(b.save()) {
            redirect(action:show, id:b.id)
        }
        else {
            render(view:"create",
                   model:[bookmark:b])
        }
    }
}
```

CRUD Operations

- Domain classes are automatically provided with methods to perform CRUD
  - save
  - get
  - delete

```
// create
def b = new Bookmark(params)
b.save()

// read
b = Bookmark.get(27)

// update
b.title = "Groovy Homepage"
b.save()

// delete
b.delete()
```

Querying

- GORM supports a number of ways to query including:
  - Dynamic Finders
  - Criteria
  - Where
  - HQL

Dynamic Finders

- Automatically translate the properties of the class into "method expressions" - at runtime!
- Uses the Hibernate Criteria API underneath
- Rich and expressive way to query

Dynamic Finder Operators

- `InList` - In the list of given values
- `LessThan` - less than a given value
- `LessThanEquals` - less than or equal a give value
- `GreaterThan` - greater than a given value
- `GreaterThanEquals` - greater than or equal a given value
- `Like` - Equivalent to a SQL like expression
- `Ilike` - Similar to a `Like`, except case insensitive
- `NotEqual` - Negates equality
- `InRange` - Between the `from` and `to` values of a Groovy Range
- `Rlike` - Performs a Regexp LIKE in MySQL or Oracle otherwise falls back to `Like`
- `Between` - Between two values (requires two arguments)
- `IsNotNull` - Not a null value (doesn't take an argument)
- `IsNull` - Is a null value (doesn't take an argument)

Dynamic Finders


```
def all = Bookmark.list()

// user like expressions
def grailsBookmarks =
   Bookmark.findAllByTitleLike("%Grails%")

// query between two values
def now = new Date()
def lastWeeks =
Bookmark.findByCreatedDateBetween(now-7, now)

// query associations
def bookmark = Bookmark.get(34)
def comments =  Comment.findAllByBookmark(bookmark)
```

Querying with Criteria


```
// returns first 10 users who have an active
// account that has been created in the last
// 30 days and that have Grails-like
// bookmarks created in the last 7 days
def now = new Date()
def users = Bookmark.withCriteria {
    comments {
        like("text","%Grails%")
        between("dateCreated", now-7, now)
    }
    between("dateCreated",now-30, now)
    maxResults(10)
}
```

Where Queries

- Use DetachedCriteria
- Compile time type checked
- Compatible with static compilation
- Use raw Groovy for criteria
- Are compose-able

Where Queries


```
def query = Person.where {
    age > 19
    homeAddress.town == 'St. Louis'
}
def results = query.list()
```

```
def adults = Person.where {
    age > 18
}

def adultsInStl = adults.where {
    address.town == 'St. Louis'
}
def results = adultsInStl.list()
```

Querying with HQL

- If all else fails, there is always HQL!:

```
// Query for Bookmark instances
def bookmarks =
 Bookmark.findAll("from Bookmark b where b.title like ?", ["%Grails%"]
)

// select only the Bookmark titles
def titles =
  Bookmark.executeQuery(
    "select b.title from Bookmark b where b.title like ?",
    ["%Groovy%"] )
```

Transaction Isolation

- Levels -
  - `org.springframework.transaction.annotation.Isolation`
  - **DEFAULT**
    - use default for the underlying driver
  - **READ_UNCOMMITTED**
    - dirty reads, non-repeatable reads, phantom reads
  - **READ_COMMITTED**
    - non repeatable reads, phantom reads
  - **REPEATABLE_READ**
    - phantom reads
  - **SERIALIZABLE**
    - no dirty reads, no non-repeatable reads, no phantom reads

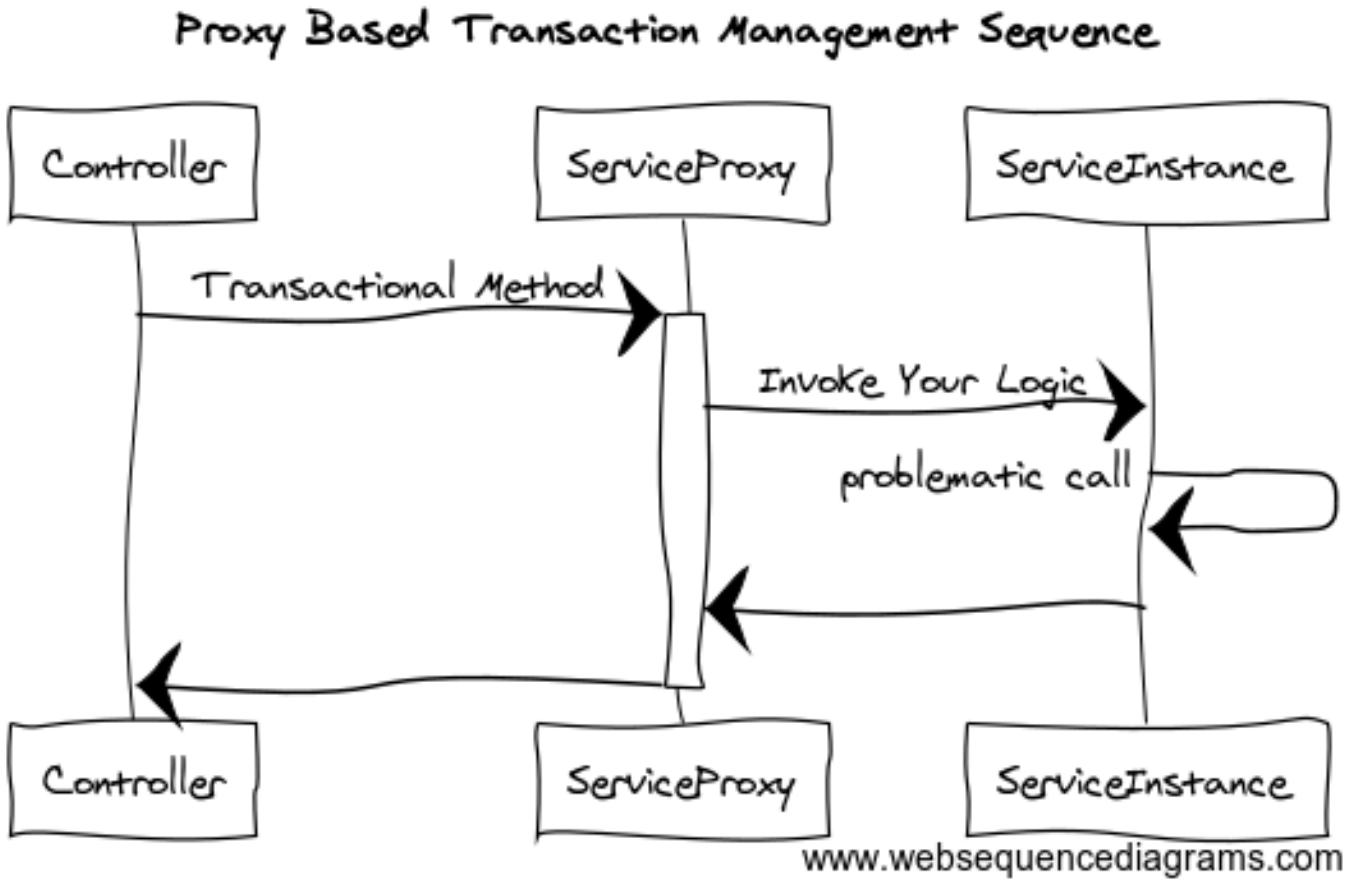
Isolation Levels

- Phantom Read
  - Same Read (Query) Returns Different Rows Within A Transaction
- Non-Repeatable Read
  - Same Row Contains Different Values Across Reads Within A Transaction
- Dirty Read
  - Read Returns Uncommitted Changes From Other Transactions

Transaction Propagation

- Levels
  - `org.springframework.transaction.annotation.Propagation`
  - **REQUIRED** - requires a transaction
  - **SUPPORTS** - join transaction if one is running
  - **MANDATORY** - throws exception if no transaction is running
  - **REQUIRES_NEW** - requires a new transaction
  - **NOT_SUPPORTED** - suspends current transaction
  - **NEVER** - throws exception if transaction is running
  - **NESTED** - executes within a nested transaction

Transactional Proxy



Transactions

- 2 Methods to control transactions in Grails:
  - `@Transactional` annotation on the class or method
  - Programmatically with the `withTransaction` method

Using withTransaction


```
// Alternatively programmatic transaction
// management can be done using
// withTransaction which takes a closure
// The status object is a Spring
// TransactionStatus instance
Bookmark.withTransaction { status ->
  def b = Bookmark.get(10)
  b.url = new URL("http://grails.org")

  // something goes horribly wrong
  status.setRollbackOnly()
}
```

Transactional Annotation


```
import grails.gorm.transactions.Transactional

@Transactional
class BookmarksService {

    def someServiceMethod() {
        // do some work...

        if(someCondition) {
            // transactionStatus implements
            // org.springframework.transaction.TransactionStatus
            transactionStatus.setRollbackOnly()
        }
    }
}
```

Workshop: GORM

GORM Data Services

- Introduced In GORM 6.1
- Auto-Generated Data Service Layer
- Less code to write
- Static type-checking
- Queries computed at compilation time
- Easy Mocking

Simple Data Service


```
import grails.gorm.services.Service

@Service(Book)
interface BookService {
    Book getBook(Serializable id)
}
```

That is it. Just an interface. No implementation needs to be written.

Data Service Injection


```
class BookController {

    BookService bookService

    def show(Long id) {
        Book book = bookService.getBook(id)

        // ...
    }
}
```

Just Works™!

GORM Data Services

- Supports abstract classes as well as interfaces
- All public methods are automatically made @Transactional
  - read-only for query methods
  - Can opt out with @NotTransactional

Abstract Data Service Class


