REAL WORLD BOOT-UP SEQUENCES

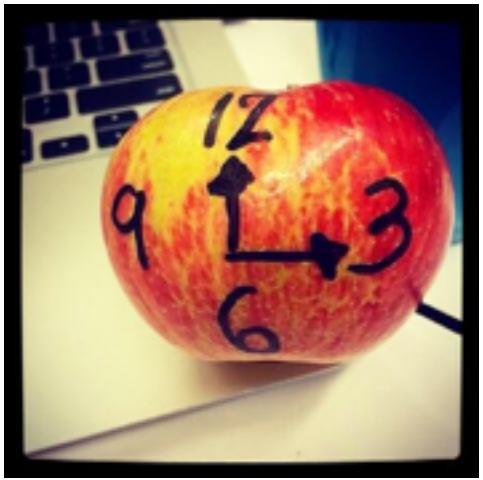
By Zach Lendon
@zachlendon

springone 2GX


How a Journey to Spring Boot is helping save our enterprise brownfield

Code examples/links/slides

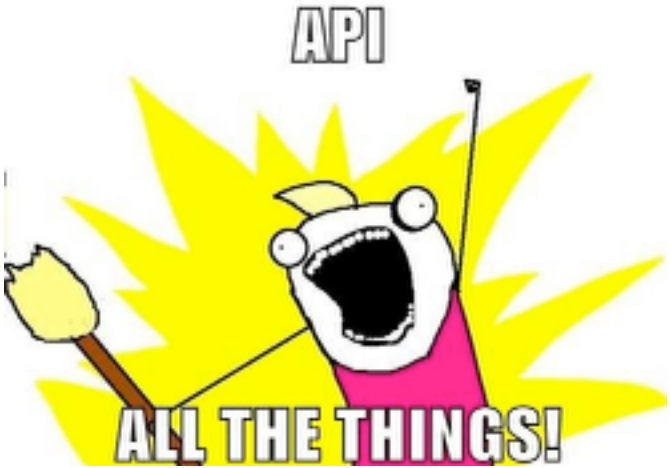
https://github.com/zachlendon/SpringOne2GX-2014
Or - contact me! More details/help available



Requisite Josh Long (@starbuxman) Quote

"Any application of sufficient complexity eventually sprouts a REST API"

- Josh Long, Building 'Bootiful' Applications with Spring Boot



“We need a ‘RESTful’ API”

- Large Spring Framework “Web Application” used for nearly all facets of hotel-related functionality you can do online
- However, rise in non-web applications, specifically mobile, required a new way to access functionality built into the existing web application

Enterprise Projects have “unique” constraints

- Re-use large, diverse set of existing ‘Spring’ services wherever possible, many of which are wired using - dare I say it - XML.
- Use JAX-RS
  - Re-use existing JAXB definitions, custom marshallers, validators, etc
- Any architectural changes in direction need to be gradual
- Approachable technology solutions for a diverse team with varying skill-sets, OS’s, etc.

Some Software Architecture Goals

- Standardize on, by using and/or extend, “best-of-breed” solutions around ancillary API functionality (logging, metrics, etc.) where possible instead of “home-grown” solutions
- Continue to support present-day devops and management comfort with WAR deployment while changing underlying structure to support future-day potential microservice-based dynamic build and deployment options
- Simplify
- Provide Efficient, quality test coverage

Project Technology Implementation

Key Software Technologies Introduced / Explored

Technologies	*Why*
Gradle	Build flexibility
Groovy / Spock	Test Efficiency
Spring + Jersey	Known core technology foundation Ultimate flexibility in adding future “pieces” Still have flexibility to use Dropwizard
dropwizard-spring (hmsonline)	Clean Dropwizard + Spring integration
Spring Boot	Known, opinionated, simpler core tech foundation Ultimate flexibility in adding future “pieces” Dropwizard similarities yet more

Spring + Jersey

- Configure Jersey ResourceConfig
  - Register your Jersey JAX-RS resources
  - Register RequestContextFilter
  - Register other Jersey Components
- Configure JerseyServlet
  - @Configuration or web.xml
- Setup Jersey JAX-RS Resources
  - Bean dependencies
    - @Component and auto-wiring (@Autowire)
    - JSR-330 and @Inject

Spring + Jersey : Technical Thoughts : Pros/Cons

- Register your own custom Jersey ObjectMapper to provide data serialization/deserialization hooks
- Spring and Jersey are Best-of-breed solutions that technically fit together nicely and easily
- Developer-friendly and test-friendly
  - A production-tested solution around for several years
- Not an opinionated or full-stack solution = pro and con
  - Lack of out-of-the-box related features to easily leverage
- Not lightweight (WAR only)
  - not micro-service ready

Dropwizard-Spring

- Open-sourced by Health Market Science (https://github.com/hmsonline/dropwizard-spring)
- Can configure Jersey resources as Spring @Component
- Can configure your Dropwizard components entirely with Spring through YAML configuration file
- Can extend project's SpringService to hook into Spring / Dropwizard "marrying" to add/modify behavior

Dropwizard-Spring : Technical Thoughts : Pros/Cons

- Use Gradle
  - Gradle Shadow over Maven Shadow
- Use swagger-jaxrs-groovydoclet for creating Swagger artifacts from Groovy doc
- JAR only deployment (out of box)
- Combining potential enterprise leeriness around Dropwizard with a custom solution to tie it with existing Spring infrastructure

Spring Boot + Jersey

- Manual or use Jersey auto configuration from https://github.com/dsyer/spring-boot-jersey
- Auto-configuration allows Spring Boot to find JerseyConfig @Component and setup Jersey for you
- Spring-boot-actuator provides management and monitoring features similar to features Dropwizard surrounds Jersey with
- @AopAutoConfiguration can simplify your AOP/AspectJ mess
- Embedded Servlet container (for JAR) vs still supporting WAR deployment
- Groovy & Spock friendly

Spring Boot : Technical Thoughts : Pros/Cons

- Integration of Best of Breed Technologies (Spring, Jersey) with strong up-and-coming candidate in Spring Boot, from a known entity already in many enterprises
- Deploy as JAR or WAR huge for enterprise devops
- Opinionated app foundation helps:
  - set repeatable patterns for how to approach build-out of API and related services
  - wrangle dependency tree, ease configuration overload/ meltdown, lessen upgrade burden pain

Conclusion

- Lots of options for creating RESTful services with Spring, even when limited to specific technology choices (for better or worse)
- Spring Boot's use cases are many
  - Help developers with app infrastructure and ancillary services so that they can have best tools available at their disposal for building custom solutions
- Not just for greenfield - Spring Boot can be your gateway drug to a more modular/micro-service future for your brownfield application(s) as well
- Don't be afraid to create your own Spring Boot configured components for maintainability of your project and use by other projects at your company (or by community if possible!)

Learn More. Stay Connected

Spring Boot is not just for greenfield!
Don't be afraid to gradually introduce in existing apps big and small

Micro Service Architecture with Spring Boot and Groovy
Thurs, 8:30 with Marco Vermeulen

@springcentral | spring.io/video