Spring Boot Microservices:
Java Development Made Slimmer and Quicker

Spring Boot and Microservices in a Nutshell

- Has the core, lightweight aspects of a Java container
- Along with Embedded Apache Tomcat 7
- Packaged in a single, shaded jar
- Offers ease of deployment, modularity

Spring Boot vs. Other Micorservice Frameworks

Cons

I've used Dropwizard, Bare bones, does not rely so heavily on "Spring Magic"
(for non-fans of Spring Magic, it's not half bad)

Pros

Spring has integrated many of its frameworks into Spring Boot
Spring Boot apps can be built using Kotlin, Grails, and React.
All can be built with Gradle as a build system.
Both Kotlin and Grails are Java Interoperable.
React can be integrated into a Java based service

Running a service is Easy

Here I'm using Java and Maven
The Key aspects:

I) The pom file (maven)
Note the parent pom
note that most of the dependencies have no version
countless frameworks are managed as a part of Spring Boot

II) Application.java (note about place in package and spring reliance on convention)
III) HelloNoFrillsController.java
IV) resources/application.yml (more on this later!)

How to Package it? How to Run it?

Packaging…

the package maven package goal from Spring Boot creates a shaded jar, including all needed dependencies. You can also name the output jar anything you like (see jar. filename and repackage in pom)

Running…

- In Intellij: Create a maven Run configuration and add the command:

  ```
  mvn spring-boot:run
  ```

- Similar in Eclipse (create a run configuration)
- You can also go to the directory containing the pom file and run the same maven command
- alternatively, assuming the shaded jar has been built, you can run the command:
  java -jar target/neofonie-poc-service.jar

Open your service!

localhost:8081/api/rest/helloNoFrills

Swagger

http://localhost:8081/api/swagger-ui.html
(Note: show HelloController.java with annotations)

Makes Black-box testing, rapid Development super easy

Error Handling

(NOTE: show HelloController.java with “exception” endpoints
and GlobalExceptionHandler.java)

Spring Boot and CRUD
(Note: I've used H2 in-file DB for persistence)

Repositories a simple matter of

- Creating an interface class
- Extending an existing Spring Repository interface:
  (CrudRepository and JpaRepository are two which are commonly used)
- Set in a model object class generic type
- Annotate the interface as a Repository
- Simply inject as needed using Spring annotation @Autowired
  (see CustomerRepository.java)

right away you get access to a basic set of commonly used CRUD methods.

Need Something Special? No Problem.
Just add method to your repository, noting Spring's rather intuitive JPA method creation syntax. For examples and an overview of this syntax, please see:

https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation

application.yml and Configuration

- bundled inside the resources is the default properties
- "But wait, I want to override these values...": simply a copy of the yml file at the same directory as the jar file and change only what you need!
- Can configure most aspects and components
- Can create custom Configuration objects (POJOs)
  (NOTE: see GlobalProperties.java, GlobalExceptionHandler.java, application.yml)
    - Annotated as Component and with path
- Global Configuration Objects
    - Annotate with Configuration
- Create and Activate Profiles
    - java -jar -Dspring.profiles.active=dev neofonie-poc-service.jar
- OR in the application.yml file

```
spring:
  profiles.active: dev
```

(NOTE: see SwaggerConfiguration.java)

What else can you do? Lots…

- Unit and Integration test support (examples included!)
- Spring Security (Basic Authentication examples included!)
- Frontend frameworks (JSP, AngularJS, Thymeleaf example included!)
- Quartz Jobs (coming soon)
- Asynchronous Services (coming soon)

Example App Repo:
https://bitbucket.org/ziad_neofonie/p4s-springboot-prototype