```
import grails.gorm.services.Service

@Service(Book)
abstract class BookService  {

    protected abstract Book getBook(Serializable id)

    protected abstract Author getAuthor(Serializable id)

    Book updateBook(Serializable id, Serializable authorId) {
        Book book = getBook(id)
        if(book != null) {
            Author author = getAuthor(authorId)
            if(author == null) {
                throw new IllegalArgumentException("Author does not exist")
            }
            book.author = author
            book.save()
        }
        return book
    }
}
```

Query Methods

- count*, countBy*
- delete*
- find*, get*, list*, retrieve*
  - (all have *By* counterparts)
- save*, store*, persist*
- update*

Data Service Query Methods


```
import grails.gorm.services.Service

@Service(Book)
interface BookService {
    List<Book> findBooks(String title)
    List<Book> findBooks(String title, Map params)
    Book findBook(String title)
}
```

Data Service Dynamic Finder


```
import grails.gorm.services.Service

@Service(Book)
interface BookService {
    List<Book> findByTitleAndAuthor(String title,
                                    String writer)
}
```

Data Service Property Projection


```
import grails.gorm.services.Service

@Service(Book)
interface BookService {
    Date findBookReleaseDate(String title)
    List<String> findBookAuthor(String publisher)
}
```

Data Service Write Operations


```
import grails.gorm.services.Service

@Service(Book)
interface BookService {
    Book saveBook(String title)

    Book saveBook(Book newBook)

    void delete(Serializable id)

    // first argument must be id...
    Book updateBook(Serializable id, String title)
}
```

Data Service Where Queries


```
import grails.gorm.services.Service
import grails.gorm.services.Where

@Service(Book)
interface BookService {

    @Where({ title == title && releaseDate > date })
    void delete(String title, Date date)

    @Where({ author == authorName &&
             releaseDate > fromDate })
    List<Book> searchBooks(String authorName,
                           Date fromDate)
}
```

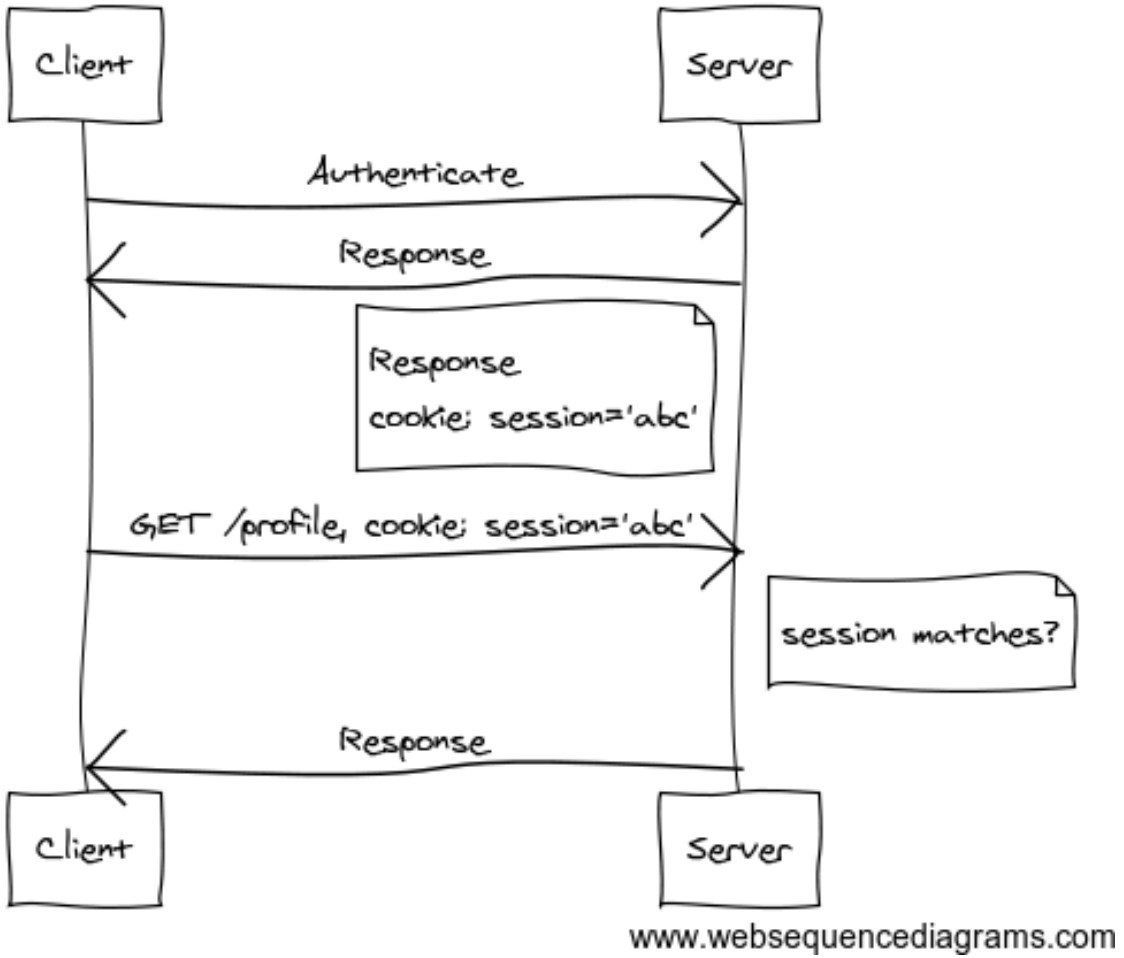
Workshop: GORM Data Services

Grails Spring Security Plugin

- Core Grails plugin that wraps Spring Security
- Simple to setup
- First-class integration with Grails Controllers and GORM
- Provides both config map and annotation-based authentication and authorization
- Extension plugins add support for OAuth2, LDAP, SAML, and more

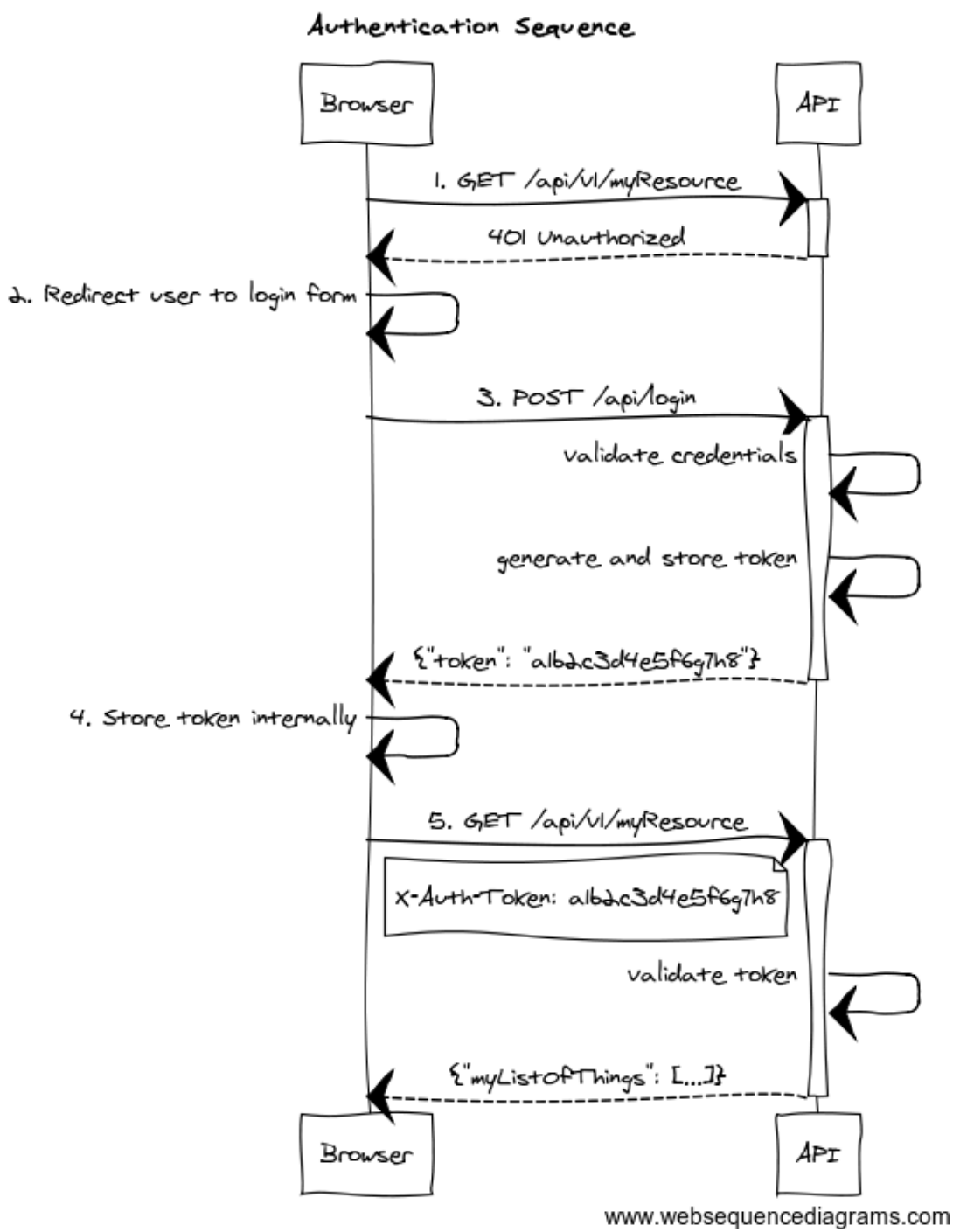
Session-Based Security

- Default model with Spring Security Core
- Session managed on the server
- Client uses a cookie containing the "jsessionID" that identifies the logged-in user
- Login/logout programmatically



Stateless Token-Based Security

- Supported via the Spring Security REST Plugin: https://grails-plugins.github.io/grails-spring-security-rest/latest/docs/
- Follows OAuth 2 conventions (can be extended to use OAuth providers like Twitter, Google, Github, etc)
- Grails Guides:
  - OAuth2 Twitter: https://guides.grails.org/grails-oauth-twitter/guide/index.html
  - OAuth2 Google: https://guides.grails.org/grails-oauth-google/guide/index.html

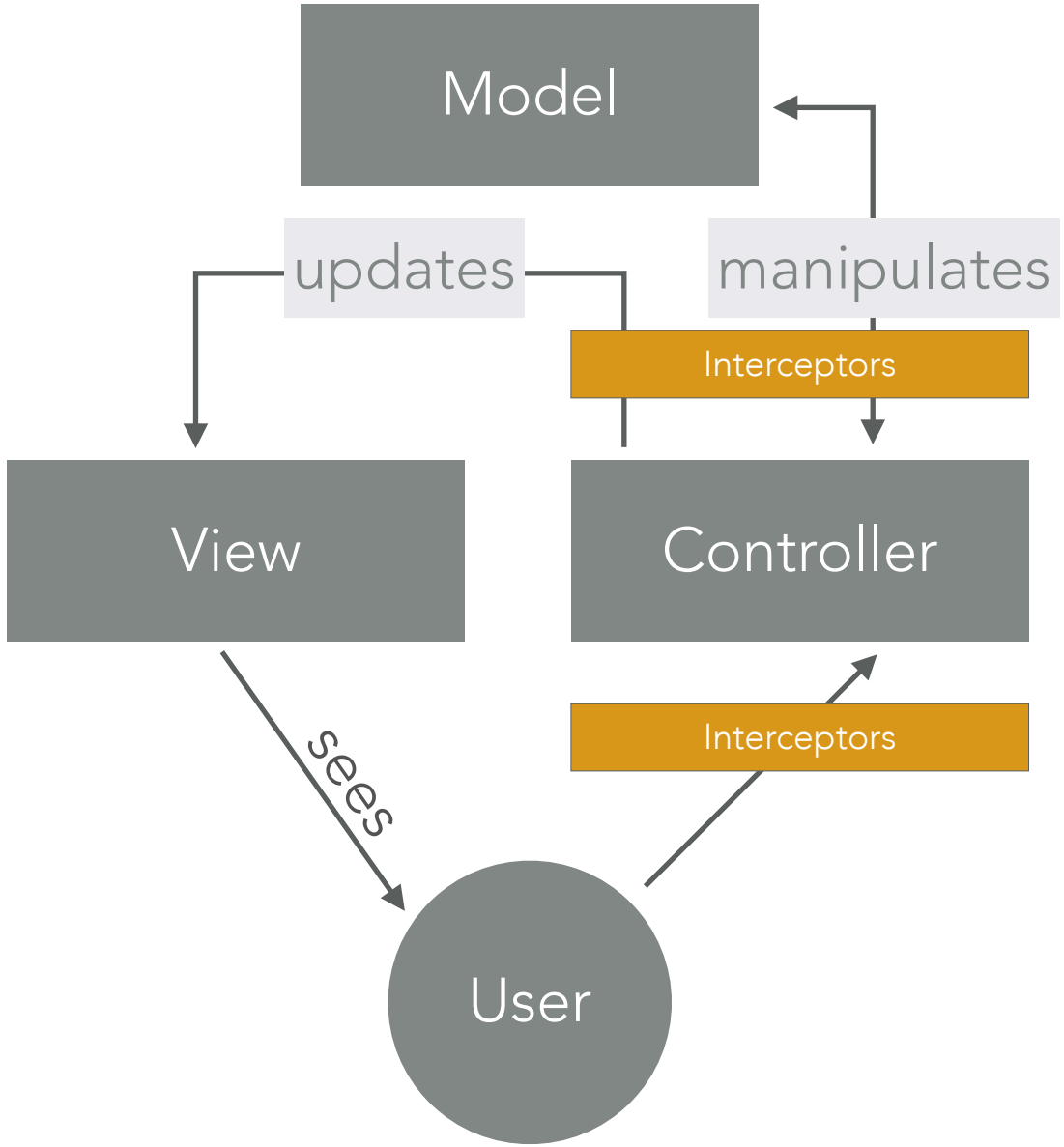


Workshop: Spring Security Plugin

Interceptors

- Interceptors allow you to add logic before/after a request
- Allow blocking or redirecting of a request
- By default, interceptors will match all requests made a controller of the same name
- Custom “matching” configured in the constructor

https://objectcomputing.com/resources/publications/sett/september-2015-grails-3-interceptors/



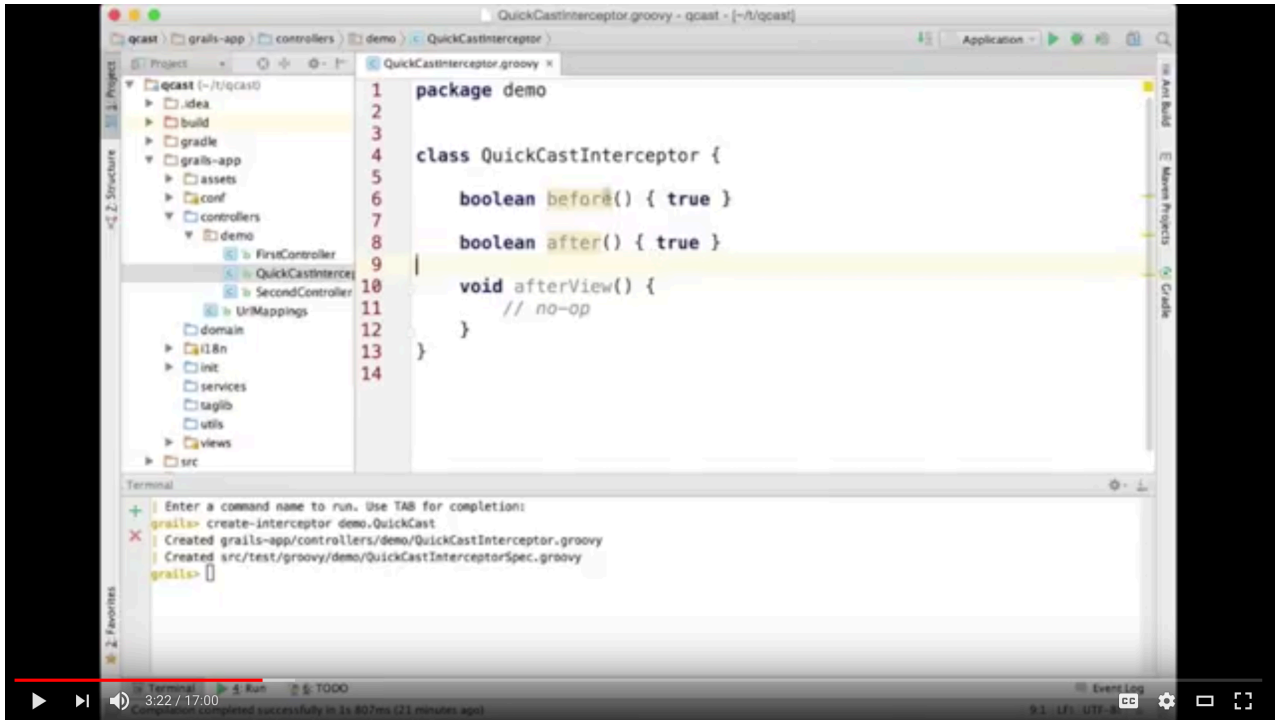
grails-app/controller/com/example/BookInterceptor.groovy

```groovy
package com.example

class BookInterceptor {

    boolean before() { true }

    boolean after() { true }

    void afterView() { }
}
```


```groovy
class CustomInterceptor {

    public CustomInterceptor() {
        // match all requests to the reporting controller...
        match controller: 'reporting'

        //match requests to create action, person controller
        match controller: 'person', action: 'create'

        //match all requests to accounting or payroll controller
        match controller: ~/(accounting|payroll)/
    }

    boolean before() { true }

    boolean after() { true }

    void afterView() { /.../ }
}
```


```groovy
class CustomInterceptor {

    int order = HIGHEST_PRECEDENCE + 100 //or LOWEST_PRECEDENCE

    boolean before() { true }

    boolean after() { true }

    void afterView() { /.../ }
}
```

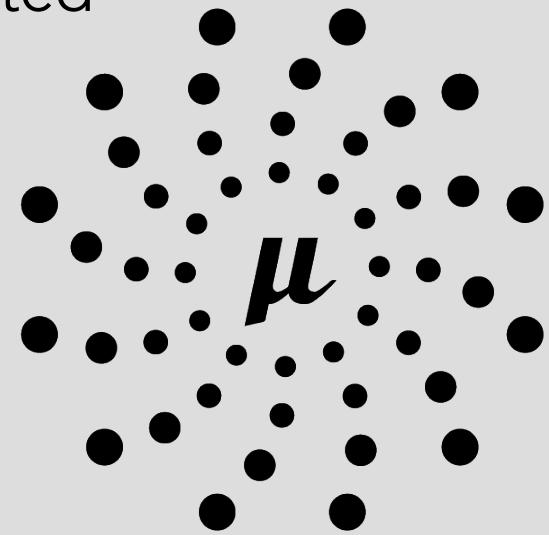
Grails Quickcast: Interceptors



https://www.youtube.com/watch?v=XnRNfDGkBVg

Grails & Micronaut Integration

- Micronaut is a new JVM framework for cloud-based applications, Dependency Injection, Aspect-Oriented Programming, and more
- Built by the same team that developed Grails
- Uses Ahead Of Time (AOT) compilation for high-performance and low memory usage
- Full D/I Container and Configuration Injection
- Declarative HTTP Client
- https://micronaut.io

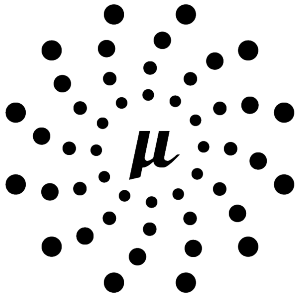


MICRONAUT®

Micronaut @Singleton Bean (in Groovy!)

src/main/groovy/eu/vies/VatService.groovy

```groovy
package eu.vies

import groovy.transform.*
import wslite.soap.*
import javax.inject.Singleton

@CompileStatic
@Singleton
class VatService {
    String url = 'http://ec.europa.eu/taxation_customs/vies/services/checkVatService'
    SOAPClient client = new SOAPClient("${url}.wsdl")

    @CompileDynamic
    Boolean validateVat(String memberStateCode, String vatNumberCode) {
        SOAPResponse response = client.send(SOAPAction: url) {
            body('xmlns': 'urn:ec.europa.eu:taxud:vies:services:checkVat:types') {
                checkVat {
                    countryCode(memberStateCode)
                    vatNumber(vatNumberCode)
                }
            }
        }
        response.checkVatResponse.valid.text() == 'true'
    }
}
```



Micronaut Bean wired into Grails Controller

grails-app/controllers/example/grails/ViesController.groovy

```groovy
package example.grails

import eu.vies.VatService
import groovy.transform.CompileStatic
import org.springframework.beans.factory.annotation.Autowired

@CompileStatic
class ViesController {

    @Autowired
    VatService vatService

    def valid(String memberStateCode, String vatNumberCode) {
        render vatService.validateVat(memberStateCode, vatNumberCode)
    }
}
```

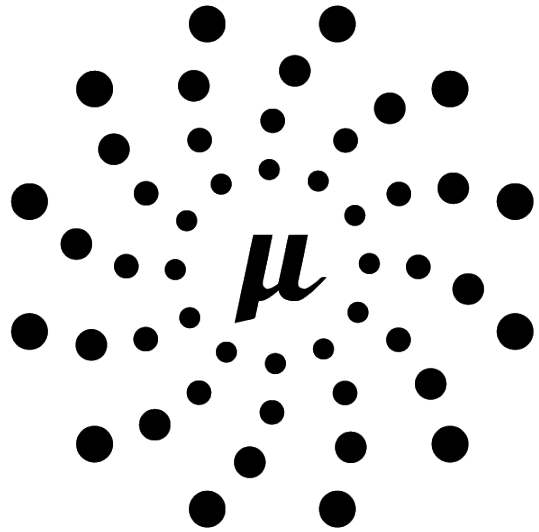
Micronaut HTTP Client

- Declarative client interfaces
- HTTP handling logic computed at compilation time
- Support for headers, data binding, authentication, multi-part uploads, file streams, and more!

```
import io.micronaut.http.MediaType
import io.micronaut.http.annotation


@Client("/hello") //or absolute URL or service ID
interface HelloClient {

    @Get(produces = MediaType.TEXT_PLAIN)
    String hello()
}
```

Micronaut Configurations with Grails

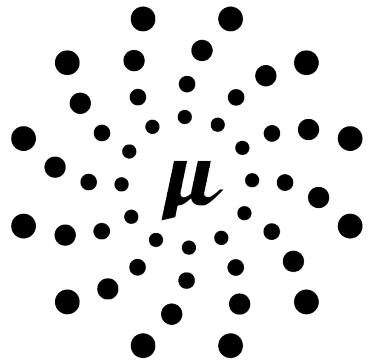
- Micronaut includes many **configuration** libraries that wrap useful tools and frameworks
- Some of these can be used with Grails, including:
  - Micronaut RabbitMQ
  - Micronaut Kafka
  - Micronaut RSS *



MICRONAUT®

Micronaut RabbitMQ

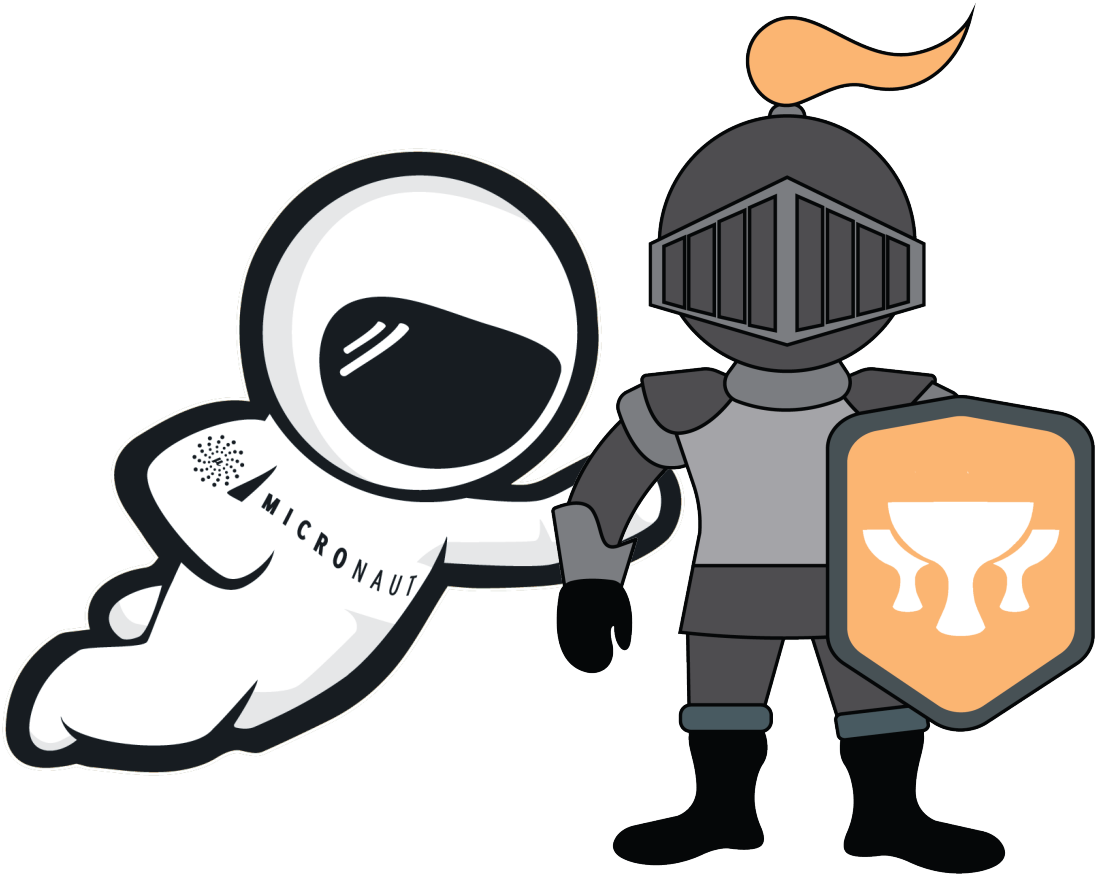
- RabbitMQ is a messaging framework
- Micronaut RabbitMQ offers a clean, declarative API for publishing and consuming RabbitMQ messages
- We can leverage Micronaut RabbitMQ to connect our Grails 4+ applications to a RabbitMQ queue



MICRONAUT®

RabbitMQ™

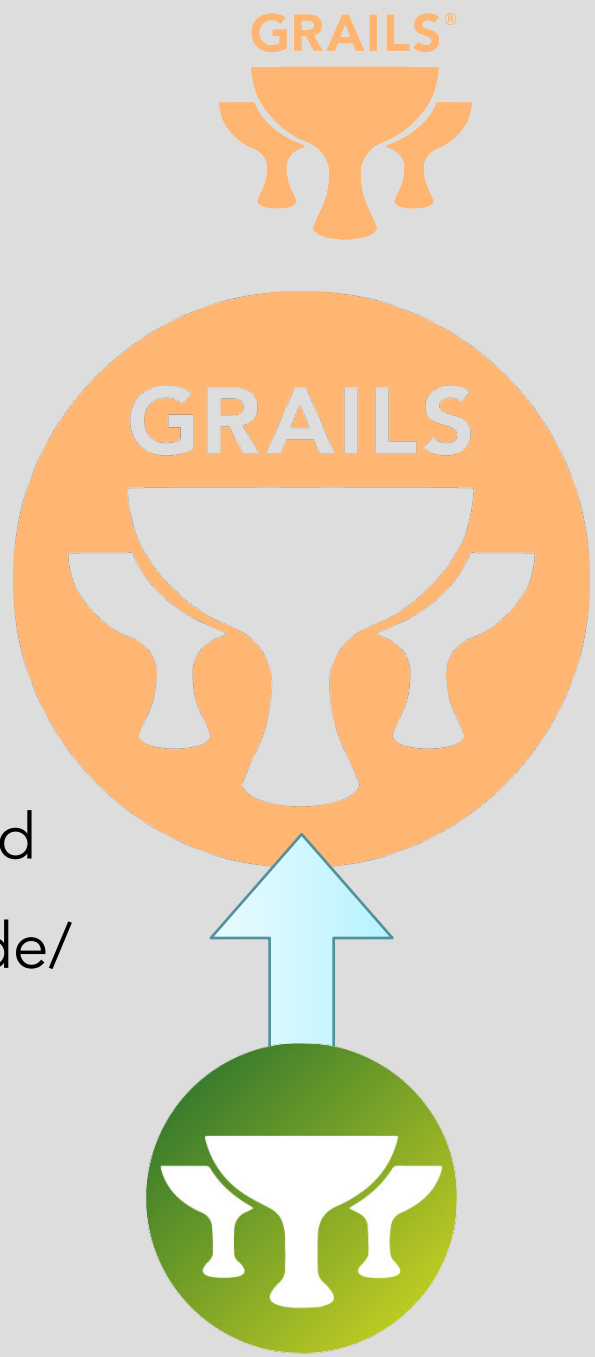
Workshop: Grails + Micronaut



https://github.com/grails-samples/grails-micronaut-rabbitmq

Upgrading to Grails 4+

- Grails 3.x → 4 is a fairly smooth upgrade - really!
- Mostly updates of underlying frameworks (Spring, Hibernate, etc)
- Deprecated classes in 3.x have been removed
- Most plugins work in 3.x and 4.x
- Many Spring configuration properties have changed
- Update notes: https://grails.github.io/grails-upgrade/latest/guide/index.html#upgradingTo4x



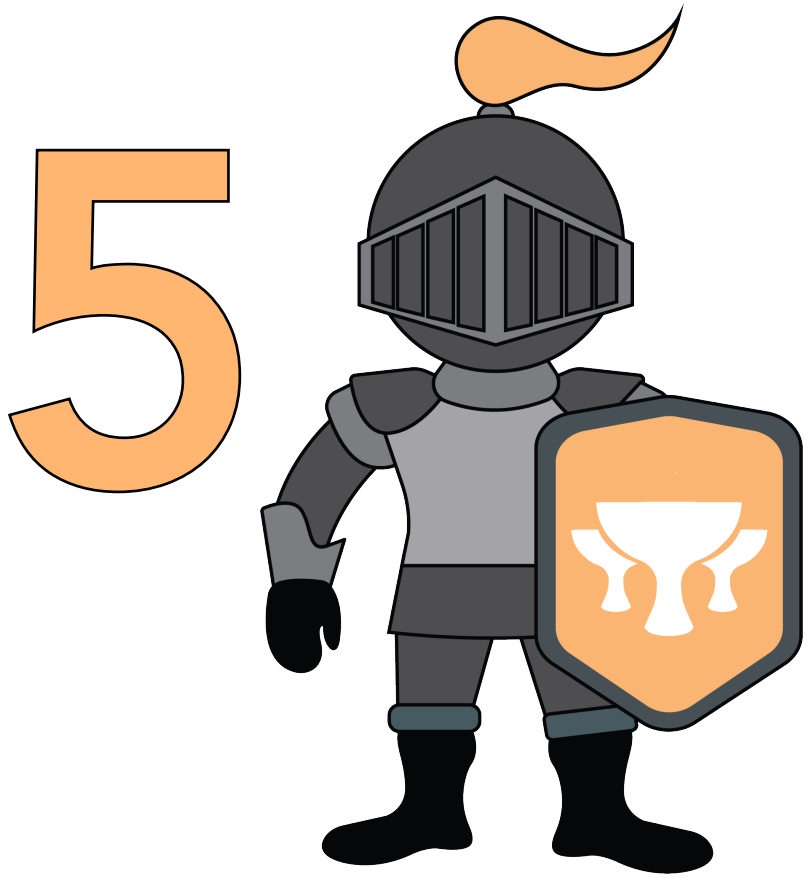
Grails Upgrade Resources

- Grails 4 Upgrade Webinar: https://grails.org/blog/2019-12-20.html
- ApacheCon Grails Upgrade Talk: https://www.youtube.com/watch?v=lmBCSAkWMlk
- GR8Conf Grails 2 → 3 Upgrade Talk: https://www.youtube.com/watch?v=IhehO9aM5bk

What's New in Grails 5.0?

- New Release Cadence
- Major updates to underlying frameworks:
  - Spring 5.3.9
  - Spring Boot 2.5.3
  - Apache Groovy 3
  - Hibernate 5.5
  - Gradle 6.9
  - Support for Java 15
- Not a breaking upgrade (for most apps)!

Workshop: Upgrade to Grails 5



Introduction to Groovy 3.0

- Major upgrade to the Groovy Programming Language!
- New "Parrot" Parser - improved Java compatibility and new syntactic sugar
- Support for Java Lambdas
- Method references: `Person::toString`
- New "identity" operator ===
- New GDK methods (`average()`, `shuffle()`, `LocalDate.minus()`, etc)
- https://groovy-lang.org/releasenotes/groovy-3.0.html

Workshop: Groovy 3.0 Demo!

How Can I Learn More?

- Grails Website: https://grails.org
- Grails Guides: https://guides.grails.org
- Grails Plugins: https://plugins.grails.org
- Follow Grails Framework on Twitter: https://twitter.com/grailsframework
- OCI Training: https://objectcomputing.com/services/training/catalog/grails

Grails/Groovy Community Resources

- Grails Community Slack channel: http://grails.slack.com
- Groovy Community Slack channel: https://groovycommunity.com
- Mr Haki's Blog: https://www.mrhaki.com
- GroovyBlogs: http://www.groovyblogs.org
- Szymon Stepniak's YouTube Channel: https://www.youtube.com/c/eprintstacktrace/
- GR8Conf EU (postponed): https://gr8conf.eu

Q & A

Grails Community Information

grails.org/blog

grails.org/foundation

github.com/grails

grails.signup.team

grails.org/support

guides.grails.org

grails.org/documentation

plugins.grails.org

objectcomputing.com/products/grails/solutions

@grailsframework

OCI | Contact Information

Zachary Klein

Senior Software Engineer

kleinz@objectcomputing.com

Jen Wiese

Community Engagement Manager

wiesej@objectcomputing.com

(314) 579-0066

grails.org

Grails Plugin System

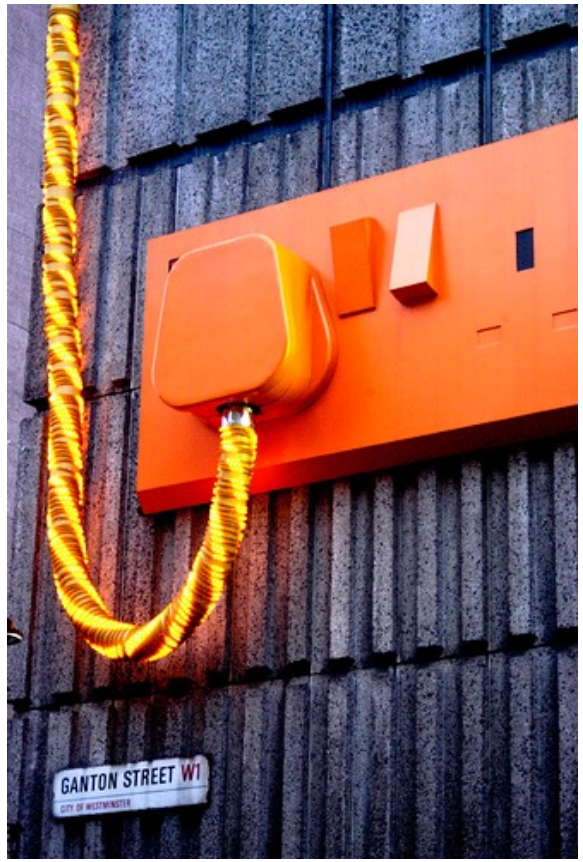
- Modular Application Development made easy!
- Grails is designed to wire together different libraries and make them easy to use
- In this sense it can be seen as a "platform for runtime configuration"
- De-coupling those components was hard without a well defined system

The Problem

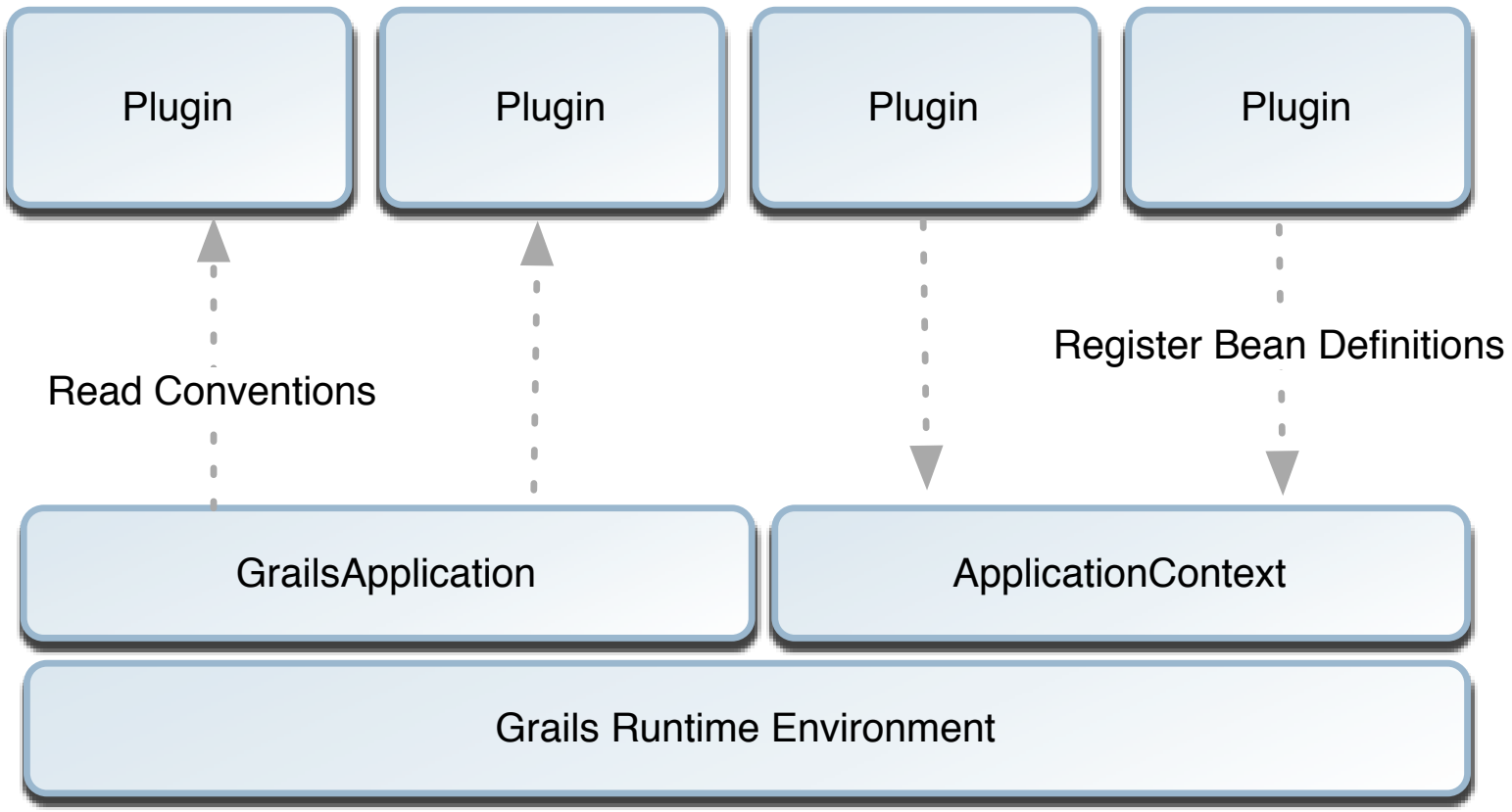
- Grails build system had become a monster with over a 1000 lines of Ant XML
- Component wiring was too closely coupled to runtime config making it brittle and easy to break
- Auto-reloading was too closely couple to the Servlet filter, making it hard to maintain

The Extension Points

- The Build System
- Spring Application Context
- Dynamic method registration
- Auto Reloading
- Container Config ( )
- Adding new Artefact Types



Plug-in Architecture



Plug-in Goals

- The system should be embrace convention-based approaches and DRY
- All required extension points should be satisfied
- Plug-ins should be easy to distribute, install with zero configuration

What is a Plug-in?

- Just like a normal Grails project!
- The only difference is the presence of a `*GrailsPlugin.groovy` file
- Use `grails create-plugin` to create one!

What can a Plug-in do?

- Add new methods, constructors, properties etc. to any class at runtime
- Perform runtime Spring configuration
- Add new controllers, tag libraries etc.
- Etc…

DEMO

Creating a Plug-in

A Basic Plug-in


```
class LoggingGrailsPlugin extends Plugin {

    def version = 3.0

    def doWithDynamicMethods() {
        application.allClasses.each { c ->
            c.metaClass.getLog = {->
                LogFactory.getLog(c)
            }
        }
    }
}
```

Application Object

- The `GrailsApplication` object, available with the "application" variable, can be inspected for convention information
- Has useful methods including:
  - `get*Classes` - Retrieves all `GrailsClass` instances for a particular artefact eg. getControllerClasses()
  - `add*Class(Class cls)` - Adds a new GrailsClass for the specified class eg. addControllerClass(myClass)
  - `get*Class(String name)` - Retrieves a GrailsClass for a given name eg getControllerClass(name)

Grails Artefacts

- An artefact is some resource that fulfills a convention such as controllers, services etc.
- They are represented by the `GrailsClass` interface
- A Full API for the different artefacts in Grails can be found at:
  - http://grails.org/doc/1.0.x/api/org/codehaus/groovy/grails/commons/package-summary.html
- New artefacts can be added via the Artefact API:
  - http://grails.org/Developer+-+Artefact+API

The GrailsClass Interface

- Grails artefacts that are Groovy sources are represented by the `GrailsClass` interface:
  - `getPropertyValue(String name)` - Get a property value
  - `newInstance()` - Create a new instance
  - `getName()` - Get the logical name
  - `getFullName()` - Get full name including package
  - `getShortName()` - Class name without package
  - `getPropertyName()` - Class name as property name
  - `getClazz()` - Get the actual java.lang.Class

Adding Basic Artefacts

- A Plug-in can add new tag libraries, controllers and services simply by creating them in the plug-in project
- Since a plug-in project is just like any project you can run and debug a plug-in in its own project before distributing it
- Once you're done package and distribute it!

Overview of Plug-in Closures

- A Plug-in can specify multiple callbacks to manipulate Grails:
  - **doWithSpring** - Participate in Spring configuration
  - **doWithApplicationContext** - Post ApplicationContext initialization activities
  - **doWithDynamicMethods** - Add methods!
  - **onChange** - Participate in reload events

Configuring Spring


```
// Configuring Spring
class SomeGrailsPlugin extends Plugin {
  def version = 1.0

  def doWithSpring() {
    { ->
     jcrRepository(RepositoryFactoryBean) {
        configuration =
             "classpath:repository.xml"
        homeDir = "/repo"
     }
    }
  }
}
```

Plugging in Dynamic Methods

- We know the conventions, but how do we easily take advantage of them to modify types?
- Grails users needed an easy way to add dynamic methods, properties, constructors etc.
- No out-of-the box solution in Groovy so we created **ExpandoMetaClass (EMC)**

Example


```
def doWithDynamicMethods() {
    application
      .allClasses
      .findAll { it.name.endsWith("Codec")
      .each {clz ->
        Object
         .metaClass
         ."encodeAs${clz.name-'Codec'}" = {
             clz.newInstance().encode(delegate)
        }
    }
}
```

Taken from Grails core, this plug-in finds all classes that end with "Codec" and dynamically creates "encodeAsFoo" methods!

The "delegate" variable is equivalent to "this" in regular

Runtime Extension Methods

- Allow A Plugin To Add Methods To Existing Classes

```
package demo

class StringExtension {

    static String getPigLatin(String s) {
        s[1..-1] + s[0] + 'ay'
    }
}
```

Runtime Extension Methods

- Descriptor Required In Order To Allow The Groovy Runtime To Discover The Extension Methods

```
# META-INF/services/org.codehaus.groovy.runtime.ExtensionModule
moduleName=string-magic
moduleVersion=1.0
extensionClasses=demo.StringExtension
```

Runtime Extension Methods

- Useful For Adding Methods To Existing Classes
- Methods Are Added At Runtime
  - no compile time dependency required
- Method Names Have To Be Static
  - ExpandoMetaClass supports adding dynamic method names

@Enhances

- The Enhances Annotation May Be Applied To A Trait
- The Trait Will Be Applied To Specified Artifacts

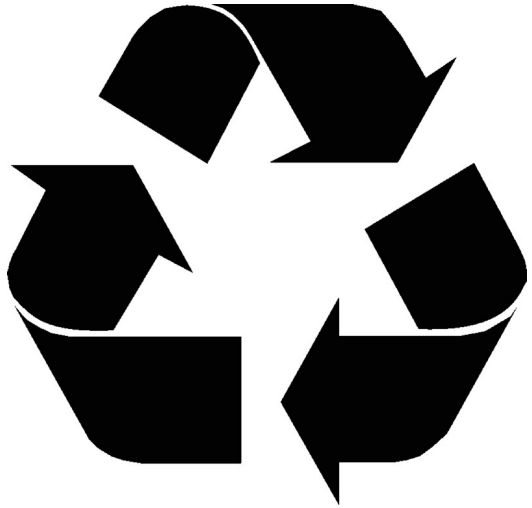
```groovy
// src/main/groovy/demo/MagicService.groovy
package demo

import grails.artefact.Enhances

@Enhances('Service')
trait MagicService {

    // this method will be added
    // to all service artifacts...
    def getMagicNumber() {
        42
    }
}
```

@Enhances

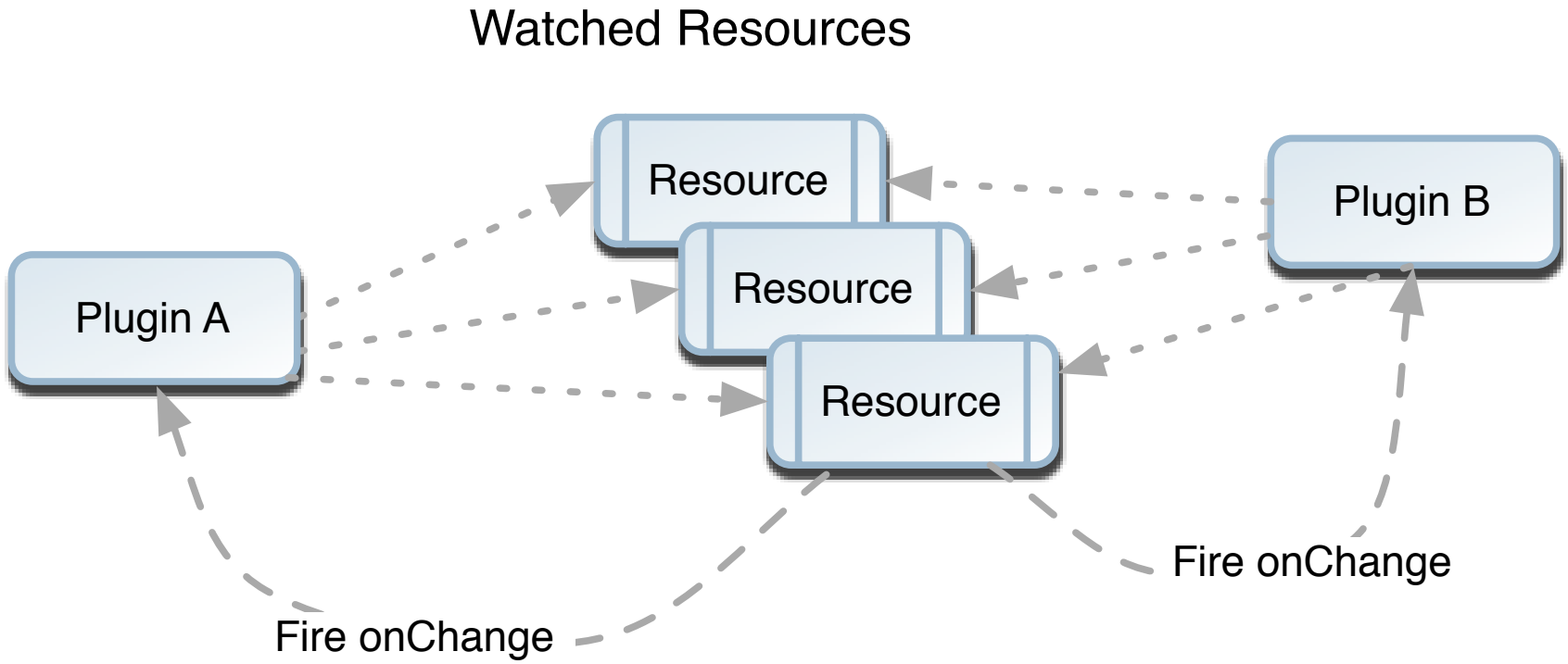
- Useful For Adding Behavior To Grails Artifacts
- May Not Be Used To Add Methods To Any Arbitrary Class
- The Trait Is Added To The Target Artifact Types At Compile Time
  - can result in artifacts provided by plugins having different sets of behavior

Reload Events

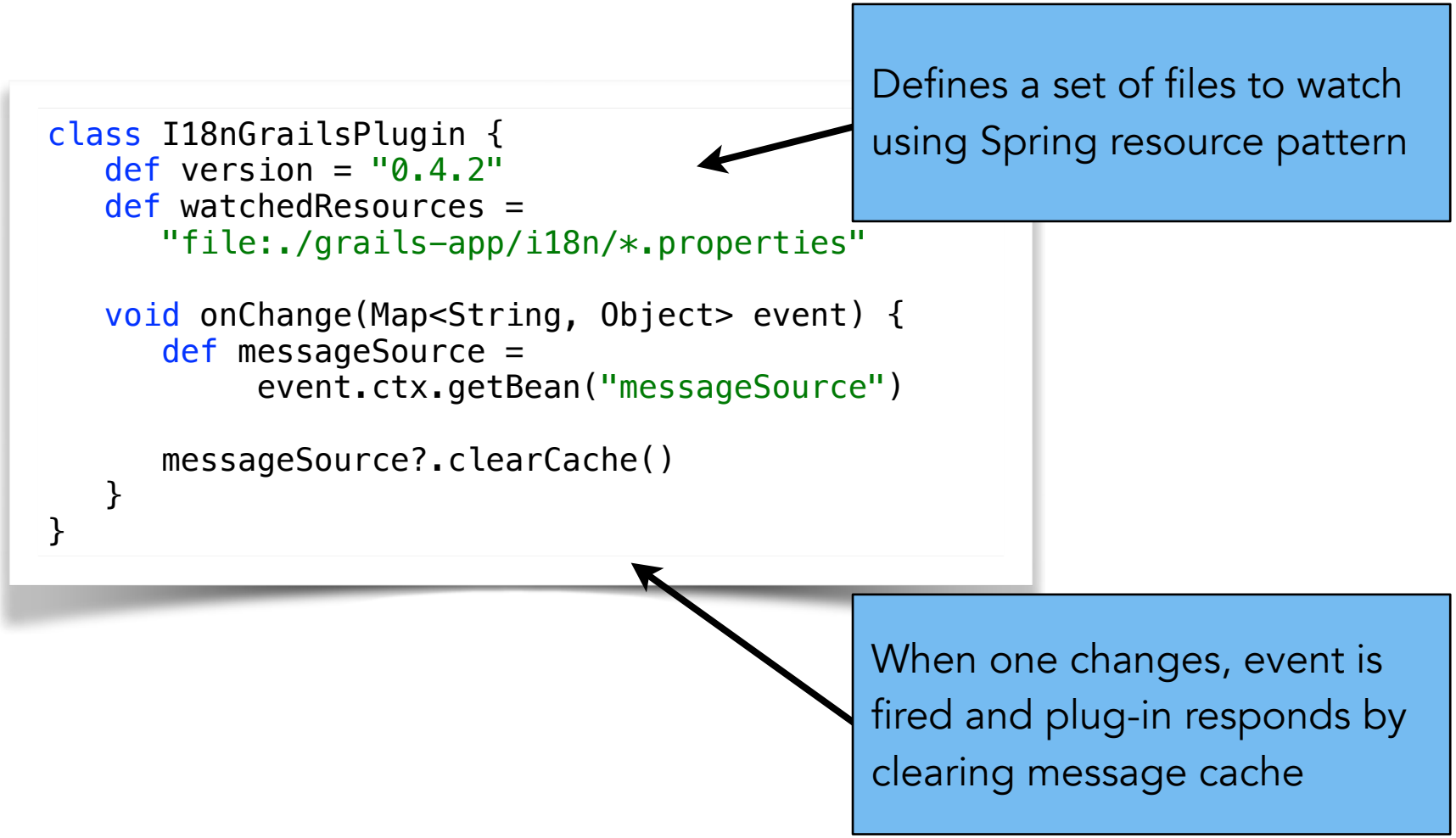
- Grails applications must be reloadable during development
- Reloading can mean many things, plug-ins are responsible
- Plug-ins can defined **watchedResources** that fire **onChange** events when modified



Watching Resources



Example Reloading Plug-in



The Event Object

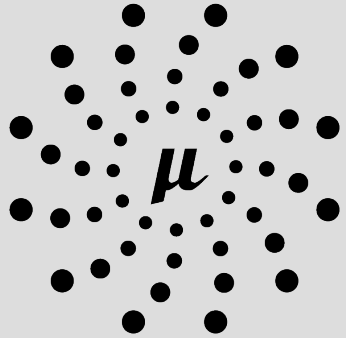
- **event.source** - The source of the change which is either a Spring Resource or a java.lang.Class if the class was reloaded
- **event.ctx** - The Spring ApplicationContext
- **event.application** - A GrailsApplication
- **event.manager** - A GrailsPluginManager

Summary

- The Grails plug-in system is flexible and powerful
- Plug-ins can do just about anything from adding new methods to configuring Spring beans
- Grails has a growing plug-in community

Micronaut Configurations

- Micronaut is the “parent context” in Grails 4+
- Micronaut Configurations are libraries that can create beans, add/read configuration, perform startup logic, fire off events, and more
- Compatible with Micronaut apps as well as Grails apps!



MICRONAUT®

Example: From a Plugin to a Configuration


```
├── build.gradle
├── gradle
|   └── wrapper
|       ├── gradle-wrapper.jar
|       └── gradle-wrapper.properties
├── gradle.properties
├── gradlew
├── gradlew.bat
├── grails-app
|   ├── conf
|   |   ├── application.yml
|   |   └── logback.groovy
|   ├── services
|   |   └── eu
|   |       └── vies
|   |           ├── VatService.groovy
|   ├── init
|   |   └── eu
|   |       └── vies
|   |           ├── Application.groovy
|   |           └── BootStrap.groovy
├── grails-wrapper.jar
├── grailsw
├── grailsw.bat
```

Grails Service provided by Plugin

grails-plugin: grails-app/services/eu/vies/VatService.groovy

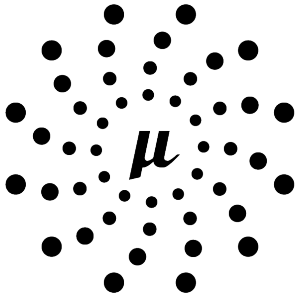
```groovy
package eu.vies

import groovy.transform.CompileDynamic
import groovy.transform.CompileStatic
import wslite.soap.SOAPClient
import wslite.soap.SOAPResponse

@CompileStatic
class VatService {
    String url = 'http://ec.europa.eu/taxation_customs/vies/services/checkVatService'
    SOAPClient client = new SOAPClient("${url}.wsdl")

    @CompileDynamic
    Boolean validateVat(String memberStateCode, String vatNumberCode) {
        SOAPResponse response = client.send(SOAPAction: url) {
            body('xmlns': 'urn:ec.europa.eu:taxud:vies:services:checkVat:types') {
                checkVat {
                    countryCode(memberStateCode)
                    vatNumber(vatNumberCode)
                }
            }
        }
        response.checkVatResponse.valid.text() == 'true'
    }
}
```

Creating a Micronaut Configuration (library)

Minimal Micronaut build

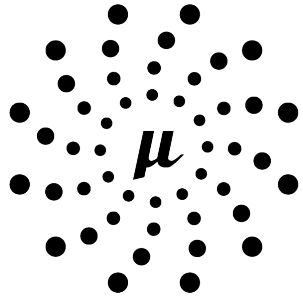
lib: build.gradle

```
plugins {
    id 'groovy'
}

repositories {
    jcenter()
}

dependencies {
    compileOnly "io.micronaut:micronaut-inject-groovy:$micronautVersion"
    compile 'org.codehaus.groovy:groovy-xml:2.5.7'
    compile 'com.github.groovy-wslite:groovy-wslite:1.1.2'
    testCompile("org.spockframework:spock-core:${spockVersion}") {
        exclude module: 'groovy-all'
    }
}
```



Micronaut Implementation of Grails Service

lib: src/main/groovy/eu/vies/VatService.groovy



```groovy
package eu.vies

import groovy.transform.CompileDynamic
import groovy.transform.CompileStatic
import wslite.soap.SOAPClient
import wslite.soap.SOAPResponse
import javax.inject.Singleton

@CompileStatic
@Singleton
class VatService {
    String url = 'http://ec.europa.eu/taxation_customs/vies/services/checkVatService'
    SOAPClient client = new SOAPClient("${url}.wsdl")

    @CompileDynamic
    Boolean validateVat(String memberStateCode, String vatNumberCode) {
        SOAPResponse response = client.send(SOAPAction: url) {
            body('xmlns': 'urn:ec.europa.eu:taxud:vies:services:checkVat:types') {
                checkVat {
                    countryCode(memberStateCode)
                    vatNumber(vatNumberCode)
                }
            }
        }
        response.checkVatResponse.valid.text() == 'true'
    }
}
```

Micronaut Bean wired into Micronaut Controller

micronaut: src/main/example/micronaut/ViesController.groovy

```groovy
package example.micronaut

import eu.vies.VatService
import groovy.transform.CompileStatic
import io.micronaut.http.annotation.Controller
import io.micronaut.http.annotation.Get
import javax.inject.Inject

@CompileStatic
@Controller('/vies')
class ViesController {

    @Inject
    VatService vatService

    @Get('/valid')
    Boolean valid(String memberStateCode, String vatNumberCode) {
        vatService.validateVat(memberStateCode, vatNumberCode)
    }
}
```

Micronaut Bean wired into Grails Controller

grails: grails-app/controllers/example/grails/ViesController.groovy

```groovy
package example.grails

import eu.vies.VatService
import groovy.transform.CompileStatic
import org.springframework.beans.factory.annotation.Autowired

@CompileStatic
class ViesController {

    @Autowired
    VatService vatService

    def valid(String memberStateCode, String vatNumberCode) {
        render vatService.validateVat(memberStateCode, vatNumberCode)
    }
}
```

Micronaut Bean wired into Spring Boot Controller

springboot: src/main/example/micronaut/ViesController.groovy

```groovy
package example.springboot

import org.springframework.boot.SpringApplication
import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.context.annotation.ComponentScan

@ComponentScan(basePackages = "eu.vies",
        basePackageClasses = SpringbootApplication.class)
@SpringBootApplication
class SpringbootApplication {

    static void main(String[] args) {
        SpringApplication.run(SpringbootApplication, args)
    }

}
```

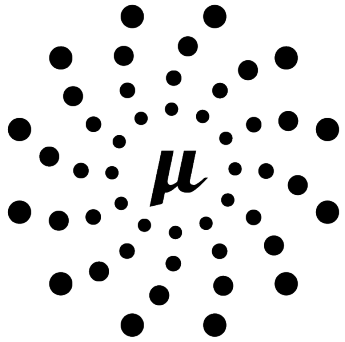
Micronaut Bean wired into Spring Boot Controller

springboot: src/main/example/micronaut/ViesController.groovy

```groovy
package example.springboot

import eu.vies.VatService
import groovy.transform.CompileStatic
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RequestParam
import org.springframework.web.bind.annotation.RestController

@CompileStatic
@RequestMapping('/vies')
@RestController
class ViesController {

    @Autowired
    VatService vatService

    @GetMapping('/valid')
    Boolean valid(@RequestParam(required = true) String memberStateCode,
                  @RequestParam(required = true) String vatNumberCode) {
        vatService.validateVat(memberStateCode, vatNumberCode)
    }
}
```

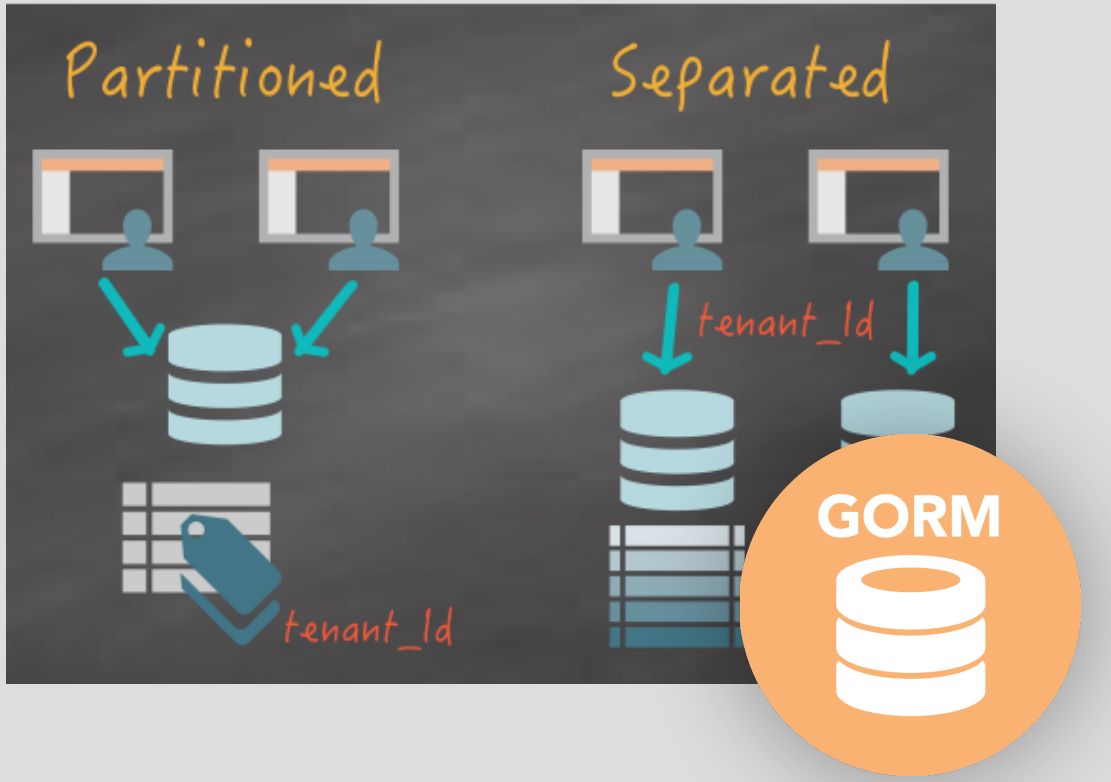
Micronaut Configurations with Grails

- Micronaut is the "parent context" in Grails 4+
- Micronaut Configurations can provide beans that will be available to be injected into Grails artifacts
- Can also bundle configuration and make it available as DI-aware `@ConfigurationProperties` beans
- Factories can wrap third-party SDKs/APIs
- Configurations can be shared among Grails, Spring Boot, and Micronaut applications
- Conditional beans, bean replacement, etc…
- Possibilities are endless!



GORM Multi-Tenancy

- Supported since GORM 6
- Multiple modes supported:
  - Database per tenant
  - Schema per tenant
  - Discriminator column (shared schema)



Database Per Tenant

- Database Per Tenant
- Most Secure Isolation
- Builds On GORMS Multiple Data Source Support

Defining Multiple Data Sources


```
# grails-app/conf/application.yml
dataSource:
    dbCreate: update
    url: jdbc:h2:./primaryDB
dataSources:
    automobiles:
        dbCreate: update
        url: jdbc:h2:./automobileDB

 ...
```

Mapping To A Data Source


```
// grails-app/domain/demo/Car.groovy
package demo

class Car {
    String make
    String model

    static mapping = {
        datasource 'automobiles'
    }
}
```

Mapping To Multiple Data Sources


```
// grails-app/domain/demo/Car.groovy
package demo

class Car {
    String make
    String model

    static mapping = {
        datasource (['automobiles', 'products'])
    }
}
```

Namespacing Data Sources

- Namespaced Queries For Domains Mapped To Multiple Data Sources

```
// the first data sources listed in the
// mapping block is the default...
def carFromAutomobilesDb = Car.get(2112)

def carFromProductsDb = Car.products.get(42)
```

Database Per Tenant


```
grails:
  gorm:
    multiTenancy:
      mode: DATABASE
      tenantResolverClass:
org.grails.datastore.mapping.multitenancy.web.SubDomainTenantResolver
dataSource:
    dbCreate: create-drop
    url: jdbc:h2:mem:books
dataSources:
    moreBooks:
        url: jdbc:h2:mem:moreBooks
    evenMoreBooks:
        url: jdbc:h2:mem:evenMoreBooks
```

Tenant Resolvers

SessionTenantResolver	Resolves the tenant id from the HTTP session using an attribute called `gorm.tenantId`
CookieTenantResolver	Resolves the tenant id from the HTTP cookie using an attribute called `gorm.tenantId`
SubDomainTenantResolver	Resolves the tenant id from the current sub-domain. For example if the subdomain is `foo.mycompany.com`, the tenant id would be `foo`
SystemPropertyTenantResolver	Resolves the tenant id from a system property called `gorm.tenantId`. Mainly useful for testing

Multi Tenant Domain Classes


```
class Book implements MultiTenant<Book> {
    String title
}

import static grails.gorm.multitenancy.Tenants.*

List<Book> books = withCurrent {
    Book.list()
}


List<Book> books = withId("moreBooks") {
    Book.list()
}
```

Mapping The Discriminator


```
class Book implements MultiTenant<Book> {
    String publisher
    String title

    static mapping = {
        tenantId name:'publisher'
    }
}
```

Multi Tenancy Grails Guides

- **Discriminator per Tenant Guide**: https://guides.grails.org/discriminator-per-tenant/guide/
- **Database per Tenant Guide**: https://guides.grails.org/database-per-tenant/guide/
- **Custom JWT Tenant Resolver Guide**: https://guides.grails.org/grails-custom-security-tenant-resolver/guide/
- **Dynamic Datasources Multi-tenancy Guide**: https://guides.grails.org/grails-dynamic-multiple-datasources/guide/